Software Performance-driven Quality Refinement of MINPHIS: an architectural view

Ishaya Gambo[1], Rhoda Ikono[2], Olaronke Iroju[3], Theresa Omodunbi[4] and Abimbola Soriyan[5]

[1, 2, 4 & 5] Computer Science and Engineering Department, Obafemi Awolowo University
Ile-Ife, Osun State, Nigeria

[3] Computer Science Department, Adeyemi College of Education
Ondo, Ondo State, Nigeria

Abstract

Performance is an important attribute of a software system. The need to use suitable techniques to analyze the performance characteristics of any software system is paramount most especially at the architectural level. Performance as a requirement often originates from the organization's business goals. In the case of the Made in Nigeria Primary Healthcare Information System (MINPHIS) as a product, the business goal is to enter new and emerging geographic markets where the system will be expected to perform effectively. In this paper we considered performance refinement issues of MINPHIS application from the architectural point of view. We present the entity-relationship diagram (ERD) as a role-based quality model of performance attributes showing the main stakeholder roles of MINPHIS software system. The performance goal refinement will be to support regulations that require life system like the MINPHIS application that keeps patient records and generates various reports for health management and research purposes. The paper also presents the module view of MINPHIS architecture, and show who the major stakeholders are and the performance refinement characterization.

Keywords: *Software Architecture, MINPHIS, Quality Refinement, Software Quality, Software Performances.*

1. Introduction

Software quality attributes should be considered throughout design, implementation, and deployment in which software architecture plays major roles. No quality attribute is entirely dependent on design, nor is it entirely dependent on implementation or deployment. Satisfactory results are a matter of getting the big picture (architecture) as well as the details (implementation) correct, as in [1]. For example some of the software qualities like performance involve both architectural and non-architectural dependencies. It depends partially on how communication is necessary among components (architectural), partially on what functionality has been allocated to each components (architectural), partially on how shared resources are allocated (architectural), partially on the choice of algorithms to implement selected functionality (non-architectural), and partially on how these algorithms are coded (non-architectural). Performance is an important quality attribute of software systems, which can be determined and ensured by the architecture. Performance failures can result in damaged customer relations, lost productivity for users, lost revenue, cost overruns due to tuning or redesign, and missed market windows. Clements in his opinion on performance pointed out that "Performance is largely a function of the frequency and nature of inter-component communication, in addition to the performance characteristics of the components themselves, and hence can be predicted by studying the architecture of a system" as in [2]. This further supports Perry and Wolf in [3], [4] and [2] that says "there is growing recognition of the role of architecture in determining the quality of a software system." In our opinion, performance can be one among the quality of a software system. More so, Clements and Northrup in [2] opined that "Whether or not a system will be able to exhibit its desired (or required) quality attributes is largely determined by the time the architecture is chosen."

Obviously, the role software architecture plays in the overall system quality has been established in [5]. If the architecture is not right, the system will not meet its requirements. In the software engineering community, software architecture has been identified as an increasingly important part of software development and quality control and management need to be carried out throughout the whole development process to ensure implementation of required quality characteristics. In this paper we argue that performance problems in software systems usually have their roots in poor architectural and poor design decisions early in the software life cycle. The software architecture presents the performance artifacts to be used for decision making and analysis during evaluation.

However, software performance greatly imposes conditions on functional requirements such as speed, efficiency, availability, accuracy, throughput, response time, recovery time, and resource usage. For the

Copyright (c) 2013 International Journal of Computer Science Issues. All Rights Reserved.

MINPHIS application, performance attribute quality attribute is of paramount concern to all stakeholders of the system. Performance as a requirement often originates from the organisation's business goals. In the case of MINPHIS as a product, the business goal is to enter new and emerging geographic markets where the system will be expected to perform effectively. In this paper we considered performance refinement issues of MINPHIS application from the architectural point of view. The performance goal refinement will be to support regulations that require life system like the MINPHIS application that keeps patient records and generates various reports for health management and research purposes. The paper also presents the module view of MINPHIS architecture, and show who the major stakeholders are and the performance refinement characterization.

2. MINPHIS Architectural Structures and Views

The MINPHIS architecture is a 2-tier architecture. There are four layers, separated from each other by well-defined interfaces depicted by dotted line. As a 2-tier architecture, it consists of data server (i.e. the FileMan database and the M software, the legacy system on MINPHIS, which can access the database directly) and the client application. The database server is where the database serves up data based on queries submitted by the application using the hierarchical database system as the case is with MINPHIS, while the application on the client computer consumes the data and presents it in readable format. Architecturally, the 2-tier architecture is intended to improve usability by supporting a form-based, user-friendly interface. It also improves scalability by accommodating up to 100 users, and improves flexibility by allowing data to be shared, usually within a homogeneous environment. As a client/server, architecture it reduces network traffic by providing a query response rather than total file transfer. It improves multi-user updating through a graphical user interface (GUI) front end to a shared database. This is why Schussel in [6] and [7] opined that "in client/server architectures, Remote Procedure Calls (RPCs) or standard query language (SQL) statements are typically used to communicate between the client and server." Based on this, in a 2-tier architecture, the user system interface can be located in the user's desktop environment and the database management services in a server that is a more powerful machine that services many clients.

However, figure 1 below depicts the module structure of MINPHIS architectural structure and view where the elements are modules seen as units of implementation. The module view represents a code-based way of considering the system. Here, areas of functional responsibilities are assigned. The module structure provides the basis for knowing the primary functional responsibility assigned to each module. The module view (which may be subsystems) describes the system's decomposition of functionality, along with the objects, procedures, functions that populate these, and the relations among them. The module structure shows the elements which are units of implementation. To a programmer, it shows how the system needs to be structured as a set of code units. Fig. 1 will actually allow us to know the primary functional responsibility assigned to each module, the actual software elements the module is allowed to use. The module view shows the relationship existing within each module or layer by generalisation or specification (i.e., inheritance), dependency and composition. This information can further be used for architectural analysis and evaluation.

3. Scenario of MINPHIS Architecture as a Two-tier Application

Scenarios are used to represent stakeholders' interests and to understand quality attribute requirements. They are a good way of synthesizing individual interpretations of a software quality into a common view. This view is more concrete than the general definition of software quality and it also incorporates the specifics of a system to be developed (i.e., it is more context-sensitive, as in [8]. Most of the considered architecture analysis methods use scenarios. The existing practices with scenarios are systematized in [9]. For the MINPHIS architecture, the architecture is client/server architecture and also a two-tier application, which contains two active components: the client, which requests data and the server, which delivers data. Basically, the application's processing is done separately for database queries and updates and for business logic processing and user interface presentation. The network binds the back end to the front end, although both tiers can be present on the same hardware. In the architecture, the clients establish connections with the server, and these connections are kept open until the clients terminate. The communication protocols are often proprietary. Both clients and servers use database-specific libraries to communicate with each other. Security is entirely managed by the database engine. Logins are controlled by the database, such as user roles and permissions on database objects.

Copyright (c) 2013 International Journal of Computer Science Issues. All Rights Reserved.

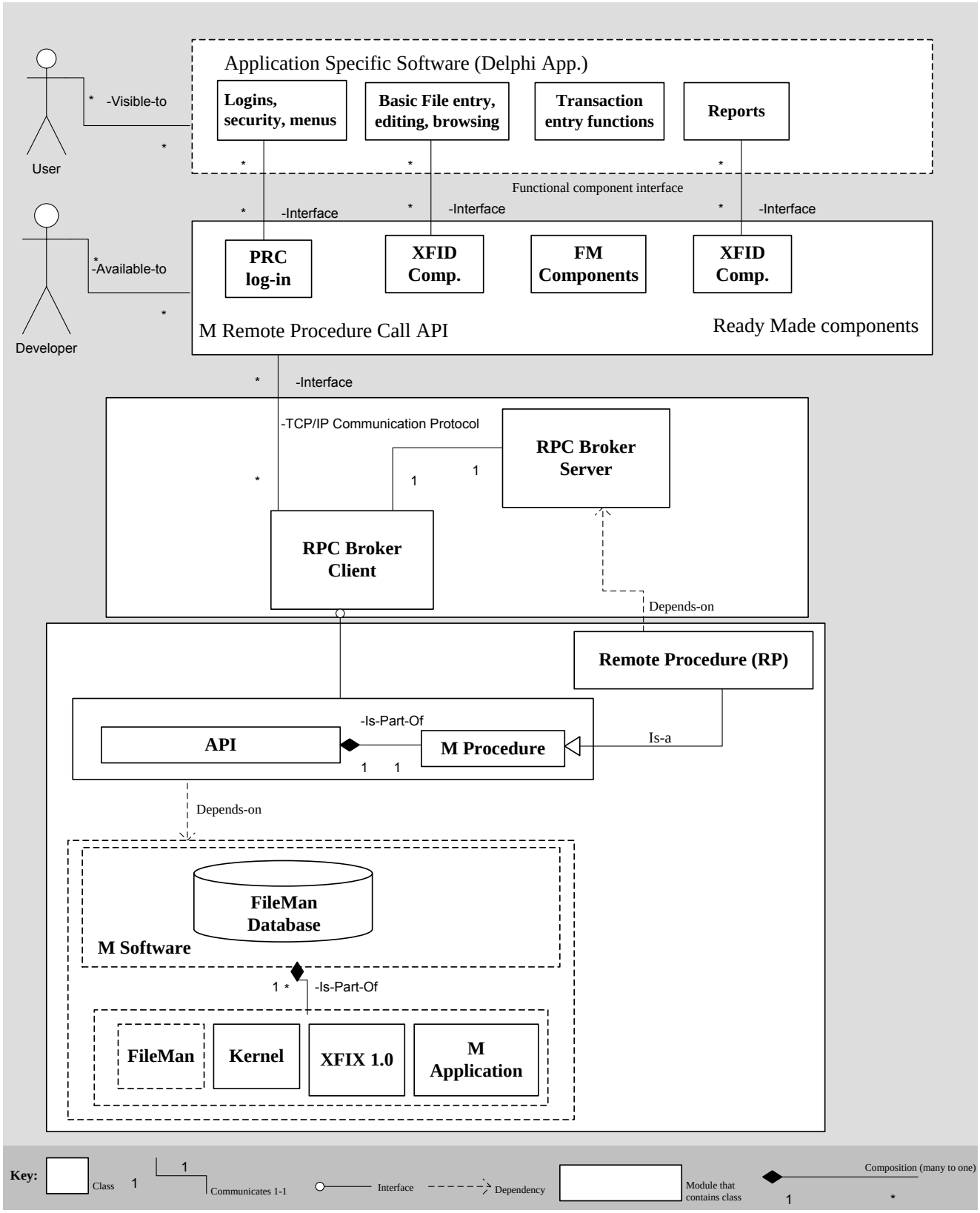

Fig. 1 Module View of MINPHIS Architecture

Copyright (c) 2013 International Journal of Computer Science Issues. All Rights Reserved.

4. MINPHIS Stakeholders and Performance Quality Characteristics

The quality of a software system is primarily perceived by its stakeholders because they have the best impression if the software system meets their requirements or not. These requirements are based on the stakeholders' expectations and needs. The software performance quality requirement of MINPHIS application specifies the overall quality characteristic used for the refinement process. In the MINPHIS system, it was discovered that the different stakeholders who include the end users, analyst, quality assurance evaluator (researcher), system administrator, developer, architect and management have performance quality expectation in addition to other quality characteristics towards the system. All the stakeholders are interested in the performance quality characteristics considering the time it takes for the system to respond to a particular request, and from the architectural point of view, we consider the stimuli that could cause the architecture to respond to changes.

The main stakeholder roles of MINPHIS application are end user, administrator and developer. The roles of the developers and end user or users of the system are the major concern in this paper. The performance quality attribute is viewed from how the system functions when it is being used. From interaction with the system, five major types of users were identified, namely Doctors, Medical Record Officers, Lab Technicians, Nurses, and Pharmacist. All these users have different levels of access and they perform different operations on the current MINPHIS. They are all interested in the performance of the system. Based on the observation made on the system (MINPHIS), the response time when the system is being used in handling and processing of patient records and data was taken into consideration. The performance issue considered is a measure of how the system responds e.g. the time to respond to events or number of events processed in a time interval. This was checked during system execution. The entity-relationship diagram (ERD) in Fig. 2 shows a role-based quality model that means it shows the main stakeholder roles of MINPHIS software system assigned to the performance quality attributes which they demand. Starting from the role-based quality model in fig. 2, performance quality attribute is identified for the MINPHIS application. Fig. 3 and 4 illustrate the mentioned refinement to metrics and quantitative values which can enable the evaluation of the performance quality attributes, using any software architectural evaluation technique like the Architecture Trade-off Analysis Method (ATAM). For the representation, again the fundamental modeling concept (FMC) ERD notation is used to show relation between characteristic and metrics. On the shown fig. 3 for the refinement of the performance characteristic I, the qualitative evaluation through scenario is based on the analysis of the response time and resource utilization.

5. MINPHIS Performance Scenarios

According to the work of Bass et al. in [1], "A scenario is a precise system-independent specification of a type of quality attribute requirement that consists of":

- A stimulus: a condition that needs to be considered when it arrives at the system
- A response: the activity undertaken after the arrival of the stimulus
- A source of the stimulus: the entity (e.g., a human or computer system) that generated the stimulus
- An environment: the conditions under which the stimulus occurs, for example, when the system is in an overloaded condition.
- A stimulated artefact: some artefact is stimulated. It could be the whole system or pieces of it.
- A response measure: the attribute-specific constraint that must be satisfied by the response.

With this, we have the performance scenario used for MINPHIS application as:
"A data from patient form is required to arrive every 10milliseconds (ms) at the patient master database under normal condition. The system has to process the stimulus within 1ms."

The above performance scenario is an event which is initiated with resource demands specified and the event must be completed within a time interval. We divided our performance quality attribute characterization into three categories: *external stimuli, architectural decisions*, and *responses*. The e*xternal stimuli* (or *stimuli)* are the events that cause the architecture to respond or change. The architectural decisions are those aspects of the architecture, which includes the components, connectors and their properties. The architectural decisions have direct impact on achieving attributes responses. A typical example of this could be the external stimuli for performance, which can be the events such as messages, interrupts, or user keystrokes that result in computation being initiated. In the case of the above generated scenario, the patient form is the external stimuli. Performance architectural decisions include processor and network arbitration mechanisms; concurrency structures including processes, threads, and processors; and properties including process priorities

Copyright (c) 2013 International Journal of Computer Science Issues. All Rights Reserved.

and execution times. Responses are characterised by measurable quantities such as latency and throughput. The performance architectural decisions are represented in the performance tactics.

The goal in presenting performance attribute characterizations is to suggest a framework for thinking about quality attributes. The attribute characterizations help to ensure attribute coverage as well as offering a rationale for asking elicitation questions. For example, irrespective of the style being analyzed, the latency (a measure of response) is a function of resources such as CPUs and LANs, resource arbitration such as scheduling policy, resource consumption such as CPU execution time, and external events such message arrivals, as in [1].

For the MINPHIS system that keeps electronic patients record, the response is the number of patients records entered through the patient registration form that can be processed in a minute for the patients database to be updated within a given time frame and the variation in the arriving time. The performance scenario begins with a request for some service arriving at the system. Satisfying the request requires resources to be consumed. While this is happening the system may be simultaneously serving other requests. In [1] it was opined that "the response of the system to a stimulus can be characterised by latency (the time between the arrival of the stimulus and the system's response to it), deadlines in processing, the throughput of the system (e.g., the number of transactions the system can process in a second), the jitter of the response (the variation in latency), the number of events not processed because the system was too busy to respond, and the data that was lost because the system was too busy." This was actually articulated in the MINPHIS application.

6. Performance issues on the Patient Data

On the patient data, a form as shown in fig. 5 was made available which serves dual purpose of collecting data about a patient coming to the hospital for the patient and finding existing patient in the hospital. This form is an important form in that it provides the required data needed by other modules of the application. The information provided through this form is used to:

- Identify a patient by the **Hospital Number** which is unique to every patient coming to the hospital
- Provide an avenue to know the previous medical history for the patient
- Enable access to other aspects of the application by the patient

The **Patient** master database stores the information that is entered through this form. The performance generated can be to register a patient using the patient registration form, and let the record be updated in the patient database for use by all other modules.

7. MINPHIS Performance Tactics

Performance is about timing. Events (interrupts, messages, requests from users, or the passage of time) occur, and the system must respond to them. There are a variety of characterisations of event arrival and the response but basically performance is concerned with how long it takes the system to respond when an event occurs. One of the things that make performance complicated is the number of event sources and arrival patterns. Events can arrive from user requests, from other systems, or from within the system. For example, the MINPHIS application as an example is a system based on powerful servers running Microsoft NT, Intersystems Cache, the VA Kernel and FileMan, and the FixIT software developed in Finland. The system, having spanned through a thorough Information System Development Process with clinical and patient information well taken care of via a wide range of reports that could aid health policy and decision makers, handles patients requests by producing different output for the Obafemi Awolowo University Teaching Hospital Complext (OAUTHC) management, and other tertiary hospital where the system is deployed for used. This system gets events from its users (possibly numbering in the tens or hundreds of thousands), the response might be the number of patients information that can be processed in a minute. In this case, the pattern of events arriving and the pattern of responses can be characterised, and this characterisation forms the language with which performance scenarios are constructed. The goal of the performance tactics is to generate a response to an event arriving at the system within some time constraint. The event can be single or a stream and is the trigger for a request to perform computation. It can be the arrival of a message, the expiration of a time interval, the detection of a significant change of state in the system's environment, and so forth. The system processes the events and generates a response. Performance tactics control the time within which a response is generated.

Copyright (c) 2013 International Journal of Computer Science Issues. All Rights Reserved.

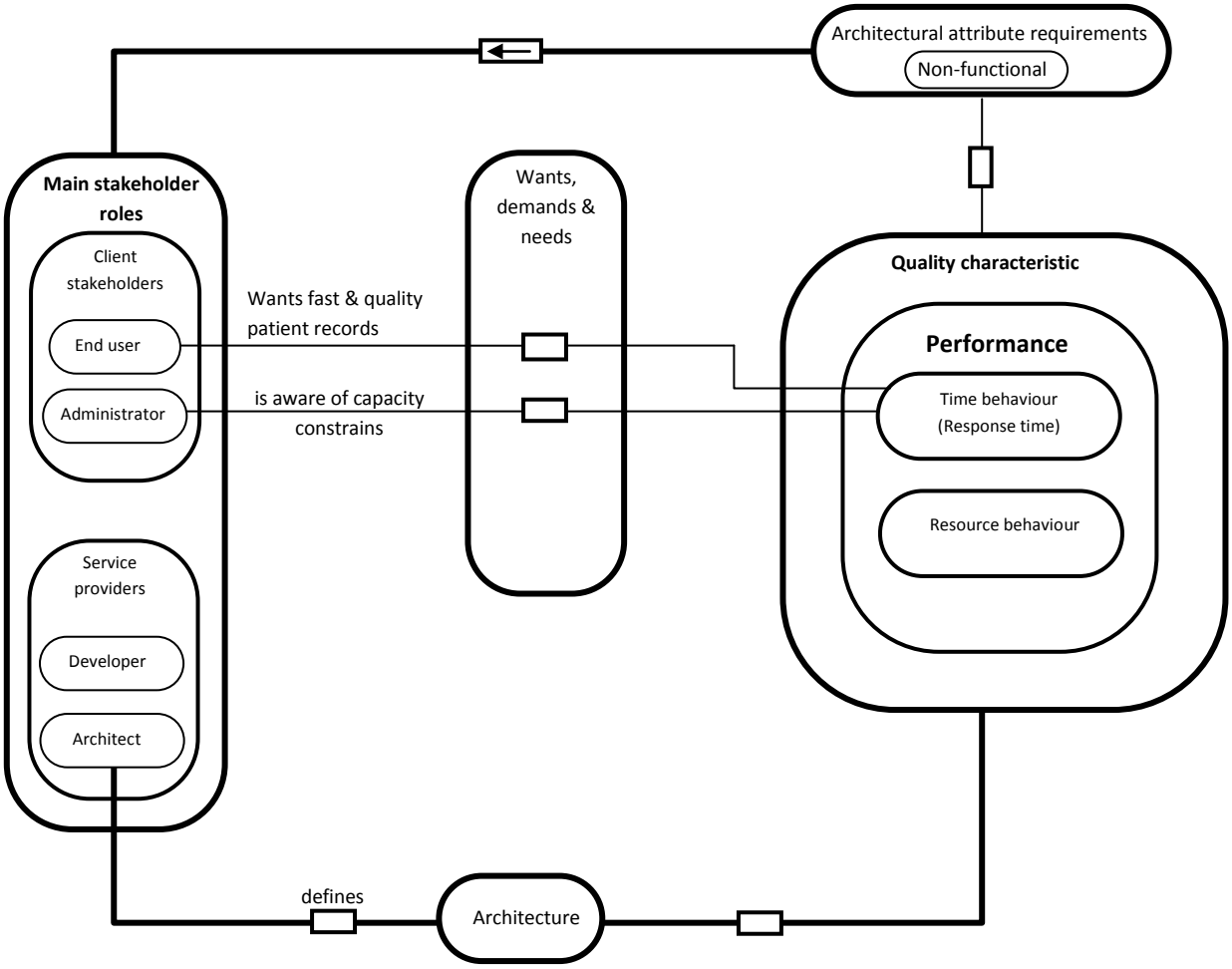

Fig. 2 MINPHIS Role-based quality model in FMC ER-diagram notation

Copyright (c) 2013 International Journal of Computer Science Issues. All Rights Reserved.

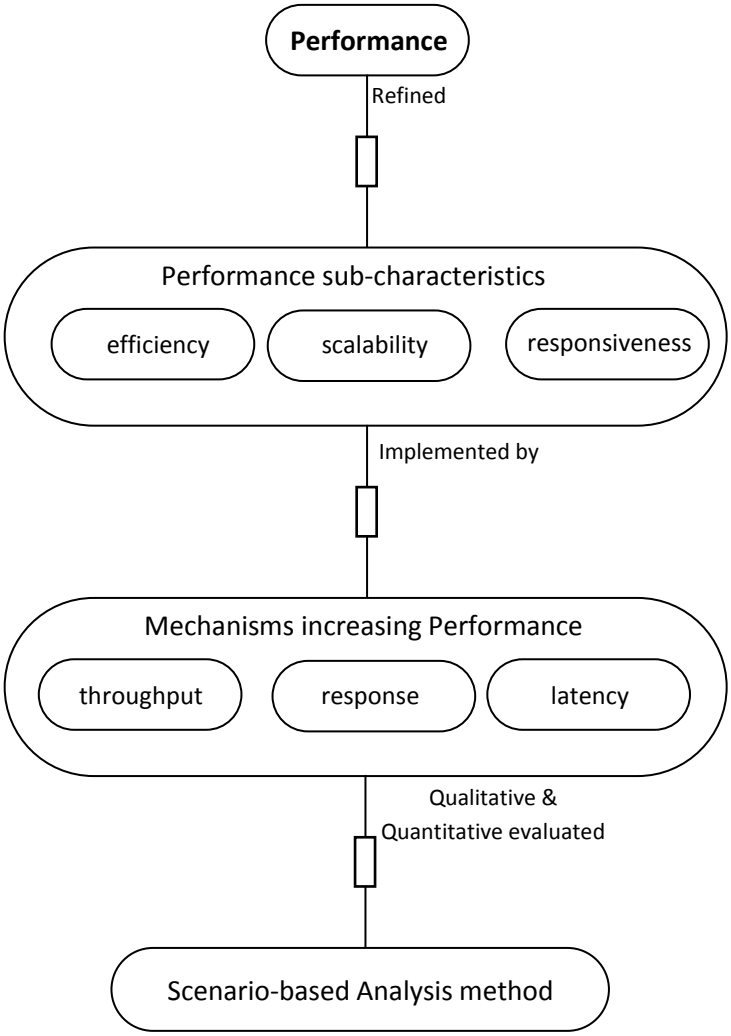

Fig. 3 Refinement process for quality characteristics performance I

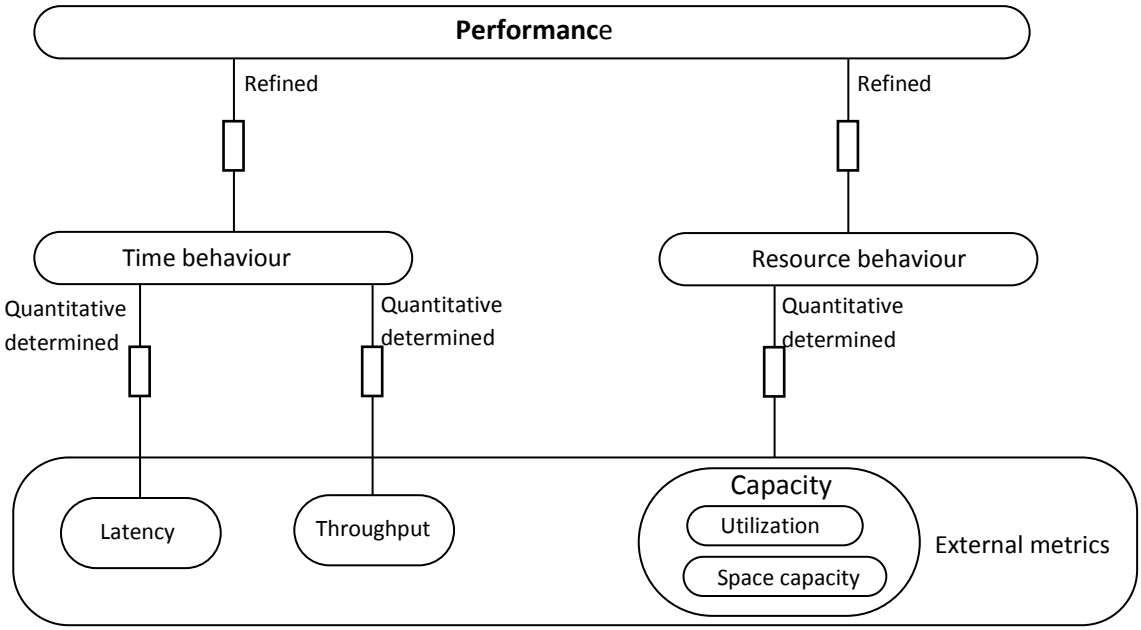

Fig. 4 Refinement process for quality characteristics performance II

Copyright (c) 2013 International Journal of Computer Science Issues. All Rights Reserved.

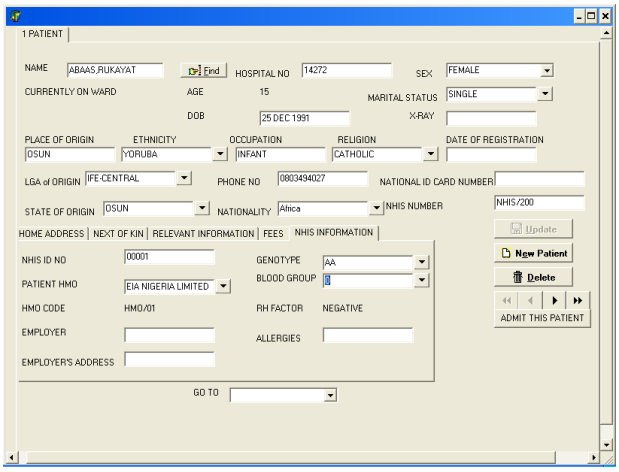

Fig. 5 The Patient Registration Form

8. Conclusion

In conclusion, quality driven refinement of software system is very necessary for a proper characterization of quality attributes. The performance refinement process as a quality characteristic of MINPHIS application further gave the metrics and quantitative values which can enable the evaluation of the performance quality attributes using an architectural evaluation method like ATAM. The characterization is a way of showing what exactly need to be considered in a given quality attribute. This implies that quality control and management must be carried out through the whole development process of software systems to ensure the implementation of required quality characteristics.

Acknowledgment

I will like to appreciate Paul C. Clement (Ph.D) who was formally at the Software Engineering Institute (SEI), Carnegie Mellon University (CMU), Pittsburgh in USA for comments, research materials and guidance during the period of my research from which this paper is written. Your hospitable spirit really gave me confidence to come closer and to have more confidence in the subject area of software architecture and software quality research. You made me feel proud of what I have done and what I will yet do.

References

[1] Bass, L., Clements, P., and Kazman, R. (2003). Software Architecture in Practice, second ed. Addison-Wesley, Reading, MA.

[2] Clements, P., and Northrop, L. (1996). Software Architecture: An Executive Overview. CMU/SEI-96-TR-003.

[3] Perry and Wolf (1992): Perry, D., and Wolf, A. (1992). "Foundations for the Study of Software Architecture," *ACM SIGSOFT Software Engineering Notes*, Vol. 17, No. 4, pp. 40-52.

[4] Garlan and Shaw (1996): Garlan, D., and Shaw, M. (1996). Software Architecture: Perspective on an emerging discipline. *Prentice Hall*.

[5] Clements, P., Kazman, R., and Klein, M. (2002). Evaluating Software Architecture. *Methods and Case Studies:* Addison-Wesley, Reading MA, Boston.

[6] Schussel, George,(1996):Client/Server Past, Present, and Future (online). Available WWW <URL: http://www.dciexpo.com/geos/> (1995).

[7] Edelstein, Herb. (1994) "Unraveling Client/Server Architecture." DBMS 7, 5 (May 1994): 34(7).

[8] Barbacci, M., Ellison, R., Lattanze, A., Stafford, J., Weinstock, C., and Wood, W. (2003). Quality Attribute Workshops (QAW), third ed. (CMU/SEI-2003-TR-016). Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA

[9] Kazman, R., Klein, M., and Clements, P. (2000). ATAM: A Method for Architecture Evaluation. *Technical Report CMU/SEI-2000-TR-004*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA.

Copyright (c) 2013 International Journal of Computer Science Issues. All Rights Reserved.