December 2003

Architecture-Based Systems Evaluation: Lessons Learned

Anna Griman
Universidad Simón Bolívar

M Perez
Universidad Simón Bolívar

Luis Mendoza
Universidad Simón Bolívar

K. Domínguez
Universidad Simón Bolívar

ARCHITECTURE-BASED SYSTEMS EVALUATION: LESSONS LEARNED

A. Grimán
Universidad Simón Bolívar
agriman@usb.ve

M. Pérez
Universidad Simón Bolívar
movalles@usb.ve

L. Mendoza
Universidad Simón Bolívar
lmendoza@usb.ve

K. Domínguez
Universidad Simón Bolívar
kdoming@usb.ve

Abstract

Information Systems are today highly complex and users are demanding ever higher quality levels. Therefore, it is useful to have a minimum understanding of the level of quality expected of those systems. Architecture is one of the components most likely to affect system quality; hence it must be evaluated at the early stages of development in order to guarantee quality attributes in the future system. The objective of this research is to present the lessons learned from applying two architectural evaluation methods to the same development process applied to a case. The methods used were the Software Architecture Design Method (Bosch 2000) and the Architecture Tradeoff Analysis Method - ATAM (Clements et al. 2002). The case study was a Knowledge Management System known as PROYECTOS DID KMS, which was applied to Research Projects conducted by Universidad Simón Bolívar. The mainly conclusion is that the major difference between both methods lies in their effectiveness, based on the dimension of the systems architecture to be evaluated. The Bosch's Method, combined with the Simulation Technique, is suitable for systems that handle large amounts of components, since this technique enables the behavior of the different quality attributes to be quantitatively observed by subjecting them to changes in a relatively short space of time. On the other hand, ATAM is appropriate for relatively small systems, where stakeholders can reach a consensus and the architect can also carefully and qualitatively compare each of the scenarios identified.

Keywords: Architecture-based evaluation, ATAM, software architecture design method, case study

Introduction

Information Systems grow more complex by the day and compliance with certain quality attributes is increasingly necessary. Guaranteeing their quality in terms of these attributes is no easy task. A quality attribute is defined as a property whose degree of satisfaction by a software system can be valued by its users (external attributes) and/or by its designers (internal attributes) (Meyer 1997). It is harder and more expensive to identify, correct or improve these attributes once the system has been completed. Software architecture is characterized by its ability to foster or penalize some of these quality attributes. This raises the need to evaluate it during an early stage of the development process, and involves conducting a cost-effectiveness evaluation. Knowledge Management Systems (KMS) are no exception. They also require the fulfillment of certain quality requirements (attributes). Many of these requirements can be promoted by the architecture.

For Bass et al. (1998), software architecture is the structure or structures of the system, composed by software components, the visible external properties of those components, and the relationship between them, with special emphasis on architecture being an abstraction of the system that suppresses the details of the components neither used nor affected. Bass et al. (1998) believe software architecture is important because it facilitates communication among stakeholders and helps in decision-making on design issues by defining restrictions involving implementation, identifying quality attributes, handling changes and using transferable and reusable models, facilitating therefore one first evaluation of the future system.

Whitten et al. (2002) propose Information System Architecture provides a unifying framework into which various people with different perspectives can organize and view the fundamental building blocks of information system. Kructhen (1999) also proposes that one system's architecture is perhaps the most important deliverable that can be used to manage these different viewpoints and thereby controls the iterative and incremental development of a system throughout its life cycle. To evaluate the architecture of the case study, the Bosch (2000) and Clements et al (2002) methods were selected for their robustness and for the easiness of finding specific documentation.

The main goal of this paper is to present the lessons learned from applying two architectural evaluation methods: the Software Architecture Design Method (Bosch 2000) and the Architecture Tradeoff Analysis Method – ATAM (Clements et al. 2002) applied to the same case study during the development process of a KMS for Research Projects conducted by Universidad Simón Bolívar (PROYECTOS DID KMS). All these benefits demonstrate the need to define and specify software architecture as a step prior to its construction and also, once designed, to evaluate it based on certain specific criteria to see at what stage all these possible benefits effectively became advantages for the developing team and the end user. It would be helpful to have a method for evaluating the architecture.

This article contains, after introduction, the research methodology, the main characteristics of the case study, and the evaluation methods applied. Next, the lessons learned from these evaluations are described, and lastly the conclusions and recommendations are presented.

Research Methodology

To carry out this investigation, the DESMET methodology was applied, which is concerned with the evaluation of methods or tools within a particular organization (Kitchenham et al. 1996). The term organization is meant to apply to a software development group in a particular company/division performing broadly similar tasks under similar conditions. In addition, the DESMET methodology can be used by academic institutions interested in experimental software engineering (Kitchenham et al. 1996).

This methodology was followed, because to DESMET evaluation exercise is comparative. That is, the researches assumes that there are several alternative ways of performing a software engineering task and them want to identify which of the alternatives is best in specific circumstances (Kitchenham et al. 1996).

As a result of DESMET application, the method obtained was Feature Analysis Case Study, a feature-based evaluation undertaken alter a method/tool has been used in a practice on a real project (Kitchenham et al. 1996). On this base, the comparison of the methods Software Architecture Design Method (Bosch 2000) and the Architecture Tradeoff Analysis Method - ATAM (Clements et al. 2002) was carried out applying both methods in the case study described in the next section, and evaluating them according to the following characteristic: Use of scenarios as an evaluation tool, Weighting of scenarios, Identification of quality attributes, Quality specification techniques, Evaluation inputs, Simulation technique, Participation by the stakeholders, Right choice of architecture, Costs, and Dimension of the Architecture.

Case Study: Proyectos DID KMS

Universidad Simón Bolívar (USB) intends to stimulate applications for financing research projects and also to promote tools to handle these projects once they have been approved. It further intends to foster clarity and precision in the formulation of projects to reduce initial rejections and guarantee successful projects. It was within this context that the initiative to develop a KMS to support knowledge management related to the USB's research projects arose (Domínguez 2001). The purpose of the KMS developed in this research is to encourage the professors at the USB to manage their research projects through a Web interface, thereby fostering collaborative work and facilitating information sharing. The benefits of PROYECTOS DID KMS will include the ability to capitalize on the knowledge generated by the research projects and keep it where it is accessible by everyone to so that information on specific areas can be sought quickly and easily. The objectives to be met are: to encourage people to apply for Research Project funding, as well as to foster the development of tools to facilitate the handling of these projects once they are approved. A further intention is to foster clarity and precision in the formulation of these projects in order to reduce initial rejections and be able to attain successful projects. All this must be done while supporting the processes that are characteristic of a KMS: capturing, generating, sharing and distributing knowledge. PROYECTOS DID KMS system enables stakeholders to capture, generate, share and distribute the knowledge handled at the USB, while enabling the university to capitalize on and store all this knowledge, giving it a competitive advantage over other organizations.

Quality Attributes

Following the session with the KMS stakeholders (developers, operators, end users, representative clients, system administrators, evaluation team, managers, etc.), architectural quality attributes were selected. These were: *Maintainability, Reliability, Security* and *Efficiency*. As can be seen below, all of these are critical for a KMS, in addition to being architectural in nature (Bass et al., 1998). Although focused on systems in general, Jacobson et al. (2000) state the following (applied perfectly to a KMS): "If we can be sure about anything, it is that any sizeable system will evolve. It will even evolve if it is still under development." This is possible thanks to the architecture, which is why *Maintainability* has to be considered one of the architectural quality characteristics to be taken into account.

As far as system *Reliability* is concerned (Bass et al., 1998), it is defined as the system's ability to remain operational over time and, like Ortega (2000), they point out that *Reliability* is related to fault tolerance and the time it takes to recover, both aspects being attainable through the architecture. Thus for KMS's domain, *Reliability* must also be considered a quality characteristic or attribute to be promoted by the architecture.

Although *Security*, according to Ortega et al. (2000), is part of the Functionality and, as shown above, the latter is not architectural. Bass et al. (1998) consider that *Security* is, as they say that prevention, detection and response to such effects involve architectural strategies that may require the existence of special components to solve it (Bass et al., 1998). A KMS in particular will not be taken as just another characteristic, but as a sub-characteristic of *Reliability*.

As regards *Efficiency*, Bass et al. (1998) relate it to the time required to respond to a particular stimulus (event) or the number of events processed in the same time interval. These authors also say *Efficiency* can be measured on the basis of the amount of information and communication between system components, which clearly is an architectural characteristic, since components can be implemented within the server layer that handles users' requests efficiently. Generally, because they capture, distribute, share and generate Knowledge, KMS have this type of information flow, therefore handling must take place as efficiently as possible. So, *Efficiency* must also be taken into account when it comes to guaranteeing product quality through the architecture as far as a KMS's domain is concerned.

Candidate Architecture

In order to specify the architecture under the Kruchten (1999) approach, an initial Class Diagram (Candidate Architecture) was proposed. It includes two of the Design Patterns proposed by Gamma et al. (1995), these being: *Chain of Responsibility* and *Observer***.** The design patterns are descriptions of communication by objects and classes that are personalized in order to remedy a problem in the general plan within a specific context (Gamma et al. 1995). Each design pattern has a particular object-oriented approach. It describes when to apply them and specifies whether they can be applied in view of another design restriction, its consequences and its tradeoffs. Figure 1 shows the Candidate Architecture with the design patterns used.

In the *Chain of Responsibility* Design Pattern, when a client issues a request, it propagates along the chain until it reaches a *ConcreteHandler* object, which takes responsibility for handling it. This leads to: reduced coupling, greater flexibility in the assignment of responsibilities and lack of guarantee in the receipt. Figure 2 shows the structure of the *Chain of Responsability* Design Pattern.

In the Candidate Architecture, the *Financial Handler*, *Activity Handler*, *Mail Handler* and *Document Handler* classes play a similar role to the *ConcreteHandler* object as these are levels in the Components Handler class that in turn would play the role of a *Handler* object. The direct advantages of applying this pattern are: better distribution and control of requests and more scalability.

In the *Observer* Design Pattern all the observers are notified when a change occurs in the status of stored objects. This had the following consequences: it reduces the coupling between Data and *Observer* Objects and it supports broadcast communication and unexpected changes. Figure 3 shows the structure of the *Observer* Design Pattern.

In the Candidate Architecture the data that will be subject to frequent changes are in the classes of the four main components, which is why the *Financial Handler*, *Activity Handler*, *Mail Handler* and *Document Handler* classes play a similar role to a *ConcreteSubject* object, whereas the *Component Handler* will play the role of a *Subject* object. Moreover, notification of updates makes sense if several professors are working on the same project. This can be known thanks to the sessions. Therefore, the *Session* class would play a similar role to a *ConcreteObserver* object, whereas the *Professor* class would play the role of an *Observer* object.

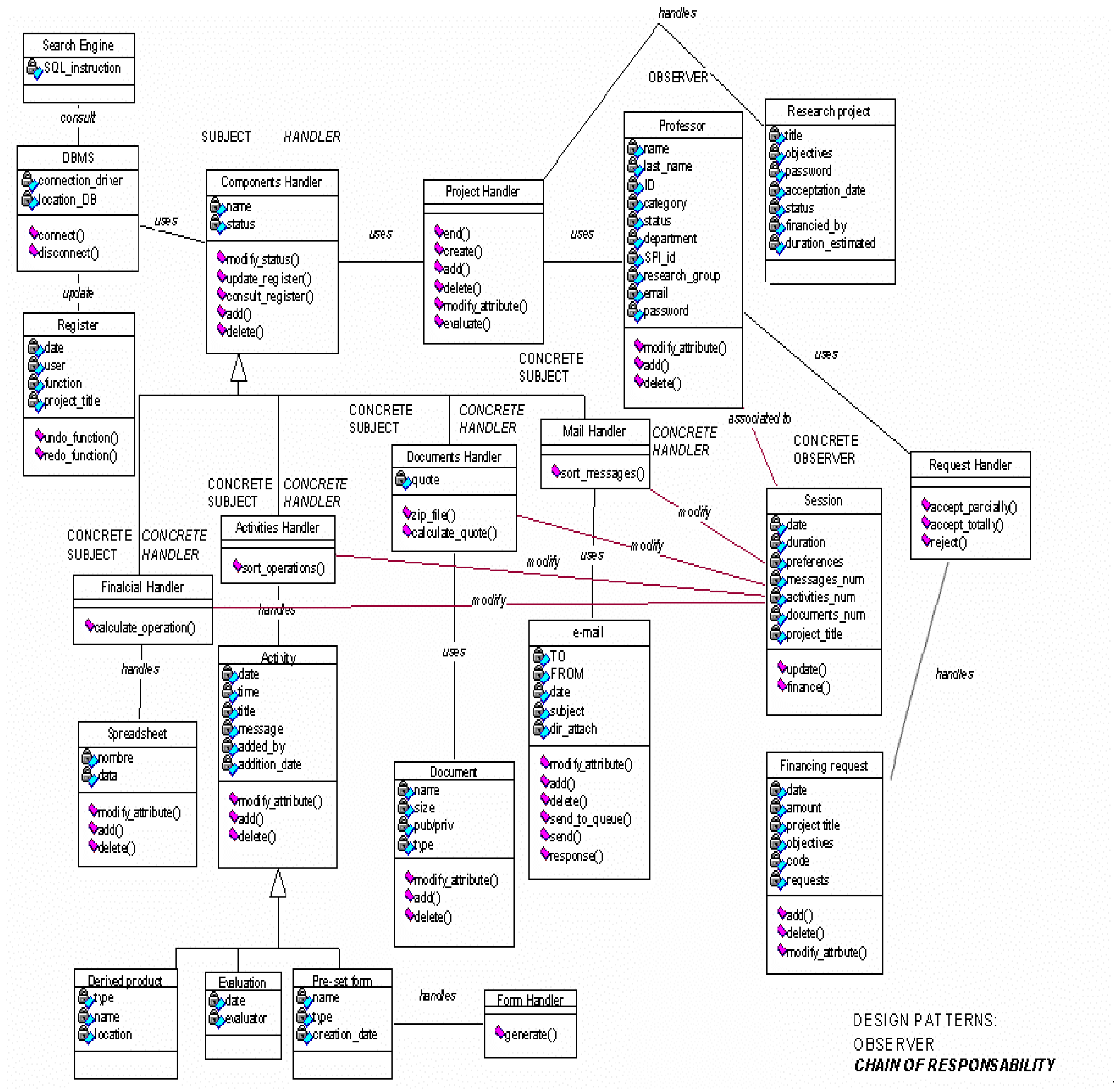

Figure 1. Class Diagram: Candidate Architecture

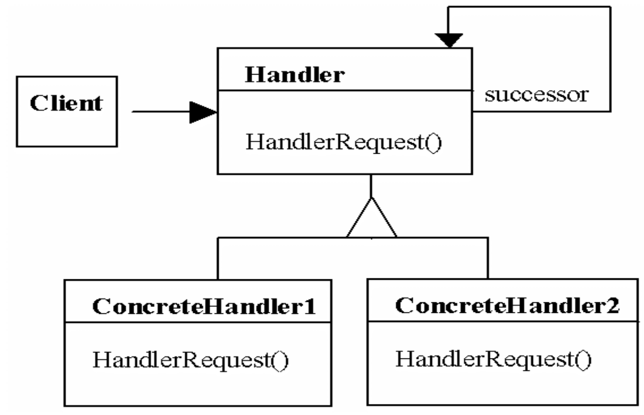

Figure 2. Structure of the *Chain of Responsibility* Pattern (Gamma et al. 1995)

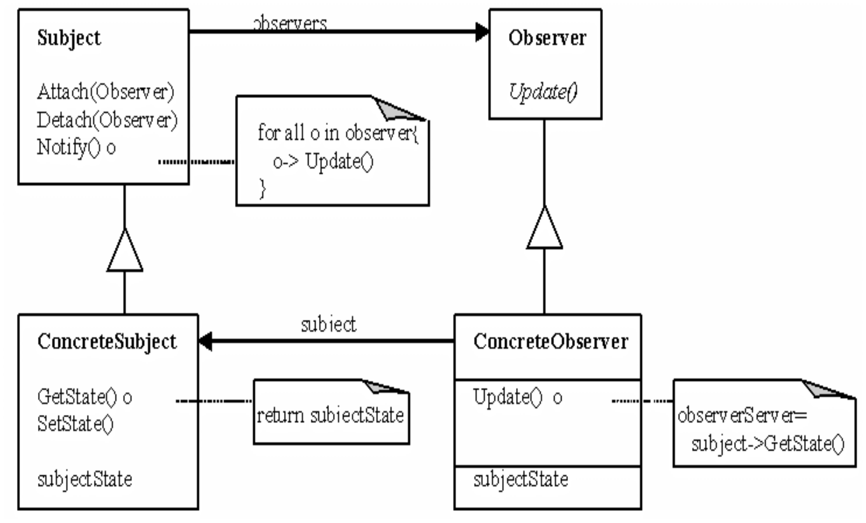

Figure 3. Structure of the *Observer* Pattern (Gamma et al. 1995)

After the system, its quality requirements and its candidate architecture are presented, the evaluation methods applied are briefly explained and learned lessons are presented.

Evaluation Methods, Techniques and Tools

As far as the Architectural Evaluation is concerned, Bosch (2000) affirms that the explicit evaluation of the quality requirements of systems architecture will minimize the risks involved in building a system that may fail, and will consequently reduce the cost of developing the system. This is why it was proposed that the architecture be evaluated to determine whether it is the best one for building a KMS. Therefore, and also for the purposes of this research, two independent evaluations were carried out: the first using the Software Architecture Design Method (Bosch 2000) and the second using the Architecture Tradeoff Analysis Method – ATAM (Clements et al. 2002). These methods were selected for their robustness and for the easiness of finding specific documentation. In each of the applications, lessons to be taken into account in future architectural evaluations were learned. Each of the methods used is briefly described below and its application on the case study is shown.

Software Architecture Design Method

This method focuses on the explicit evaluation and design of quality requirements in software architectures. According to Chirinos (2001), it is easier to evaluate quality attributes using the method proposed by Bosch (2000) for Systems Architecture design. Bosch (2000) also proposes that one of the following Evaluation techniques be chosen: Scenario based, Simulation based, Mathematical Model based, and Experience based. In keeping with the purpose of this research, the Simulation based evaluation will be used, in which the implementation of high-level Systems Architecture is used. The Bosch method works with an initial architecture which evaluates and transforms it in order to enhance it (see Figure 4), and affirms that there are four categories of transformation for architecture: *imposition of an architectural style*, *imposition of an architectural pattern*, *application of a design pattern and conversion of the quality requirements in terms of functionality*. This method is broken down into three fundamental phases: (a) functionality based architectural design; (b) evaluation of software architecture; and (c) transformation of software architecture. When the Simulation Technique is applied within the method proposed by Bosch, the following steps are followed:

Define and implement the context of the system: This activity seeks to identify the interfaces of the software architecture, their context, and to decide how these interfaces should behave within the context.

Implement the architectural components: The second phase establishes the choice of profile for each quality attribute associated with the candidate architecture; Table 1 shows the profile associated with the *Maintainability* attribute. Similarly, the corresponding profiles for *Efficiency* and *Reliability* were built.

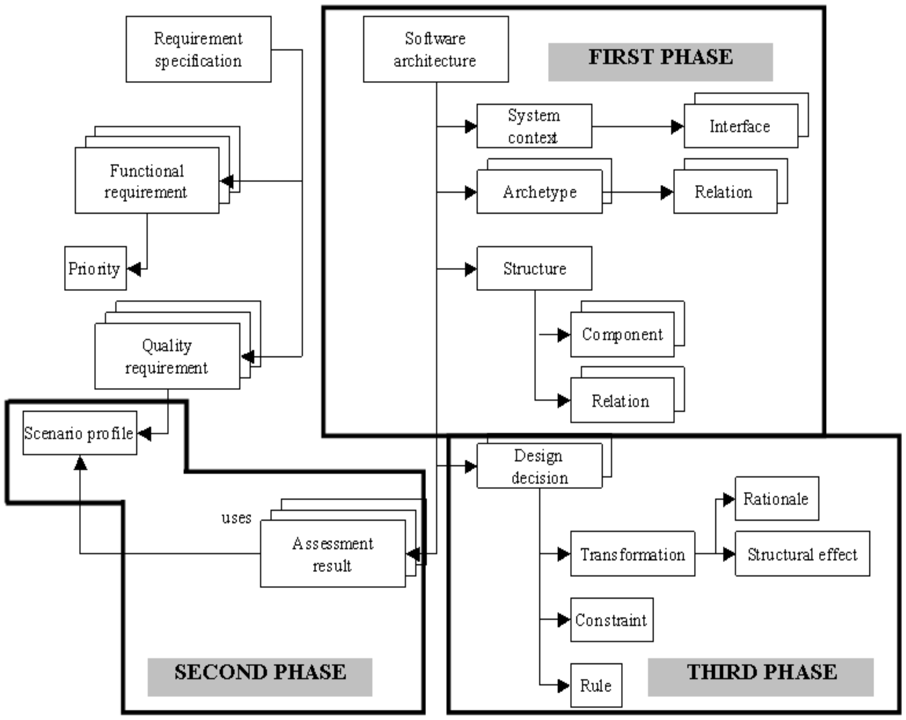

Figure 4. The Steps Taken by the Bosch Method

Table 1. Maintainability Scenario Profile

Category	Internal Attribute: Metrics	Maintainability Scenario	Weight	Relative weight
Management	Control Structure System: number of modules	S1: Add a new module meeting the system's requirerements to the different handlers	8	0,18
	Control Structure System: depth of the trace / Level of Coupling: number of interconnections	S2: Change a requirement in the Financial Tool (Financial Handler)	4	0,09
	Control Structure System: depth of the trace / Level of Coupling: number of interconnections	S3: Change a requirement in the Discussion List (Mail Handler)	6	0,14
	Control Structure System: depth of the trace / Level of Coupling: number of interconnections	S4: Change a requirement in the Document Repository (Document Handler)	10	0,23
	Control Structure System: depth of the trace / Level of Coupling: number of interconnections	S5: Change a requirement in the Activities Agenda (Activities Handler)	5	0,11
	Control Structure System: depth of the trace / Level of Coupling: number of interconnections	S6: Change a requiremment to Add/Modify/Query Project Information (Project Handler)	9	0,20
Finance	Control Structure System: depth of the trace / Level of Coupling: number of interconnections	S7: Change a requirement in Request Financial Support Application (Request Handler)	2	0,05

Once all the classes making up the architecture were defined as interfaces, with emphasis on those classes that are related to the scenarios within each of the profiles obtained at this stage, a module was implemented with a view to providing the functionality needed to evaluate these scenarios. This module communicates with its interface through message-passing to the other modules present in the architecture.

Implement and initiate the profiles and simulate the system: In order to implement the profiles, the different scenarios described in the profiles must be executed. It is important to note that the evaluation technique chosen (Simulation) implies certain restrictions. In particular, RAPIDE language does a simulation using what is known as a Poset Browser, where the architecture simulation is shown as a trace (a series of events generated by the different components of the architecture communicating between one another). Figure 5 shows the trace corresponding to the *Maintainability* profile.

Transformation of the Architecture: The quality attributes specified in the profiles must be measured on the traces obtained. Then the architecture must be transformed to improve one or more of these attributes. The idea is to select a transformation that does not adversely affect any of the quality attributes (Bosch 2000).

Out of the four types of transformation proposed by Bosch, it was decided to use the transformation of the Application of a Design Pattern, as it can only be applied to part of the architecture (Bosch 2000). This one was also chosen because it increases *Maintainability*. This transformation makes it easier to replace algorithms in order to execute a specific task (Bosch 2000), promoting the *Maintainability* attribute, which is very important within the domain of a KMS. It was therefore decided to replace the *Observer* pattern of the Architecture in Figure 1, with the *Command* Design pattern (Gamma et al. 1995), since this pattern fosters *Maintainability* better. Figure 6 shows the candidate architecture (Transformed). Note that the design patterns used in their Architecture are: *Chain of Responsibility* (once again) and *Command*.

In the *Command* Design Pattern (Gamma et al. 1995), when a client issues a request it propagates along the chain until it reaches a *ConcreteCommand* object that takes responsibility for handling it. This leads to the following consequences: reduced coupling, easy addition of new commands, support for recording changes, support for transactions and support for the *Undo* operation. Figure 7 shows the structure of the *Command* Design Pattern.

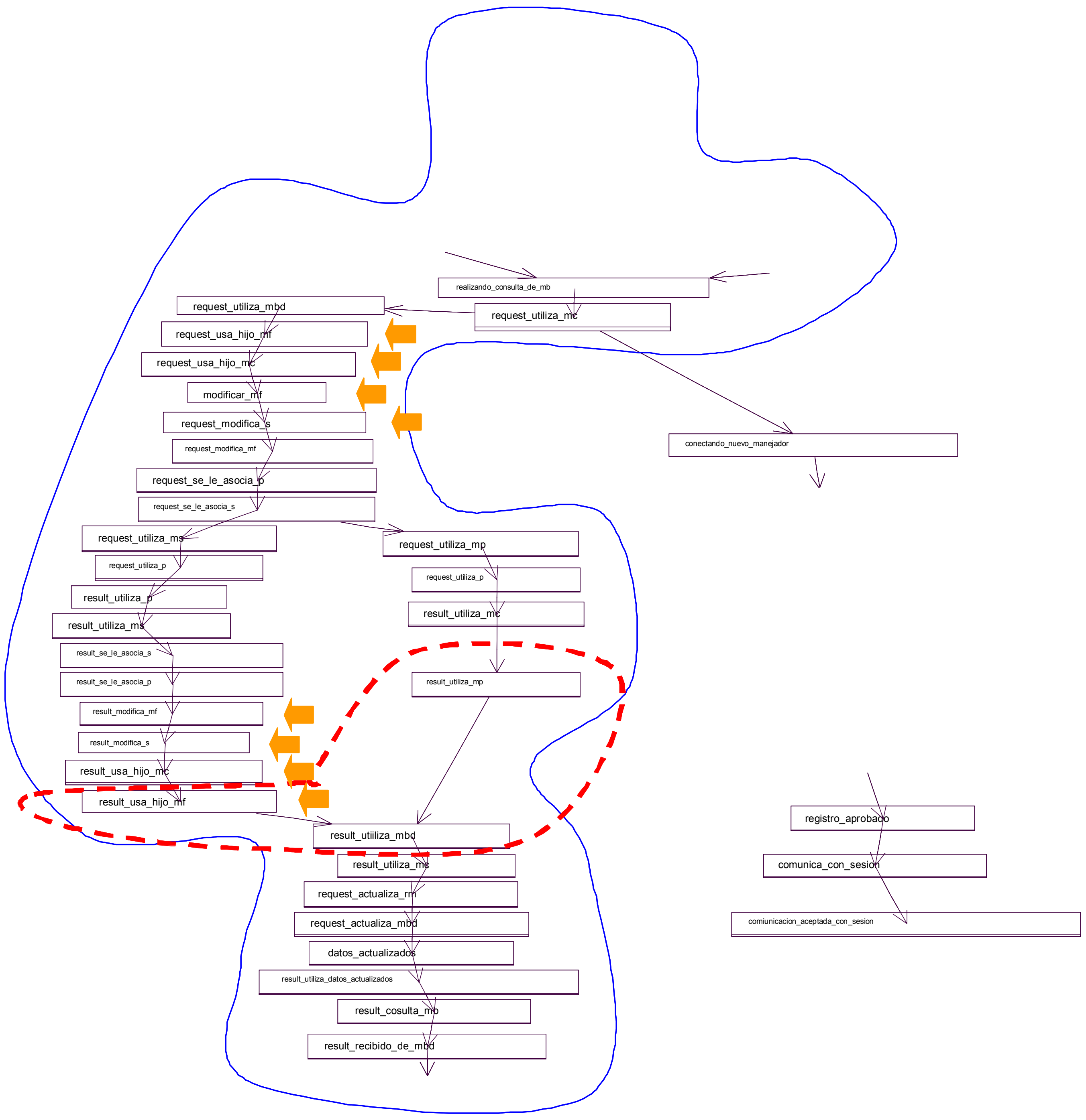

Figure 5. Trace in RAPIDE Corresponding to the Maintainability Profile

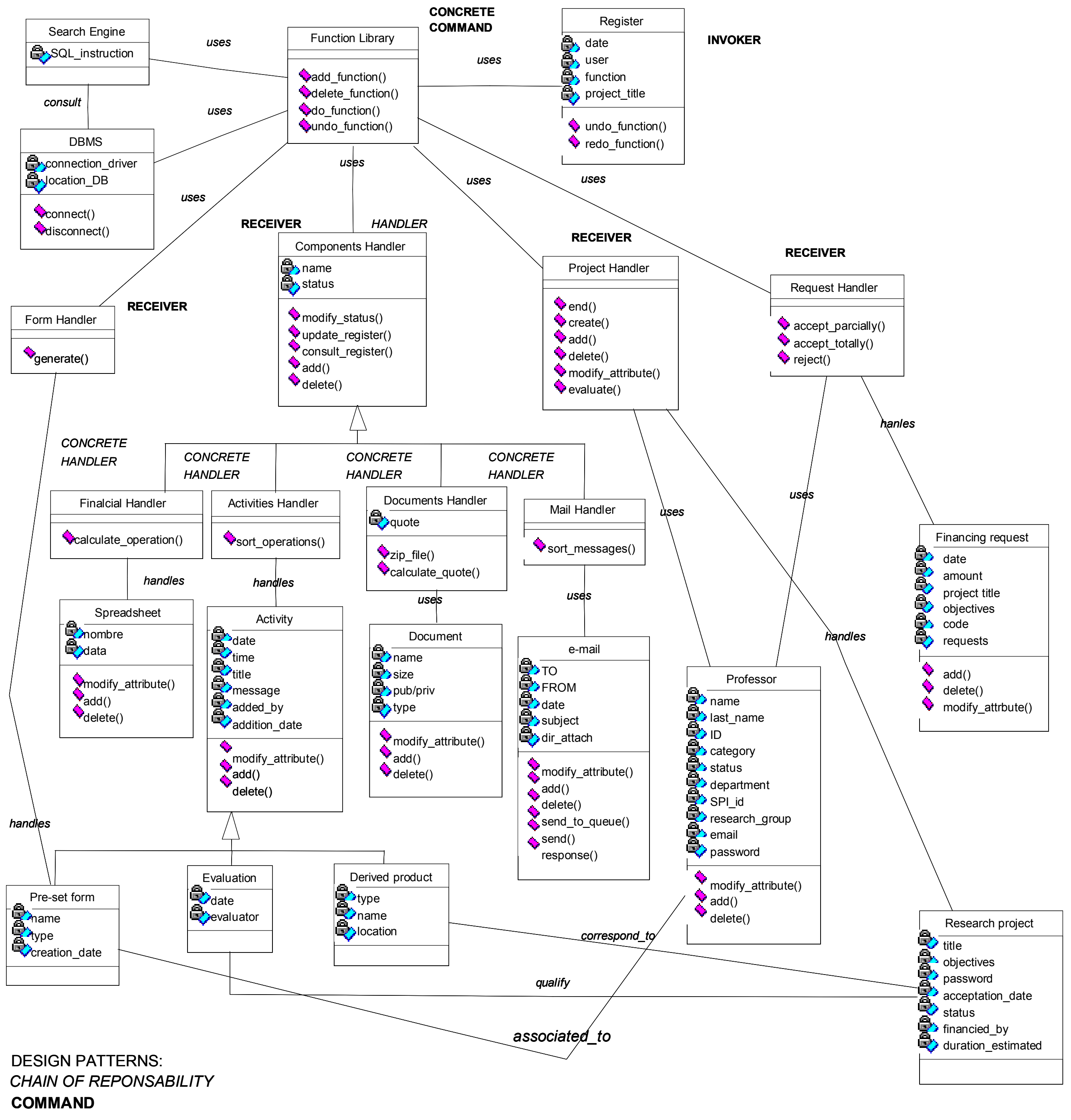

Figure 6. Candidate Architecture Transformed

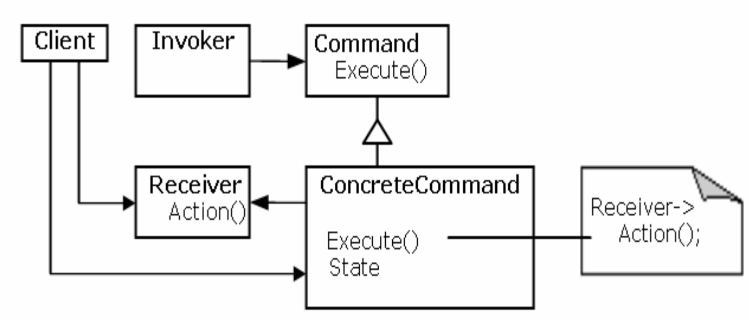

Figure 7. Structure of the *Command* Pattern (Gamma et al. 1995)

In the Candidate Architecture 2, requests would reach each handler that uses some of the functions defined in the *Functions Library*, but before executing them the *Record of Modifications* which in turn checks the feasibility of the function and invokes its execution, is updated. For all these reasons, the *Format Handler*, the *Component Handler*, the *Project Handler* and the *Request Handler* play a similar role to a *Receiver* object, while the *Function Library* class would play the role of a *ConcreteCommand* object and the *Record of Modifications*, would play in turn the role of *Invoker*.

Following application of the above steps, Table 2 presents a summary of the most important measurements for both architectures. Remember that the Candidate Architecture 1 is the same Candidate Architecture show in Figure 1.

Table 2. Summary of the Measurements Taken for Both Architectures

Profile	**Metric**	**Sub-characteristic**	**Candidate Architecture 1**	**Candidate Architecture 2 (transformed)**
Maintainability	Control Structure System	Number of Modules	Addition of one module	Addition of two modules
	Level of Coupling	Number of Interconnections	Average: 2 interconnections	Average: 2.14 interconnections
	Control Structure System	Depth of the Trace	Average: 52 levels	Average: 51.5 levels
Reliability	Fault Tolerance	Functional Dependence	1	2
Efficiency	Behavior over time Use of resources	Use of CPU and memory Response Time	Average: 52	Average: 51.5

When analyzing Table 2, Candidate Architecture 2 (Transformed) shown in Figure 6, can be seen to have the same characteristics:

- **Maintainability:** Improves the Control Structure System metrics as it increases the number of modules that can be added. Despite the fact that the initial architecture in the last two metrics is only slightly higher compared to the transformed architecture, this last Candidate Architecture 2 (Transformed) improves the first as regards the addition of modules. This aspect is relevant for the type of system evaluated here since the knowledge stored varied and is diversified, making it necessary to include extensions to the system and hence constantly to modify the requirements, which is important when it comes to generating, coding and transferring knowledge. Also, in the connections of the Candidate Architecture 2 (Transformed) there is only one connection to the modules added, and in the initial architecture there are two connections. This makes the transformed architecture appear to be more *Mantainable* for this type of system.

- **Reliability**: Candidate Architecture 2 (Transformed) evidently has more Functional Dependence than Candidate Architecture 1. However, one must bear in mind that the *Command* pattern has recovery properties that the *Observer* pattern present in the initial architecture lacks. As far as this characteristic is concerned, this is considered an advantage for this architecture. As far as this characteristic is concerned, it is also important to stress that only one type of measurement was taken: Functional Dependence.

- **Efficiency**: Since the number of events is reduced, there is a decrease in the use of CPU and memory, which amounts to shorter response time.

It can thus be concluded that **Candidate Architecture 2 (Transformed), predicts more support for KMS based on the quality attributes measured in the simulation.** This makes it possible to anticipate problems in the design proposed, so these would be points to which the architect must pay attention.

Having shown the application of the method proposed by Bosch, we shall now present the application of the ATAM method for the same case study.

Architecture Tradeoff Analysis Method – ATAM

The Architecture Tradeoff Analysis Method (ATAM) is an appropriate method for analyzing software architectures that considers multiple quality attributes. ATAM consists of 4 phases, corresponding to time segments in which a series of activities take place (Clements et al. 2002). Phase 0 corresponds to the creation of the evaluation team and the establishment of agreements between the evaluating organization and the organization that owns the architecture to be evaluated. During Phases 1 and 2, the true phases of the ATAM evaluation, the nine steps described in Table 3 are carried out. Phase 1 focuses on the architecture and concentrates on eliciting and analyzing the architectural information. Phase 2 focuses on the stakeholders and concentrates on eliciting its points of view and checking the results of Phase 1. Lastly, in Phase 3 the final report of the architectural evaluation is produced (Clements et al. 2002).

Table 3. Steps and Outputs of the ATAM. Adapted from (Clements et al. 2002)

Steps / Outputs	Prioritized Statement of Quality Attribute Requirements	Catalog of Architectural Approaches Used	Approach- and Quality-Attribute-Specific Analysis Questions	Mapping of Architectural Approaches to Quality Attributes	Risks and Non-risks	Sensitivity and Tradeoff Points
1. Present the ATAM						
2. Present business drivers	X				X	
3. Present architecture		X			X	X
4. Identify architectural approaches		X	X		X	X
5. Generate quality attribute utility tree	X					
6. Analyze architectural approaches		X	X	X	X	X
7. Brainstorm and prioritize scenarios	X					
8. Analyze architectural approaches		X	X	X	X	X
9. Present results						

As the main objective of this article is to present the architectural evaluation of a KMS, including its design patterns, only steps 3 to 6 of the ATAM are described (see shaded area of Table 3). Additionally, as shown in Table 3, the outputs expected from the architectural evaluation will be built during the process.

Step 3 of the ATAM covers the description of the architecture at a high level. The architectural style considered the most appropriate for PROYECTOS DID KMS, was the **Layer** style. It encourages both the logic and the data to be controlled by a server. Using Application Service Provider (ASP), customers simply have to place their orders and wait for the server to respond. The server converts the calculations and operations into HTML response. This is why the way data is presented will vary depending on the browser used by the client. Figure 8 shows the distribution of the different layers. Three layers are proposed: Presentation, Logic and Data.

During Step 4, the architectural *approaches* identified for the architecture are described. Intuitively they are believed to foster quality attributes. Clements et al. (2002) use the term *approach* because not all architects are familiar with the language of Architectural styles, making it difficult to enumerate all the styles used. However, all architects make architectural decisions and the set of these is known as the *approach.* The level of abstraction of the architectural decisions used in this research is the level of "application of design patterns". Thus the Candidate Architectures taken are those presented in the previous sections: Candidate Architecture 1 (Figure 1) and Candidate Architecture 2 (Transformed) (Figure 6). The following step describes how the Quality Attribute Utility Tree is built.

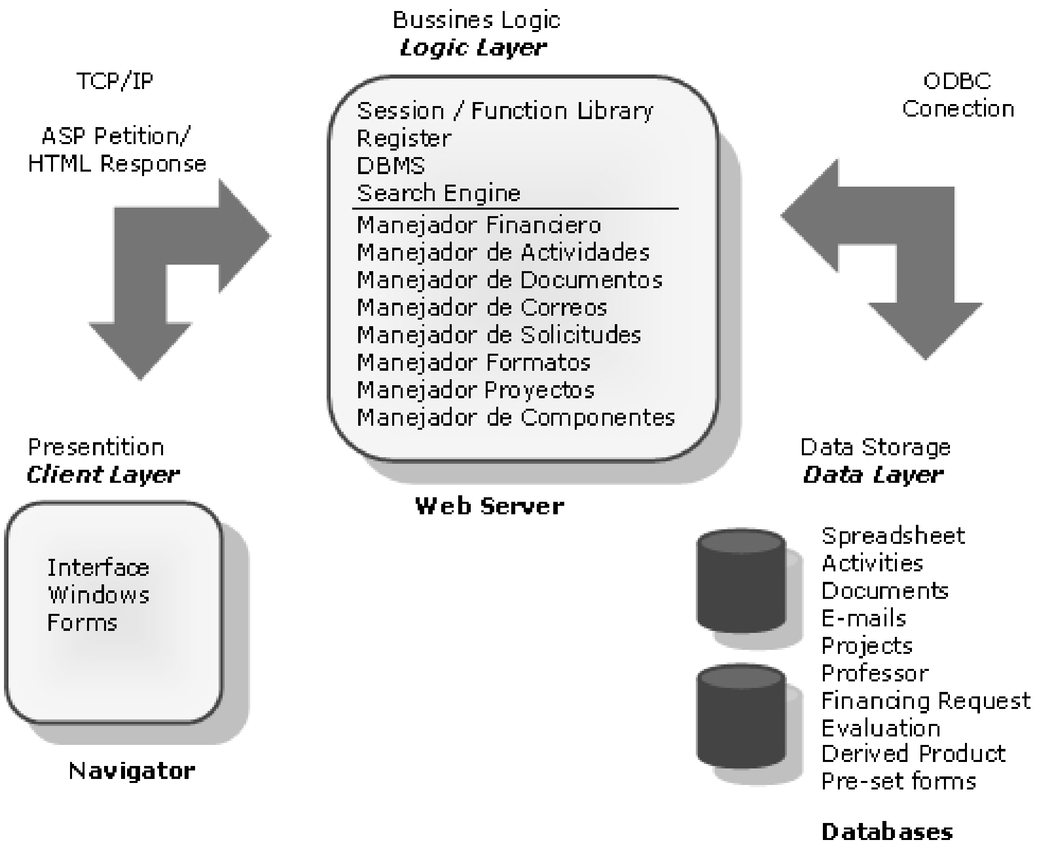

Figure 8. Architectural Style Proposed

Quality Attribute Utility Tree

The output from Step 5 is the Generation of the Utility Tree; this is a prioritization of quality attribute requirements, shown as scenarios. Table 4 shows the Utility Tree built based on the quality architectural attributes chosen during the stakeholders' session.

Analysis of the Architectural Approaches

Step 6 of the ATAM method consists of the analysis of each of the scenarios identified in the Utility Tree in terms of the architectural decisions promoted by each one. The Sensitivity Points of the quality attributes and the Tradeoffs between them are also identified. A Sensitivity Point is a property of one or more components (and/or its relationships) that is critical in order to obtain a response from a particular quality attribute, while a Tradeoff is a property that affects more than one attribute and is a Sensitivity Point for more than one attribute (Clements et al. 2002). The results of the analysis of the scenarios are recorded in tabular form and are normally captured by the person responsible for documenting the design. To build this table, the scenario is broken down into its Stimulus and Response, in order to ensure that each has been captured accurately. Each scenario generates a sequence of steps. These steps provide support for the group discussion, which leads to the Architectural Decisions, the Risks, Non-risks, Sensitivity Points and associated Tradeoffs. Listing the steps is a useful way of considering a scenario, but it is not the ultimate goal, which is to determine the impact that the set of Architectural Decisions has on the ability to attain the scenario (Clements et al. 2002). Table 5 shows the qualitative analysis of one of the 25 scenarios shown in Table 4.

Table 4. Utility Tree

Level 2: Quality attribute	Level 3: Refining the quality attribute	Level 4: Quality attribute scenario
Maintainability	M1: Changes to a module must be made at a low cost, without them significantly affecting other modules	M1.1: Change to a requisite in the *Financial Tool,* taking no longer than 1 day without affecting the *Discussion List*, the *Activity Agenda*, the *Application for Financing*, the *Document Repository* and the *Research Projects*
		M1.2: Change to a requisite in the *Activity Agenda,* taking no longer than 1 day without affecting the *Discussion List*, the *Financial Tool,* the *Application for Financing*, the *Document Repository* and the *Research Projects*
		M1.3: Change to a requisite in the *Application for Financing,* taking no longer than 1 day without affecting the la *Activity Agenda*, the *Financial Tool*, the *Document Repository* and the *Research Projects*
		M1.4: Change to a requisite in the *Document Repository,* taking no longer than 1 day without affecting the *Discussion List*, the *Activity Agenda*, the *Application for Financing*, the *Financial Tool* and the *Research Projects*
		M1.5: Change to a requisite in the in Add/Modify the *Research Projects,* taking no longer than 1 day without affecting the *Discussion List*, the *Activity Agenda*, the *Application for Financing*, the *Document Repository* and the *Financial Tool*
		M1.6: Inclusion of a new component, taking no longer than 1 month, without affecting the rest of the components
	M2: Inclusion of a new component does not imply a considerable increase in coupling	M2.1: Inclusion of a new component must involve a minimum number of interconnections
Reliability	F1: Data integrity must not be lost during a fault	F1.1: There must be at least one recovery block responsible for data integrity when remedying a fault in the *Financial Tool*
		F1.2: There must be at least one recovery block responsible for data integrity when remedying a fault in the *Discussion List*
		F1.3: There must be at least one recovery block responsible for data integrity when remedying a fault in the *Activity Agenda*
		F1.4: There must be at least one recovery block responsible for data integrity when remedying a fault in the *Application for Financing*
		F1.5: There must be at least one recovery block responsible for data integrity when remedying a fault in the *Document Repository*
		F1.6: There must be at least one recovery block responsibility for data integrity when remedying a fault in the *Research Project*
	F2: The average time between faults must not significantly affect system availability	F2.1: There must be at least one component responsible for Exception Handling
		F2.2: The system must process at least 150 successful transactions a day
	F3: In the event of a fault, the user must be sent a detailed notification	F3.1: F3 must take less than 20 seconds.
	F4: The average time taken to remedy faults must be adequate	F4.1: A fault must be remedied in less than 10 minutes.
		F4.2: The operator must be capable of canceling / reinitiating the system in less than 15 minutes.
	F5: The version of the system delivered must contain all the defects identified during the test stage	F5.1: No more than 3 global variables must be used by each component of the architecture
		F5.2: The architecture must have functionally independent modules that facilitate unit testing

Level 2: Quality attribute	Level 3: Refining the quality attribute	Level 4: Quality attribute scenario
Security	S1: Research Project Management must be guaranteed securely through the Internet, taking data security into account	S1.1: There must be at least one component responsible for checking authorization for access
Efficiency	E1: Facilitates rapid, easy and direct searches	E1.1: Responses to requests must not take longer than 5 seconds
		E1.2: Searches must be done without using commands
		E1.3: Searches must be done in a single step
	E2: The system must show the relevant information in text mode to ensure that the main operations take place quickly	E2.1: On pages showing information with images, these must not take up more than 30% of the page.

Table 5. Analysis of a Scenario Associated with the Maintainability Attribute

Scenario #: M2.1.	**Scenario (M2.1) Inclusion of a new component must involve a minimum number of interconnections**				
Attribute	Maintainability				
Environment	During perfective maintenance work				
Stimulus	Inclusion of a new component in response to a new requirement				
Response	Involves a minimum number of associations				
Architectural decisions		**Risk**	**Sensitivity**	**Tradeoff**	**Non-risk**
DA1 Have a component that manages service requests and communicates directly with the specific components (use of the **Observer** design pattern)		R1	S2		
DA2 Include a component that handles requests and communicates directly with the C*omponent Manager ((*use of the **Command** design pattern)			S2		NR2
Reasoning	DA1 has Risk R1 related to the **Observer** pattern associated with it. What may appear to be a simple change on a data object may trigger cascading updates to the observer objects and their dependent objects. Still worse, just as updates are diffused, so are errors and because their origin is unknown, recovery operations may be difficult to carry out. Both DA1 and DA2 denote Sensitivity Point S2. *Maintainability* is sensitive to the **number of interconnections** when it comes to adding new modules or components. If this only implies, in addition to the necessary update of the *Change Record* and the *Programmer Manual*, the addition of a respective class with its corresponding link, the attribute will be positively affected. DA2 is thought not to have a Non-risk condition, as no negative consequence related to the use of the **Command** pattern is known. As far as the analysis of architectural decisions is concerned, it can be affirmed that DA2 gives a better response to the Stimulus, as it generates less interconnections than DA1. However, as indicated previously, both denote a Sensitivity Point that is critical for the attribute.				
Diagram of the architecture	See Figure 6				

Taking into account the reasoning and the architectural decisions of each of the analyses of the previous scenarios, Table 6 shows a summarized comparison of both architectures based on the advantages, disadvantages, risks and non-risks associated with the use of different design patterns.

Candidate Architecture 1 can be said not to have more advantages than Candidate Architecture 2 (Transformed) since the risks included in it are nonguaranteed receipt in the event of the chain not being properly configured. Candidate Architecture 2 (Transformed) on the other hand guarantees data integrity since it stores the previous values, which facilitates a much fuller and more reliable control of modifications, fostering *Reliability*; each transaction encapsulates a series of activities and participants, which has a positive impact on effectiveness, but it may end up being inefficient and, further, it enables there to be better control and distribution of functions, thereby fostering *Maintainability*.

Table 6. Scenario Analysis Summary for Both Architectures

Attributes	Candidate Architecture 1	Candidate Architecture 2 (Transformed)
Maintainability	Has a **Risk** associated with the **Observer** pattern, since just as updates are diffused, so are errors.	Has a **Non-risk** condition as no negative consequence related to the use of the **Command** pattern is known. Provides a significant **advantage** by permitting better control and distribution of functions.
Reliability	Has a **Risk** condition associated with the **Observer** pattern, since because the source of the errors is unknown, recovery operations may be hard and difficult to undertake.	Has a **Non-risk** condition identified for the Reliability attribute. Has the **advantage** of guaranteeing data integrity since the previous values are stored, which provides much fuller and more reliable control of modifications
Efficiency	By using the **Observer** pattern, all the observers are notified when there is a change in the state of the data stored and, depending on the number of Observers; this may reduce response time (**disadvantage**).	By using the **Command** pattern, each transaction encapsulates a set of activities and participants, which has a positive impact on effectiveness, but it may become less Efficient (**disadvantage**).
	Both architectures have a **Risk** condition related to the **Chain of Responsibility** pattern, since there is no guarantee of receipt in the event that the chain of transmission of responsibilities has not been properly configured.	

Having analyzed Table 6, it can be affirmed that Candidate Architecture 1 is no more advantageous than Candidate Architecture 2 (Transformed) and, furthermore, it implies greater risks for the application of the design pattern proposed. By contrast, Candidate Architecture 2 (Transformed) has a Non-risk condition and, in turn, fosters the *Maintainability* and *Reliability* attributes.

Having completed all the steps required by the ATAM method for evaluating the architecture, the level of detail was found to be sufficient to begin developing the system with a good degree of certainty as regards the ideal structure of the architecture and the quality characteristics and attributes expected.

Lessons Learned

"The software architecture of a system is the earliest artifact that enables the priorities among competing concerns to be analyzed, and it is the artifacts that manifest the concerns as systems qualities. The trade-off between performance and security, between maintainability and reliability, and between the cost of the current development effort and the cost of future developments are all manifested in the architecture… An architecture is the summary result of a set of business and technical decisions" (Bass et al. 1998). It is therefore necessary to gather together all the lessons learned in the different applications of the evaluation methods described above. Each one is presented below:

1. **Use of scenarios as an evaluation tool:** Both methods are based on scenarios. A scenario is a short statement describing an interaction of one of the stakeholders with the system (Clements et al. 2002). In order to evaluate Architecture, both methods propose that scenarios be identified as an input for the evaluation. The stakeholders are responsible for identifying them.

2. **Weighting of scenarios:** Meaning that both for the formulation of Profiles (Bosch) and for the Utility Tree (ATAM), it is suggested that the scenarios be weighted; in other words, for each of the scenarios, the stakeholders must specify their importance in relation to the rest. This importance stems from the probability of occurring and the non-functional requirements.

3. **Identification of quality attributes:** Both methods consider identification of the quality attributes as an input for their evaluations, but do not give any details or guidance on how to obtain them.

4. **Quality specification techniques:** ATAM proposes the Utility Tree technique in order to specify the quality attributes. This technique breaks them down to the level of their scenarios with their respective weighting. The Bosch method proposes the Profiles technique, which specifies for each attribute its possible scenarios with their absolute and relative weighting.

5. **Evaluation inputs:** An ATAM input is at least two possible architectures. This calls for a high degree of expertise by the architect in the type of system to be evaluated. One input from the Bosch method is an architecture that satisfies the functional requirements, which is transformed following its evaluation.

6. **Bosch method simulation technique:** In order to apply the Bosch method, the simulation technique with the Architecture Definition Language (ADL), RAPIDE was applied. This technique lengthened development time and required training and expertise on the ADL used.

7. **Participation by the stakeholders:** Both methods require participation by the stakeholders in the system. Even though this increases development costs, it also fosters commitment.

8. **Right choice of architecture:** Once both methods have been applied, the ideal architecture is obtained according to the evaluation carried out. Both methods document the "attention focus" for the architect; in other words, they pinpoint the architectural elements to which special attention must be paid.

9. **Costs:** In terms of cost (professional hours) both methods were the same, because ATAM increased the number of participants and the Bosch method combined with Simulation Technique increased the time taken to train developers.

10. **Dimension of the Architecture:** ATAM is appropriate for medium-sized systems where stakeholders can reach a consensus and the architect can carefully and qualitatively compare each of the scenarios identified. The Bosch method, combined with the Simulation Technique, is suitable for systems that handle large amounts of objects, because the Simulation enables the behavior of the different quality attributes to be observed when they undergo significant modifications in a relatively short space of time.

Conclusions

Systems Evaluation must start at an early stage in the development process to keep down reworking. Today, successful systems are characterized by complying with certain quality attributes such as Efficiency and Reliability, among others. Architecture is an aspect to be evaluated and one that guarantees that the quality attributes desired will be fostered.

In this research, were applied two architectural methods to the same case study. The methods used were the Software Architecture Design Method (Bosch 2000) and the Architecture Tradeoff Analysis Method - ATAM (Clements et al. 2002).

The importance that both methods afford to stakeholder participation when designing a system must be emphasized, as part of the guarantee of its future use. The method proposed by Bosch combined with Simulation Technique is based on quantitative measurements and characteristics, while the ATAM method fosters the qualitative analysis of the scenarios.

The lessons learned facilitate the choice of architectural evaluation method, depending on the characteristics of the system and the expertise of the developers. The ability to keep learning should not be discarded, which is why the recommendation is to apply both methods in different domains so as to refine the lessons presented here.

This research only analyzes both methods for the KMS domain; it is necessary to extend the comparison to other domains to study the application of them for different kinds of systems. It does not recommend any particular method since it recognizes the architect necessities as a major driver for selecting the right one.

Acknowledgments

This research was financed by Fondo Nacional de Ciencia, Tecnología e Innovación (FONACIT) of the República Bolivariana de Venezuela, through the S1-2001000794 project.

References

Bass, L., Clements, P. and Kazman, R. *Software Architecture in Practice*. Addison Wesley, 1998.

Bosch, J. *Design and use of Software Architecture*. Addison-Wesley, 2000.

Chirinos, L.; Losavio, F. and Pérez, M. Feature Analysis For Quality-Based Architecture Design Methods. *Proceeding de las Jornadas de Ciencias de la Computación Chilenas 2001* (JCCC2001). Caracas, Venezuela, 2001, 1-9.

Clements, P., Kazman, R. and Klein, M. *Evaluating Software Architecture. Methods and Case Studies. SEI Series in Software Engineering.* Addison-Wesley, 2002.

Domínguez, K. *Sistema de Gestión del Conocimiento para Proyectos de Investigación. Trabajo de Grado publicado*, Universidad Simón Bolívar, 2001.

Gamma, R.; Helm, R.; Johnson, R. and Vlissides, J. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

Jacobson, I., Booch, G. and Rumbaugh, J. *El Proceso Unificado de Desarrollo de Software.* Addison-Wesley, 2000.

Kitchenham, B., Linkman S., and Law D. "DESMET: A methodology for evaluating software engineering methods and tools," *IEEE Computing & Control Engineering Journal* (8:3), June 1997, pp.120-126.

Kruchten, P. *The Rational Unified Process.* Addison Wesley Longman, Inc. 1999.

Meyer, B. *Object Oriented Software Construction*. Prentice Hall, 1997.

Ortega, M., Pérez, M. and Rojas, T. A model for software Product Quality with a Systemic Focus. *Proceedings of The 4th World Multiconference on Systemics, Cybernetics and Informatics SCI 2000 and The 6th Intrernational Conference on Information Systems, Analysis and Synthesis* (Orlando, Julio 2000) Callaos and Callaos eds., 532-538.

Whitten, J., Bentley, L., Dittman, K. *System Analysis and Design Methods*. Fifth Edition. McGraw Hill. 2002.