Evaluating User Interface Management Systems based on Quality Attributes and Unit Operations

Abdelkareem M. Alashqar
Information Systems Dept.,
Faculty of Computer and Information Sciences,
Mansoura University, Egypt

Ahmad Abo Elfetouh
Information Systems Dept.,
Faculty of Computer and Information Sciences,
Mansoura University, Egypt

Hazem M. El-Bakry
Information Systems Dept.,
Faculty of Computer and Information Sciences,
Mansoura University, Egypt

ABSTRACT
Software architecture is an essential early stage in the software design process. In this stage, the architect should give the quality attributes a special consideration because a good level of meeting these attributes can be performed by well-designed architecture. This means that there is a close relationship between quality attributes and software architecture. However, quality attributes can be achieved through the appropriate application of a set of unit operations. A unit operation is a systematic designing operation that can be applied directly to system architecture. Architectural styles (patterns) include high level design decisions that address quality attributes. Many general architectural styles are defined in the literature. For the domain of user interactive systems there are many architectural styles that address some important quality attributes. In many cases, it is essential to evaluate software styles in terms of their achievement of the required quality attributes by analyzing the relationships between these attributes, unit operations, and styles. This evaluation can help and facilitate the process of selecting a specified style. In this paper the authors propose a structured quantitative evaluation method to show a rank of four well-known user interface management systems (UIMSs) in terms of their supporting a set of six important selected quality attributes.

Keywords
Quality Attributes, User Interface Management Systems, Unit Operations, Software Architectural Styles.

1. INTRODUCTION
Software architecture is designed after determining and organizing software requirements. An essential goal of designing system architecture is to meet the required functionality as well as the quality attributes. Quality attributes are also called non-functional requirements (NFRs). Ideally, the architect relies on architectural design styles (patterns) in designing software architecture. Many software architectural styles are found in the literature and also used by practitioners. Such styles include good design decisions. One category of these styles is user interface architectural styles. They are called User Interface Management Systems (UIMSs) and used in interactive systems to achieve usability [1]. Common and well-known UIMSs include Seehim, Arch/Slinky, MVC and PAC. A major issue in UIMSs is the separation between the semantics of the application and the interface provided for the user to make use of that semantics. This separation of concerns is supported by several good quality attributes such as portability, reusability, modifiability, performance, scalability and the capability of applying multiple interfaces. The achievement of quality attributes of a system are closely connected with the software architecture for that system. Furthermore, these qualities can be achieved through the appropriate application of a set of unit operations which are fundamentally applied in traditional engineering disciplines. Specific architectures can be derived from an understanding of the unit operations and the quality attributes that will be achieved by that architecture. The authors in [2] define and discuss six unit operations used frequently by experienced designers. They argue that codifying derivations based on unit operations and their relationship with quality attributes will allow the creation of architectures to become a rote activity as it is in traditional engineering. These defined unit operations are: Separation, Abstraction, Uniform Decomposition, Resource Sharing, Replication, and Compression. Many UIMSs architectural styles have been described in the literature (see for example [2] [3] [4]). The relationships of UIMSs architectural styles, unit operation and quality attributes are discussed in [5]. To evaluate UIMSs and to provide guidance for the software architect in selecting the most appropriate UIMS architectural style, the interactions between quality attributes, unit operations and styles should be analyzed and quantitatively measured using matrices manipulation. The results of this systematic method should be considered as decision criteria within quality based architectural design process. In this paper, the authors propose an approach for quantitative evaluation of four well-known UIMSs in terms of supporting a set of six given important quality attributes by considering architectural unit operations to refine the evaluations.

The rest paper is organized as follows: Sections 2 discusses four well-known UIMSs styles in addition to a discussion of a set of six important quality attributes that supported by these styles. Section 3 provides quantitative evaluations of the relationship between quality attributes and unit operations. Section 4 shows quantitative evaluations of the degree to which UIMSs are achieving unit operations. Section 5 provides calculation results of the quantitative effect of incorporating the quality attributes into architectural styles via matrix transformation method using the data from the "quality-unit operation" and "unit operation-style" quantified relationships. Section 5 discusses related work, and Section 6 provides conclusion and the future work orientation. The approach used throughout this paper is the analysis of the effect of incorporating six unit operations into four UIMSs styles in achieving a specific set of quality attributes.

2. USER INTERFACE MANAGEMENT SYSTEMS
The four most widely used user interface management systems (UIMSs) are explained next.

1.1 Seeheim model
Seeheim model is considered as the first UIMS that was proposed at a workshop in 1985 at Seeheim in Germany [6].

Seeheim model encompasses three main components as depicted in Figure 1; which are Application Interface, Dialogue Control and Presentation Interface. The Application Interface describes the Application semantics from the viewpoint of the user interface and describes the data structures and the procedures that the Application exports to the user interface. The Presentation Interface defines the behavior of the system as perceived and manipulated by the user. The Dialogue Control is viewed as a mediator between the Application Interface and the Presentation. Although the Dialogue Control plays as the initiator of direct link between the other two components, the lower little box permits the Application Interface to bypass the Dialogue Control in order to improve performance.

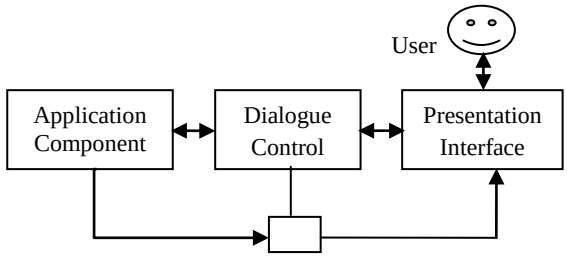

Fig 1: The Seeheim Model

The Seeheim model is an example of the functional decomposition approach where the three main components work as the semantic, syntactic and lexical interactions. The semantic includes the description of functions, the syntactic define the sequence of both inputs and outputs, and the lexical facilitates the sequence of user actions. The Seeheim model also constitutes a pipe-filter structure that provides a sequence of data transformation [3] [4].

1.2 Arch/Slinky model

Arch has more additional layers than the Seeheim model but it provides some improvements in functional decomposition structure as shown in Figure 2. These improvements include a refinement in the level of abstraction of each component in addition to an explicit definition of the data structures exchanged between its main components. This level of abstraction is performed by two main components called Virtual Application and Virtual Toolkit adapters.

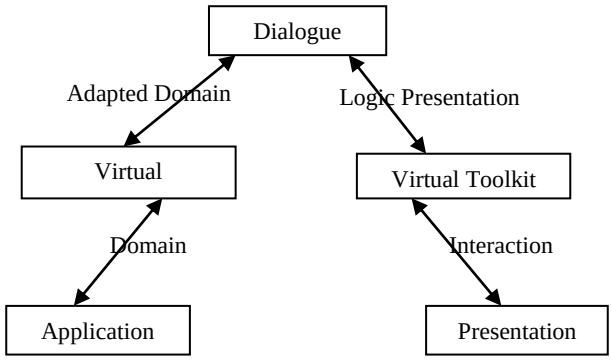

Fig 2: The Arch/Slinky Model

As shown in Figure 2, at one side of the model, the Application component covers the domain-dependent concepts and functions. And at the other side, the Presentation component is responsible for presenting the domain concepts and functions in terms of physical interaction objects to the user. The Dialogue Component plays an important role in regulating the sequence of tasks via the Arch model components.

Arch applies a clear abstraction between the major functional components of the UIMS. So there is no direct interaction between the Application, the Dialogue and the Presentation. The interaction of exchanging data is performed by two additional adaptors; the Virtual Application and the Virtual Toolkit. The Virtual Application is intended to accommodate various forms of mismatch between the Application and the user interface of the system. As shown in Figure 2, data transfer through the Virtual Application is performed in terms of domain objects. Ideally, domain objects match the user's mental representation of a particular domain concept. In many cases, the domain specific objects may be implemented in inappropriate way for users, so these objects may need to be adapted. The Virtual Toolkit is the other adapter component that separates the rendering of domain objects from the actual interaction toolkit of the target platform by providing logical presentation objects. This separation adds more flexibility for modification when changing the physical interaction toolkit in the UIMS. In such cases there is no need to change the logical presentation objects. The Slinky part of the model referred to the ability to expand and balance the allocation of functions to components. Slinky notion provides the ability for a given implementation architecture to place the dialogue, virtual application, and application in a separate or group them in a single structural components [4] [7].

1.3 Model-View-Controller (MVC)

MVC is a type of Agent-based models. Agent-based models structure an UIMS as a collection of computational units called agents. Each agent has a state, possesses an expertise, and is capable of initiating and reacting to events. Some agents called interactor agents can present the user with data about its internal state. The agent resembles an implemented object in object oriented programming environment. In MVC, an agent is modeled along three functional perspectives: the Model, the View, and the Controller. A Model defines the abstract functional perspectives of the agent. The View defines the perceivable behavior of the agent for output. The Controller defines the perceivable behavior of the agent for inputs. The overall means of interaction behavior with user is ideally achieved by the View and the Controller [4].

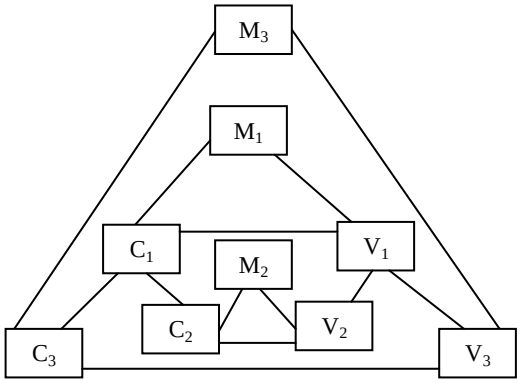

Fig 3: The MVC Model

MVC was proposed in the Smalltalk programming environment [8] [9]. Smalltalk was one of the earliest successful object-oriented programming systems whose main feature was the ability to build new interactive systems based on existing ones [3]. Figure 3 shows the implementation of the MVC model in the Smalltalk environment where Controllers and Views are implemented as hierarchies classes. Models, which are domain-dependent, are organized according to the domain requirements. It is not necessarily to organize Models in a hierarchical way. Each component in the

MVC can communicate with the other by means of method invocation. Typically, the Controller translates the user's actions into method calls on the Model. The Model sends a notification to the View and the Controller informing them that its internal state has been changed. The View reading the exact changes occurred in the Model and then updates the user interface displays according to an external request. The Controller acts as the mediator between the Model and the View. Practically the Controller may encompass instances for both the Model and the View. Because the View and the Controller need to know about changes occurred in the Model, the later provides a registration facility to permit multiple Views and Controllers to interact with the Model for any important changes. This technique allows adding and removing Views for different user workstation simply. However, all registered Views will be notified according to any Model changes despite that some of them do not need these changes.

1.4 Presentation-Abstraction-Control (PAC)

PAC model proposed by [10]. As an MVC, PAC is a type of agent-based models [4]. The main components of PAC are: the Presentation component which combines both input and output behavior [3], the Abstraction component which is considered as the functional core of the UIMS, and the Control component which expresses multiple forms of dependencies. The main role of the Control component in one agent is to achieve dependencies between the Abstraction and the Presentation components of the agent and to communicate with other agents in the UIMS. In the PAC model there is no direct communication between Abstraction and Presentation components. Instead this communication, coordination and also any aspects of data transformations are achieved by Controls. As shown in Figure 4, the flow of data between agents transit through the Controls in a hierarchical way where the connectors of a PAC hierarchy achieve communication relationships. These relationships in PAC between components do not represent class relations as in the object-oriented implementation of the MVC model that discussed previously.

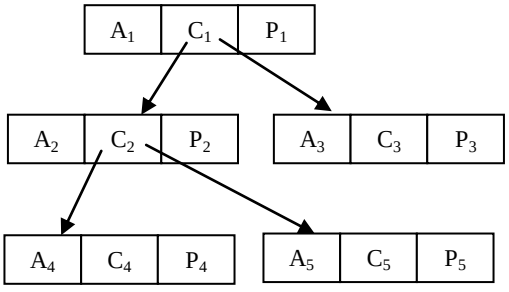

Fig 4: The PAC Model

By comparing MVC to PAC, it is important to denote that MVC separate input techniques from outputs, whereas PAC localizes them in the Presentation part. Contrary to PAC, MVC has no explicit concept of mediator for expressing the relationships and the coordination between agents. However MVC and PAC sometimes outperform other models because they encompass different functional decompositions in their architectural design styles.

1.5 Quality Attributes Supported by UIMSs

An important issue in the area of research of developing UIMSs architecture is to apply good design decisions in order to achieve good level of quality attributes. The authors explain here six important quality attributes supported by an UIMS architectures [2] [3] [11]:

- **Scalability:** is the ability to expand the system to meet any future changes and modifications by simply increasing its size.
- **Modifiability:** is the ability to extend the system functionality by adding new required business features.
- **Portability:** is the ability of a system to execute on different hardware and software platforms. It includes developing the system to be operational on various operating systems.
- **Performance:** is the measure of how well the system responds to its inputs. Important measures include response time, resource utilization, and throughput.
- **Reliability:** is the ability of the operational system to provide a good error-free level. The mean time between failures is considered an important measurement for reliability.
- **Reusability:** is the ability to reuse significant number of existing qualified components or modules in the current system. Reuse includes, functions, classes, group of classes, and small working packages in the system being developed. The main objective of reusability is to reduce time of development.

These quality attributes will be included in the evaluation of the four well-known UIMSs.

3. INTERACTION BETWEEN QUALITY ATTRIBUTES AND UNIT OPERATIONS

The achievement of quality attributes of a system are closely connected with the software architecture for that system. These qualities can be achieved through the appropriate application of a set of unit operations. Specific architectures can be derived from an understanding of the unit operations and the quality attributes to be achieved by that architecture. The authors in [2] define and discuss six unit operations used daily by experienced designers in traditional engineering fields. The defined unit operations are separation, abstraction, uniform composition, resource sharing, replication, and compression. Understanding the relationship between unit operations and quality attributes is an important issue in the design process of system architectures. These unit operations will be used in the evaluation approach adopted in this paper. A discussion for each of the six unit operations is stated next [2] [5] [13] [14] [15].

- **Separation:** this unit operation provides the capability for the designer to isolate several pieces of functionality and distribute them into several components. Each component has an interface to its environment. This isolation helps in achieving specific system characteristics or quality attributes. A good example of separation is UIMSs, where presentation, dialogue and application functionalities are separated.
- **Abstraction:** Abstraction unit operation supports the creating of a virtual machine. The benefit of virtual machine is to hide the component functionality from its physical implementation. Although virtual machines are complex to create, they can be reused by other software components. Virtual toolkits are popular virtual machines in UIMSs. The developers usually create such virtual

toolkits only once to support user interactive components that can be operational into various platform environments.

- **Compression:** compression is the contradictory of separation unit operation. While separation means adding new layers, compression means removing unnecessary layers from the architecture. The normal case in software engineering is applying separation which is supported by different fields in computer science such as the client-server style, distributed and parallel computing, and object-oriented development. Compression is rarely used but designer. Architect may use compression in software architecture to improve system performance by eliminating or combining components to reduce overhead communication between system components.

- **Decomposition:** decomposition is a consequence of separation unit operation where large system components are further decomposed into smaller ones. Separating a large component into small uniform size components is a type of decomposition called modularity or uniform decomposition. Applying modularity improves the cohesion between system components and hence facilitates the modification of any future changes. Two techniques can be applied in the decomposing process. The first one named "part-whole" where the system can only be built from the resulting decomposed components. The other is called "is-a" where each of the decomposed components represents a specialization of its parent's functionality. The MVC style applies modularity where the user interface is decomposed into a set of uniform components each of them contains a model, a view, and a controller. PAC also applies the same techniques in decomposing the system into presentation abstraction control.

- **Replication:** replication unit operation means duplicating the same component within the system architecture. The main goal of this operation is to enhance reliability and performance. When components are replicated, it reduces the possibility system failure. Distributing the processing into more than one component also increases the performance of the system. Reliability is increased through increased redundancy by having several components perform the same operation. Whereas performance may be increased through increased parallelism by dividing a single function among several components.

- **Resource Sharing:** is a unit operation that allows the resource (data or services) which has a control or a manager to be shared among multiple components. Resource sharing can improve portability and modifiability of systems because it reduces the coupling among components. But it hinders the system performance because of the additional overhead that added when applying access control mechanisms. The components that use a shared resource are less likely to be reused in later applications because of the tight relationships among components. Databases, integrated computer-assited software engineering (CASE) tool environments, and servers in a client-server system are common examples of resource sharing mechanism.

Table 1 illustrates "quality-unit operation" relationship for six quality attributes and six unit operations that concern a UIMS. The relationships between the unit operations and quality attributes are selected to be under focus in the evaluation process that adopted in this paper. The assessment of how much the unit operation supports quality attributes is adapted from [2] [12]. Where the meaning of symbols used in Table 1 is as follows: "+1" means that a style positively supports a quality attributes, "0" stands for neutral or no support, "–1" means that the style has a negative influence on achieving a quality attributes.

In addition to the analysis results stated in the discussion part of unit operations which explains the assigned numbers in Table1 1, it can be stated that applying separation will break down the large system into smaller pieces, where these smaller pieces have specific interfaces with each other. This leads to an ability to increase size of the system to become more scalable, so +1 is put in the separation/scalability cell in Table 1. If separation encloses the capability of hiding platform dependencies, it helps in achieving portability (+1). In contrary, separation requires the creation of additional interfaces, hence in this case, it hinders performance (-1).

4. INTERACTION BETWEEN UIMS ARCHITECTURES AND UNIT OPERATIONS

For comparing and ranking UIMSs styles, the relationships between the unit operations and these styles are deeply studied. Each style is also analyzed to evaluate the degree to which it supports the unit operations. In this paper, a five point scale is adopted to identify the degree to which the style achieves the unit operation. Where "+2" means that a style strongly supports a unit operation, "+1" stands for some support, "0" stands for neutral or no support, "-1" means that the style has a negative influence on achieving a unit operation, "-2" means that the style has a strong negative influence on achieving a unit operation. This five point scale has been used by different researchers for the purpose evaluating architectural styles. For example, in [17] a five point scale is used to calculate the total score for architectural style in achieving specific quality attributes.

Table 1: Relationships between quality attributes and unit operations

Unit Op./Quality	Scalability	Modifiability	Portability	Performance	Reliability	Reusability
Separation	+1	+1	+1	-1	0	+1
Abstraction	+1	+1	+1	-1	0	+1
Compression	-1	-1	-1	+1	0	-1
Uniform Decomposition	+1	0	0	0	0	0
Replication	-1	-1	-1	+1	+1	-1
Resource Sharing	0	+1	+1	-1	-1	-1

And in [12] a five point scale is used to identify the effect magnitude of incorporating tactics within architectural styles. Table 2 summarizes the findings of the degree to which Seeheim, Arch/Slinky, MVC and PAC styles support unit operations. For example, it is obvious that all the styles apply separation on their architectures. This is achieved by separating what the user interacts (e.g. presentation in Seeheim and View in MVC) from the application model, and the mediator between the both. Seeheim style applies functional decomposing in "is-a" fashion by separating a system into three main components, while MVC style applies "part-whole" decomposition unit operation. The difference is that Seeheim style assumes that the most important scenarios to guard against are porting from toolkit to toolkit, and isolating the application, presentation and dialogue from changes in each other. MVC style assumes that modifications are likely to occur between different functional objects, and so makes the minimization of the effect of such changes its main quality goal. More separation is done by the Arch/Slinky style on the functionality on each of the presentation and the model components. In addition to that more separation is also done on MVC style so that input is departed from output and hence a value of +2 is assigned, while a value of +1 is assigned to the two other styles. By applying abstraction, the connection between the presentation and the dialogue components is made indirect. Abstraction mechanism allows additional component(s) to be inserted between the presentation and dialogue that maps between the both, demonstrating a virtual presentation toolkit to the dialogue, thus forcing the dialogue to conform to the abstractions presented by the virtual toolkit. Arch/Slinky style applies abstraction in this way so a value of +2 is assigned to it. A lower positive value +1 is assigned to the remainder set of styles under focus with regard to abstraction. The reason is that all of these styles apply separation mechanism, and separation is considered the super-type of abstraction, although they are not the same concept.

The feedback line that directs the connection between the presentation and the application bypassing the dialog, resulting in adding compression on this architecture so a value of +1 is assigned. However, the opposite of compression is done on Arck/Slinky and MVC styles, while the degree of compression is negatively lower in PAC. So a value of -2 is assigned to MVC style and a value of -1 is assigned to PAC style. The analysis shows that none of the four UIMSs styles support replication so a value of 0 is assigned to all of them.

Uniform decomposition applied heavily in PAC style where the system is decomposed into uniform components; all user inputs and outputs are combined in the presentation component, the abstraction encompasses the functional core of the application, while the control of an agent is in charge of communicating with other agents as well as of expressing dependencies between the Abstraction and the Presentation. Thus a value of +2 is assigned. A lower degree of uniform decomposition is applied to Seeheim and MVC so a value of +1 was assigned. While uniform decomposition unit operation is not supported by Arch/slinky so a value of 0 was assigned. Because system performance was a critical issue in user interfaces particularly graphical user interface, and resource sharing has the most harmful effects on system performance, resource sharing is not applied to UIMS style so a value of -1 is assigned to all styles.

Table 2: The degree of achieving unit operations by UIMSs styles

Unit op./UIMS	Seeheim	Arch/Slinky	MVC	PAC
Separation	+1	+2	+2	+1
Abstraction	+1	+2	+1	+1
Compression	+1	-1	-2	-1
Uniform Decomposition	+1	0	+1	+2
Replication	0	0	0	0
Resource Sharing	-1	-1	-1	-1
Average	+0.5	+0.33	+0.33	+0.33

5. EVALUATING UIMSs STYLES BASED ON QUALITY ATTRIBUTES AND UNIT OPERATIONS

Section 4 of this paper provides a discussion of the quantitative achievement of UIMS styles for each unit operation which was shown as "unit operation-style" relationship. The degree to which an UIMS style achieves the set of all unit operations is performed by computing the averaging values for the set of unit operations against each UIMS style. Ideally, these averaging values range from -2 to +2 because all of the assigned values in the "unit operation-style" relationship fall into this range. Each averaging value shows the quantitative degree for a style in achieving the set of unit operations, where the value of -2 represents the least achieving state, and the value of +2 represents the most achieving state. The averaging values for the quantitative achievement degree of unit operations by the four styles in Table 2 exposed that Seeheim is the most supporting style for unit operations, followed by Arch/Slinky, MVC, and PAC that have the same lower supporting averaging values. Although such calculation results are important, they do not provide sufficient support for system architect who is in position of evaluating architectural styles. Hence, to provide the system architect with more convenient quantitative evaluations, the interaction between quality attributes and unit operations should be considered. To do so, the data from "quality-unit operation" relationship should be considered and combined with the data from "unit operation-style" relationship, where the calculation results are based on the following steps:

1. Let M is the matrix representing the "quality-unit operation" relationship. M is composed of a set of m_{qu} values where m_{qu} represents the effect value of the unit operation u on the quality q, as shown in Table 1.
2. Let N is the matrix representing the "unit operation-style" relationship. N is composed of a set of n_{us} values where n_{us} is the value of the achievement of unit operation u by the style s, as shown in Table 2.

3. Calculate W = *M* * *N*, where an element w_{ab} in *W* is the scalar product of the a^{th} row of *M* with the b^{th} column of *N*. This will produce the "style-quality" matrix that represents the effect value of incorporating quality attributes in architectural styles as shown in Table 3.

4. Calculate the arithmetic mean value by dividing the sum of all values under each style divided by the total number of quality attributes. This will produce the effect of each style in supporting the set of six quality attributes.

Matrix multiplication that stated in Step 3 is very important because it takes into account the interaction between quality attributes and unit operations, where this type of interaction can be positive, neutral or negative, as stated previously in Section 3. It is important to note that the mean values computed when following Step 4, are relative to the number of quality attributes under focus which is constrained to six in this paper. In other words, if the number of quality attributes is reduced to five, the mean results will dramatically change.

Following the above calculation steps will give the results shown in Table 3, which are also graphically depicted in Figure 5. The results mainly show the quantitative effect of incorporating six quality attributes, Scalability, Modifiability, Portability, Performance, Reliability and Reusability into four UIMS styles Seeheim, Arch/Slinky, MVC and PAC. The results expose that when all of the six quality attributes under focus are considered in the evaluated system, then the degree of how much an UIMS style supports for these attributes is ranked from highest to lowest as: MVC > PAC > Arch/Slinky > Seeheim. This means that MVC is the most supporting style for the focused set of quality attributes while applying six unit operations; Separation, Abstraction, Uniform Composition, Resource Sharing, Replication, and Compression. The results also guide the system architect on how different styles vary in providing best support for specific quality attributes as shown in Figure 6. It is noticed that, MVC is best in terms of scalability, modifiability, and portability of the system; however it is one of the lowest in terms of performance. On the other hand Arch/Slinky style is the best in terms of reliability.

6. RELATED WORK

A lot of user interface management systems and their architectural styles have been proposed and described.

Table 3: The effect of four well-known UIMSs on six quality attributes

	Seeheim	Arch/Slinky	MVC	PAC
Scalability	2	3	6	5
Modifiability	0	2	4	2
Portability	0	2	4	2
Performance	0	-2	-4	-2
Reliability	1	2	1	1
Reusability	2	4	6	4
Mean	0.83	2.66	2.83	2

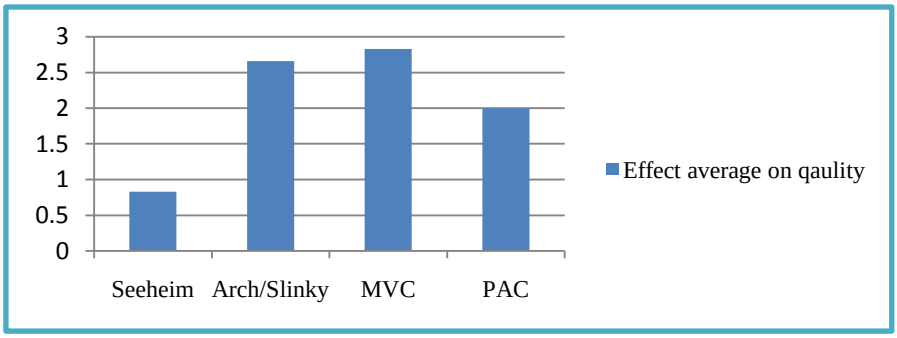

Fig 5: The effect average of UIMSs on quality

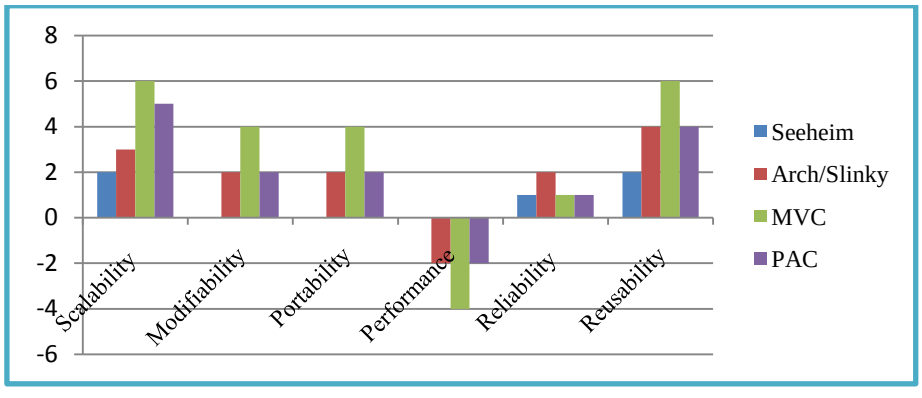

Fig 6: The effect of four well-known UIMSs on each of the six quality attribute

Though, USIMs styles evaluations are limited and mostly qualitative, the authors' work aims at providing a quantitative method for comparing, evaluating and ranking UIMSs architecture. The proposed method uses, in particular, information drawn from unit operations descriptions as given in [2] and the quantitative effect of these unit operations on quality attributes.

Kazman and Bass [5] propose a general model that relates user interface architecture styles to unit operations and quality attributes. In particular they provide an analysis concentrates on the interaction between unit operations and user interface architectures. The authors' approach may be seen as complementary to this framework as they consider both the effect of a unit operation on a style and the effect of the unit operation on other quality attributes.

A similar work is done in [12] and [16] where a matrix calculation is used in incorporating the impact values of architectural styles on quality attributes. The authors in [16] present a quantitative evaluation of a set of selected architectural styles regarding their support for the evolvability quality attribute. They defined subcharacteristics of evolvability and mapped them to properties for good architectural design in order to be able to determine the impact on evolvability. Whereas the authors in [12] proposed a quantitative approach to selecting architectural styles starting from a subset of quality requirements. Their approach relies on a quantitative assessment of the impact of architectural tactics on quality requirements, in the one hand, and the impact of incorporating these tactics in architectural styles, in the other hand. In [16], the approach used for the analysis of the relationships between styles and quality is based on a case study and an evaluation by experts , while in [12] it is based on the analysis of the generic structures and behavior of tactics and styles. A similar to latter approach is used in the authors' paper to analyze the UIMSs architectural styles in supporting unit operations.

The authors' work differs from [12] in that it focuses on the evaluation of UIMSs architectures. Besides, it introduces unit operations in the evaluation process. Moreover, while [16] consider only one quality attributes, and [12] consider two quality attributes in the evaluation, this paper considers six quality attributes in analysis of the proposed method.

7. CONCLUSION AND FUTURE WORK

In this paper, the authors proposed a quantitative approach to evaluate the effect of selecting UIMSs styles against selected set of quality attributes. This approach relies on a quantitative evaluation of the effect of architectural unit operations on quality attributes, in the one hand, and the effect of incorporating these unit operations in architectural styles on the other hand. The authors illustrate the approach using four well-known UIMSs styles and evaluating their support for Scalability, Modifiability, Portability, Performance, Reliability and Reusability quality attributes. The authors believe it is a key step towards selecting a suitable user interface architectural design style.

In the future, the authors plan to extend the evaluation approach considering the trade-offs among quality attributes. Furthermore, they plan to improve the stated analysis and results in this paper by considering sub-characteristics of quality attributes. In addition to that, the numerical value assigned to architecture style regarding each unit operation should be proved by experts. The authors also plan to use other evaluation techniques such as aggregation methods, Analytical Hierarchy Process (AHP) and fuzzy integral.

8. REFERENCES

[1] Kasik, D. J., "A user interface management system. Computer Graphics", 16(3), July, 1982.

[2] Kazman, R. and Bass, L., "Toward Deriving Software Architectures From Quality Attributes", 1994.

[3] Dix, A., Finlay, J., Abowd, G. and Beale, R., "Human-Computer Interaction", 3rd editions, Pearson Education, 2004.

[4] Joëlle Coutaz, (2001), "Software Architecture Modeling For User Interfaces".

[5] Kazman, R. and Bass, L., "Software Architectures for Human-Computer Interaction: Analysis and Construction", Submitted to ACM Transactions on Human-Computer Interaction, 1996.

[6] Pfaff, G. and Ten Hagen, P.J.W. "Seeheim workshop on User Interface Management Systems" (Berlin), Springer-Verlag, 1985.

[7] Kazman, R., Bass, L., Abowd, G., and Webb, M., "SAAM: A Method for Analyzing the Properties of Software Architectures." Proceedings of ICSE-16, Sorrento, Italy, May, 1994, 81-90.

[8] Krasner, G. E. and Pope, S. T., "A cookbook for using the model-view-controller user interface paradigm in Smalltalk", -80. JOOP, 1(3), August, 1988.

[9] Lewis, S., "The Art and Science of Smalltalk", Hewlett-Packard Professional Books, Prentice Hall, Hemel Hempstead, 1995.

[10] Coutaz, J., "Pac, an object oriented model for dialog design", In H. J. Bullinger and B. Shackel, editors, Human–Computer Interaction – INTERACT'87, pages 431–6. North-Holland, Amsterdam, 1987.

[11] ISO/IEC 9126-1, "Software engineering —Product quality — Part 1: Quality model", 2001.

[12] Kassab, M., El-Boussaidi, G., and Mili, H., "A Quantitative Evaluation Of The Impact Of Architectural Patterns On Quality Requirements". Springer's Studies in Computational Intelligence Book Series, Volume 377, pp. 173-184, 2011.

[13] Pfaff, G., "User Interface Management Systems. New York: Springer-Verlag, 1985.

[14] Dijkstra, E.W., "The Structure of the 'THE' Multiprogramming System." Communications of the ACM 11, 5 (May 1968): 341-346.

[15] Rochkind, M.J., "An Extensible Virtual Toolkit (XVT) for Portable GUI Applications," pp. 485-494. Digest of Papers, COMPCON, San Francisco, CA: Thirty-Seventh IEEE Computer Society International Conference, February 1992.

[16] Bode, S., and Riebisch, M., "Impact Evaluation for Quality-Oriented Architectural Decisions Regarding Evolvability", The 4th European conference on Software architecture, pp.182-197, 2010.

[17] Galster, M., and Eberlein, A., Moussavi, M., "Systematic selection of software architecture styles". IET Software, 2010, Vol. 4, lss. 5, pp. 349-360.