

# ATTRIBUTE-BASED TECHNIQUES TO EVALUATE ARCHITECTURAL STYLES FOR INTERACTIVE SYSTEMS<sup>1</sup>

**Losavio F., Chirinos L.**

Laboratorio LaTecS - Centro ISYS  
Universidad Central de Venezuela  
Ap. 47567, Los Chaguaramos, 1041-A, Caracas, Venezuela  
{flosavio, lchirino}@isys.ciens.ucv.ve  
Phone: 58-2-6052293

**Pérez M.**

Laboratorio LISI - Departamento de Procesos y Sistemas  
Universidad Simon Bolivar  
Apartado Postal 89000, Caracas, Venezuela  
movalles@usb.ve  
Fax: 58-2-9063303, Phone: 58-2-9063314

## ABSTRACT

This paper addresses the problem of evaluating architectural styles or patterns with respect to specific quality characteristics, in order to make a selection for a particular domain, using attribute-based techniques. The domain of interactive systems, where the graphical user-interface plays an important role and where the adaptation (modifiability) is a crucial aspect of the architectural design, will be studied here. In this work the ABAS (Attribute Based Architectural Style) technique, formulated by Klein and Kazman, is used to evaluate a multiagent-based architecture with respect to the modifiability quality attribute. Moreover, the ABAS structure of the information on the style is extended in this work considering the ISO 9126 quality model of the domain, customized in our case for interactive systems. The PAC (Presentation, Abstraction, Control) ABAS is defined as a specialization of the Mediator ABAS. The ISO quality model as part of the ABAS technique, systematizes and helps to establish the qualitative and quantitative reasoning for a better understanding and determination of the measurable properties and their dependencies on the expected or predictable quality requirements of the final system.

**KEYWORDS:** architectural style, architectural pattern, attribute-based evaluation, ABAS, architectural design, quality model, interactive systems, PAC, ISO 9126

## INTRODUCTION

The goal of this paper is the application of a software quality model-based technique for evaluating styles or patterns characterizing the architecture of a particular problem domain or context with respect to non-functional characteristics, required or predictable in the final software system. In this context, evaluation of an architectural style or pattern means to capture and measure essential properties of the architecture affecting the overall final system behavior, in order to establish sound criteria for selecting the style or pattern. An architectural style [1], or architectural pattern [6], provides a skeleton for components or for the global architecture of the system. In what follows we

---

<sup>1</sup> This work is a result of the CEE INCO SQUAD project EP 962019 and the CDCH ARCAS project 03.13.4584.00 of the Universidad Central de Venezuela

will consider an architectural style or pattern as a general description of the pattern of data and interaction among the components. An informal description of the benefits and drawbacks of using the style is also provided [6], [15]. A component of the style may be a design pattern, in the sense of [10].

We are interested in the interactive systems domain, where the graphical-user interface (GUI), plays an important role. The technique that will be used for evaluating architectural styles is based on quality models. These models are established to determine the influence of the system internal properties, such as extensibility or reusability, on the external non-functional characteristics, such as maintainability, which are properties observed in the final software system. The external quality characteristics, defined as part of the software initial non-functional requirements, in general cannot be directly measured. They are refined into sub-characteristics, which can be measured by quantifiable entities or attributes such as time or size, through all the stages of the software development process (the design or the implementation phase for example). It is agreed that these characteristics affect mostly the software system structure or architecture. Important approaches based on quality models have been developed: ISO 9126 [12], Dromey [9], [3] and ABAS (Attribute-Based Architectural Styles) [15]. These approaches share the fact of considering quality models related to the final product or intermediate products (deliverables) obtained in the early stages of the software development process.

The problem of establishing the quality of software architecture is, in general, not an easy one. As we will see below, the non-functional requirements elicitation is considered in several methods that include architectural design as a stage in the software development process [16], however, they do not consider explicitly a particular approach for constructing the quality model. Actually, there are very few approaches to explicitly handle the conflicts in quality requirements during the architectural design stage [4], [15], [5], [13]. Consequently, the lack of a supporting method or systematization drives to design software architectures in an *ad-hoc*, intuitive, experience based manner, with the consequent risk of unfulfilling some of the system properties.

Few traditional software development methods deal explicitly with quality architectural design. New methods are arising. The Jan Bosch method [5] considers the design of software architectures taking account of the quality requirements from the early stages of development. The architectural design process, seen as an optimization problem, is viewed as a function taking as input the functional requirements specification and generating as output the architectural design. In the first step, a first version of the architecture is produced, not accounting of the quality requirements. Then, this design is evaluated with respect to the quality requirements. Each quality attribute is given an estimated value. These values are compared with the values of the quality requirement specification. If all the values are as good or better than required, the architectural design process is finished. Otherwise, a second step transforms the initial architecture, during which, quality value for some attribute improves. This design is again evaluated and the same process is repeated, if necessary, until all quality requirements are fulfilled or until the software engineer decides that there is no feasible solution. In this case the software architect needs to renegotiate the requirements with the customer. Each transformation (quality attribute-optimizing solution), generally improves one or some quality attributes, affecting others negatively.

Another method, ATAM (Attribute Tradeoff Analysis Method) [13], is similar to the one formulated in [5]. It is based on the previous SAAM (Software Architectural Analysis Method) [14] and proposed as a technique for understanding the tradeoffs inherent in architecture evaluation. The method provides a way to evaluate software architecture's fitness with respect to multiple competing quality attributes. Since these attributes interact, the method helps to reason

about architectural decisions that affect quality attribute interactions. The ATAM is a spiral model of design, postulating candidate architectures followed by analysis and risk mitigation, leading to refined architectures. The technique used for helping the reasoning is based on the ABAS.

Both methods are quite similar. However, one of the major differences between these approaches is that Bosch [5] includes concrete guidelines on how to transform or refine the architecture in order to meet the quality requirements. ATAM, does not provide guidelines for refinement, concentrating instead more on the identification of the tradeoff points, e.g. design decisions that will affect a number of quality attributes. Nevertheless neither the ATAM nor the Bosch method propose a particular approach to construct a global quality model, which is one of our main goals.

In this work we have applied the ATAM's ABAS technique to identify the relevant quality attributes, in order to evaluate the fitness of the proposed architectural style. We have selected ABAS because it is directly related with the evaluation of architectural styles, which are reusable design components, introducing the *attribute quality model* notion to characterize the style and a reasoning framework to analyze the corresponding quality aspects. ABAS considers only one attribute at a time. If several attributes have to be studied, the ABAS technique is reapplied. However, the ABAS approach does not precise how to construct the quality model, or how to exactly capture quantitative aspects. In this sense, we have extended the ABAS technique by constructing a quality model using the ISO 9126 approach, to improve the elicitation of non-functional requirements. This model offers a global and better view of the quality requirements for the domain of interest, and it enriches the ABAS analysis with better quantitative considerations.

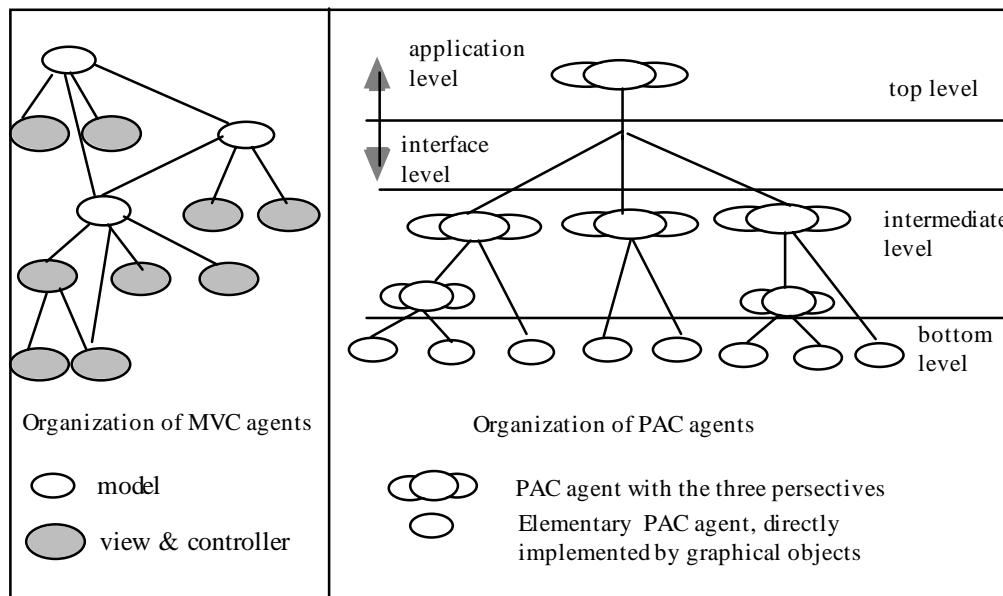
It is of general agreement in quality models approaches and in architectural design issues, that the context is relevant to the architectural decisions, because it may greatly affect the system's initial requirements. Architectures are responses to specific quality requirements [2]. We have chosen interactive systems as our context or problem domain, because, on one hand it is a required part of modern software applications. On the other hand, we have experience in designing efficient architectures for these systems [19],[20]. Interactive systems allow a high degree of user interaction through their GUI, enhancing the *usability* of the application, in the sense of providing a convenient access to their services, allowing the user to spend less time learning the application, producing results more quickly [8]. The architecture of OO interactive systems must reflect the paradigm of the separation between the abstract or semantic aspects (functional core) of the system and its presentation. Usually the core does not change much in time, remaining relatively stable. GUI, however, is exposed to frequent changes and adaptation. The system's architecture must support the *modifiability* of the GUI without causing major effects to the application specific functionality or the underlying data model. For example, systems may have to support different GUI standards, customer-specific "look and feel" metaphors, or interfaces that must be adjusted to fit into a customer's business processes. Moreover, *reliability* in interactive systems means that they must be *robust*, in the sense of recovering from all the possible exceptional situations (errors or exceptional conditions). Usability, modifiability and reliability are non-functional characteristics that are required in the final system and the architecture of an interactive system should guarantee these aspects.

Besides this introduction and the conclusion, this paper is structured in four main sections: the first one describing the main general requirements for the architectural styles for interactive systems. The second section describes the ISO 9126 approach to quality model; this generic model is customized for interactive systems. The third section presents the attribute-based architectural styles (ABAS) technique. Finally the last section contains the definition of the ABAS for the PAC

style, the main result of this work. The qualitative and quantitative aspects are analyzed using the considerations derived from the ISO quality model with respect to the modifiability attribute.

### ARCHITECTURAL STYLES FOR INTERACTIVE SYSTEMS

Architectural styles for interactive systems are based on the *data indirection* principle, for separating semantics from presentation aspects. PAC (Presentation, Abstraction and Control) [1] and the well known MVC (Model-View-Controller) [6], [11], [10] multiagent models are used. Interface agents, with the abstraction, control and presentation perspectives, constitute the PAC model. PAC differs from the MVC model in the sense that the PAC presentation encapsulates the view-controller pair of MVC for capturing and treating external events. In PAC, an additional control notion is introduced for communicating the abstraction with the presentation perspectives. Moreover, communication among the PAC agents is only allowed through their respective controls, and a tree-like hierarchy of agents is established. Subagents (as for example a window with menu and palette areas) constitute an agent.



**Figure 1. Architectures based on multiagent models**

The top-level agent, at the highest level in the PAC hierarchy (see right side of Figure 1), provides the functional core or semantics of the system, or for example, the communication with the system's repository to store and retrieve persistent data. Often, this agent does not provide a presentation perspective, or this is reduced to just an icon representing the whole system. The other PAC agents depend or operate on this core. The bottom level agents represent self-contained semantics, which may be directly implemented by elementary toolkit graphic objects. The agents on the intermediate level represent in general composite interface objects that are a combination of other composite agents or elemental agents. Figure 1 below shows both architectures. Notice that the hierarchy for MVC is represented by the nested views. Models may indistinctly communicate among them.

In general, the MVC model, provides a system architecture that is usually more efficient than the PAC architecture, in the sense that the communication between PAC agents, that are hierarchically organized, may impact system efficiency. Moreover, the granularity of the design must be taken

into account to keep small the number of elementary PAC agents. However, PAC may support multitasking and task specific user interfaces better than MVC does. MVC does not separate self-reliant subtasks, into cooperating but loosely coupled agents. Moreover team's work, system prototyping and extensibility is favored using the PAC architecture, because it provides clear separation of concerns between different system tasks. In [21] it is shown that the PAC architecture can easily be extended to distributed client/server architecture for OO systems. The *Mediator* design pattern [10] characterizes the PAC style, for modeling its control perspective [19]. MVC instead is characterized by the *Publisher/Subscriber* [6] design pattern, also known as Subject/Observer [10]. The model follows a publisher pattern, and the view-controllers are the subscribers.

From the above discussion, we have seen that both MVC and PAC styles are used to architecture interactive systems. Moreover, MVC is being employed as a reusable framework in commercial graphical toolkits. PAC instead, has a major theoretical importance since it greatly favors the independence of communications among the agents, without losing their control, and it seems very well adapted to be used to architecture distributed applications with multitasking. For this reason and for abridging our presentation, we have chosen to characterize only the PAC style in this work. The same approach can be applied for MVC.

As we have pointed out in the Introduction, a software system architecture is defined in terms of computational elements and their interactions, with the topology and structure of the system. The quality of an architecture is expressed by its structure and by the distribution of its functionality. A software system is in general constituted by several structures that define the global system structure. A complete architecture has many dimensions or *views*. Sometimes structure and view [16] are considered as synonyms. The non-functional characteristics affect mostly the system architecture and they are considered early, at the moment of formulating the requirements for the system. The global view of the non-functional characteristics is given by a quality model of the system, which concerns a family of systems in a particular domain. In the following section the quality model, according to ISO 9126 [12], will be instantiated with the quality characteristics of multiagent model-based interactive systems.

### **ISO 9126 QUALITY MODEL FOR INTERACTIVE SYSTEMS USING THE PAC STYLE**

The quality characteristics of the ISO 9126 quality model (*Functionality, Reliability, Usability, Efficiency, Maintainability, Portability*) [12], have to be customized and refined to construct the quality model, according to the domain or application. In this model the external and internal quality characteristics are refined into attributes, which are the measurable entities. Figure 2 shows the customization of this model for interactive systems using multiagent model-based architectural styles. In particular, the PAC (Presentation, Abstraction, Control) style will be characterized through this work [1], [6], [19].

The non-functional requirements for interactive systems, as we have pointed out in the Introduction, are *Reliability, Usability* and *Maintainability*, which are expressed as *external* characteristics, observable by the end-user on the final system. Since functionality or functional conformity is an obvious requirement that must be present, it is not shown in this model, in order to abridge the presentation. An interactive system is meant to be reliable, in the sense of *Robustness* or capable of recovering from wrong user's inputs, usable in the sense of being easy to learn and friendly for the user, maintainable since the GUI is exposed to frequent changes. *Reusability, Instanciability* and *Abstraction* are *internal* sub-characteristics of *Reliability*, because they are observable during the software development process. The system is constructed using reusable

design components, GUI frameworks and reusable toolkit libraries. The reusable frameworks are instantiated for each GUI agent. Abstraction can be measured in terms of its internal attributes, such as *size*, with usual object-oriented metrics. Usability is refined into *Understandability*, *Learnability* and *Operability*, which are also external characteristics, observed at run time. They are further refined into *Complexity*, which is an internal sub-characteristics that is directly measured by the *size* attribute, which affects the *time* external attribute for Usability (time spent to learn the GUI functionality for Learnability, for example). Maintainability is refined into *Reusability*, *Modifiability* and *Extensibility*, which are internal sub-characteristics. Modifiability is considered a sub-characteristic of maintainability, since the property of being modifiable concerns changes on existent entities, differing from extensibility, in the sense of considering the addition of new entities in order to satisfy new requirements. On the other hand, reusability as a sub-characteristic of maintainability is refined into *Modularity* and *Flexibility*. Modularity affects reusability in the sense that a decoupled entity with high cohesion, for example, is easier to be reused. In this sense, *Cohesion* is taken as the internal attribute for Reusability, and it is measured by standard OO metrics. Flexibility considers the adaptability property in particular contexts or domains and it is measured by the *coupling* attribute. In this work, we will consider only Modifiability, refined further into *Complexity* and *Coupling*, also internal sub-characteristics. Complexity affects modifiability, but it is also affected by coupling. Low coupling reduces complexity, hence modifiability improves. The measurable attributes for modifiability affect also the *time* external attribute for Maintainability (time spent for modifying some functionality for example). Complexity is also measured by the attribute *size*, considering for example the depth of the tree, the average number of agents of the superior level in the PAC hierarchy. Coupling is measured by the internal attributes *Fan-in/Fan-out*. Notice that in this model, the same attribute can be measured in different ways [12], [7], [18], depending on the kind of deliverable of the specific development stage.

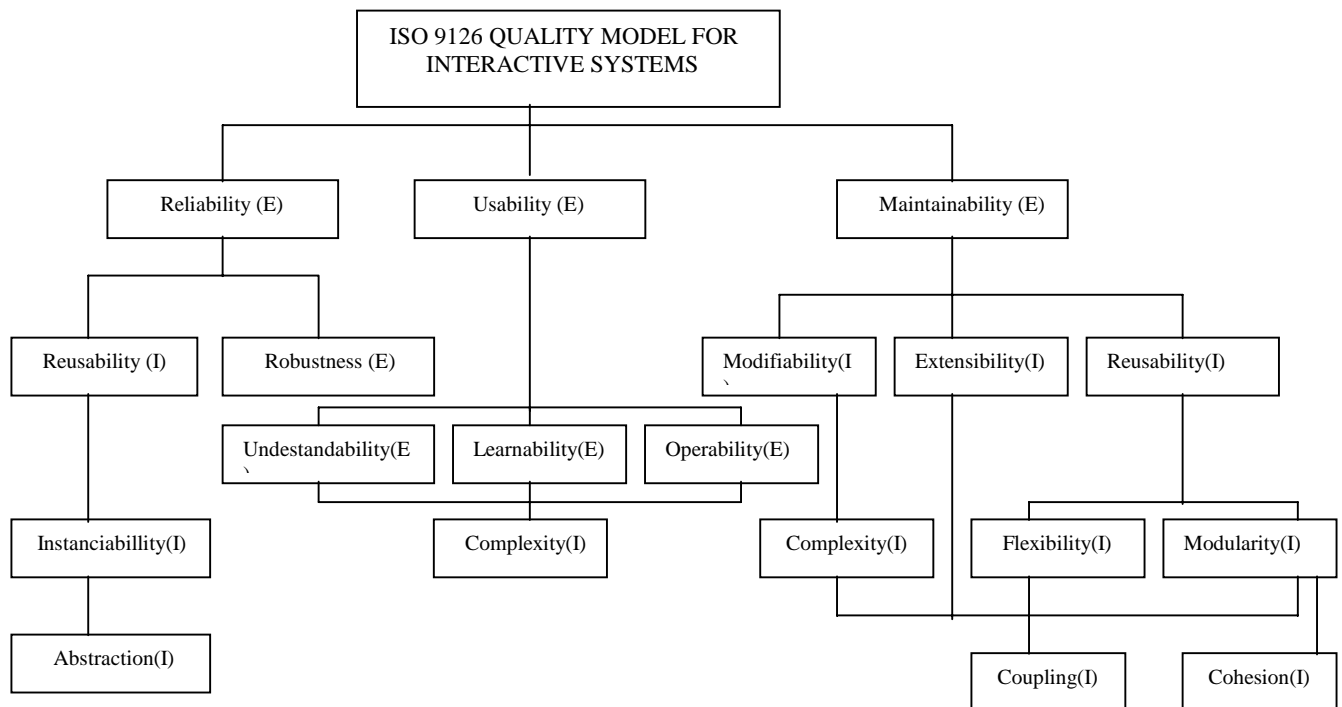


Figure 2. Quality Model for Interactive Systems based on ISO 9126

The complete analysis of the ISO 9126 quality model allows to reason quantitatively about all the quality characteristics desirable or required in the final software system. We have selected Modifiability as a relevant internal property in the interactive systems domain, and its measurable attributes have been shown in Figure 2. The fact of guaranteeing a highly modifiable architecture, enhances a highly maintainable final software system.

### **ATTRIBUTE-BASED ARCHITECTURAL STYLES**

An architectural style or architectural pattern [6], [22] provides a skeleton for components or for the global architecture of the system. It includes a description of the component types and their topology, the pattern of data and control interaction among the components. An informal description of benefits and drawbacks and possible uses of the pattern is also provided. In some design patterns [10], which according to Buschmann, are components of architectural patterns, a description of a possible implementation in a target language is also given. Some formal descriptions of the style provide also an invariant condition, to provide sufficient conditions for a formal design description to be an instance of the architectural pattern. These conditions may be mechanically tested [17]. Architectural styles or patterns are important artifacts because they define classes of designs along with their associated known properties. They offer experience-based evidence of how each class has been used historically, along with a qualitative reasoning to explain and justify their specific properties. An example of this kind of reasoning is “use pipe and filter style when reuse is desired and performance is not a top priority”. Architectural styles are powerful because they provide the user with previous and tested knowledge of experienced designers for reuse. Nevertheless, the application of the style to a particular problem or domain is not straightforward because the criteria for the classification and comparison of styles are not clear enough and the descriptions of the styles are imprecise and generally informal.

The notion of Attribute-Based Architectural Style (ABAS) [15] is conceived to make architectural styles the foundation for more precise reasoning about architectural design. This is accomplished associating a *reasoning framework* (quantitative or qualitative) with the *description* of an architectural style. The reasoning framework is based on the establishment of a *quality model* specific to a quality characteristic, called *attribute* in this approach. Only one attribute at a time is considered when ABASs are used in design or analysis, because ABAS is associated with only one attribute reasoning framework, called an *attribute model*. For example, if an architectural style is interesting from both a performance and a reliability point of view, it would be motivation for creating the respective performance and reliability ABASs. The authors claim that using ABASs is a step in moving architectural design closer to being an engineering discipline. Design and analysis of software architecture is based on reusable design components: reusing known patterns of software components with predictable properties. The information for characterizing an ABAS quality attribute is divided into three categories: - *External Stimuli* that causes the architecture to respond or change. - *Responses*, that are quantities measured or observed in the requirements or attributes desirable in the architecture. - *Architectural decisions* that are aspects (components and connectors) and their properties, characterizing the style, that have a direct impact on achieving attribute responses. The main purpose of every ABAS is to organize consistently the existing specialized body of knowledge in each of the quality attributes communities. This knowledge can be reused in every ABAS related to a particular quality attribute. Table 1 shows the four parts of the ABAS structure:

<b>Structure</b>	<b>Description</b>
1. <i>Problem description</i>	<i>Informal description of the design and analysis problem that the ABAS is intended to solve, including the quality attribute of interest or whose presence is desirable in the architectural style, the context of use, constraints and relevant attribute-specific requirements.</i>
2. <i>Stimulus/Response attribute measures</i>	A characterization of the stimuli to which the ABAS is to respond and the quality attribute measures of the response. Construction of the <i>quality model for the attribute</i>
3. <i>Architectural style</i>	Description of the architectural style in terms of its components, connectors, properties of those components and connectors, and pattern of data and control interactions (their topology) and any constraints on the style. Description of the <i>architectural decisions</i> .
4. <i>Analysis</i>	Description of how the quality attribute models are formally related to the architectural style and the conclusions about “architectural behavior”. <i>Establishment of the links</i> or tradeoff points, between the quality characteristics required and the measured properties affecting them. A reasoning and analysis and design heuristics are formulated

**Table 1. The ABAS Structure**

The ABAS structure is similar to those proposed in the catalogues of architectural styles [22], [6], with respect to Part 1 and 3 of Table 1. The main difference consists in adding explicitly the information on the characteristics of the quality attribute relevant to the particular style, expressed in Part 2 of Table 1. These are the measures of the responses and constitute the quality model for the specific attribute. Moreover, Part 4 of the structure, analysis, is used to establish the link between the quality model of the attribute, and the measures of the attribute. The aspects discussed in Parts 2, 3 and 4 constitute the reasoning framework for establishing the quality characteristics of the architectural style.

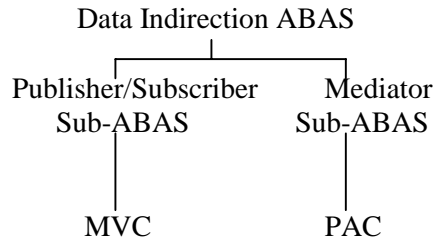
From the above discussion, an ABAS is seen as a reusable design component, providing a quality model for a specific characteristic which is predictable in the context of the application where the particular ABAS will be used. For example, if the modifiability attribute is required, all the ABAS using different forms of data indirection could be analyzed according to the structure of Table 1.

Our extension to ABAS consists in adding, as the starting point to apply the ABAS information framework shown in Table 1, the ISO 9126 quality model customized to the problem domain or context of use. In this way, the problem description is enriched with the global view of the non-functional requirements of the domain, and the quality model for the attribute in part 2 will be taken directly from this ISO customized model. An approach such as that described in [4] and applied in [7] to the domain of the interactive systems, could be used to derive exact measures for the attributes.

## ABAS FOR THE PAC STYLE

Many ABASs are intended to achieve similar goals. For example, modifiability ABASs use various forms of indirection. So they fall naturally into groups, called ABAS families. Multiagent models, used to architecture the GUI component of interactive systems, belong to this ABAS family. MVC is characterized by the Publisher/Subscriber (Subject/Observer [10]), which is defined as a sub-ABAS of the Data Indirection ABAS [15]. The control perspective of the PAC style can be also modeled as a Data Indirection ABAS. In consequence, the Mediator design pattern characterizing the PAC style, is considered a sub-ABAS of Data Indirection. In order to construct an ABAS for the PAC style, enhancing for example, *modifiability*, we must consider the most relevant architectural styles belonging to the family, and other styles possibly involved. For PAC we consider the Data Indirection and the Mediator as members of the same family. In what follows, we will describe these ABASs, according to the approach of [15]. Only Mediator and PAC ABAS will be treated in details. For the Data Indirection, only the Problem Description and Architectural Decisions will be presented. Further details can also be seen in [15]. Figure 3 shows the ABASs hierarchy characterizing the multiagent model architectural styles for interactive systems.

Notice that a similar approach can be followed to study the MVC style, discussing the Publisher/Subscriber ABAS with respect to the modifiability attribute. In order to abridge this presentation, only the PAC style will be treated here. For the description of the ABASs, we will follow the structure shown in Table 1.



**Figure 3. ABASs hierarchy for interactive Systems**

### Data Indirection ABAS

#### *Description of the Problem*

In the context of interactive systems, the producers and the consumers of shared data are kept from having knowledge of each other's existence and the details of implementation. This is accomplished by interposing an intermediary, a component and/or protocol, between the producer and the consumer of shared data items. The general principle is that reducing the data or control coupling between distinct components enhances the modifiability. Having the intermediary coupled to both the producers and the consumers reduces coupling, and hence having them decoupled from each other.

#### *Architectural style*

Table 2 shows the values of the architectural parameters corresponding to the architectural decisions for Data Indirection, with respect to modifiability.

<i>Modifiability Architectural parameters</i>	
<i>Name</i>	<i>Value</i>
Topology	star
client knowledge of data schema	complete knowledge
activeness of repository	passive

**Table 2: Architectural Decisions for the Data indirection style**

For the complete description of the Data Indirection ABAS, the reader is referred to [15].

**The Mediator Sub-ABAS**

*1. Description of the Problem*

The intent of the Mediator design pattern [10] is to define an object that encapsulates the interaction of a set of objects. Mediator promotes loose coupling by keeping objects from referring to each other explicitly (encapsulation), and let you vary their interaction independently. Consumers and producers are called colleagues. Mediator is a distinguished colleague. It favors the communication among colleagues that do not know each other, but only their Mediator; therefore the number of interconnections is reduced.

*1.1 Criteria for Choosing this sub-ABAS*

Mediator must be selected if the following conditions are satisfied:

- Colleagues do not know each other
- A colleague only knows the mediator
- Mediator knows all its colleagues
- There is no coupling among colleagues
- There are no dependencies cycles among colleagues
- Mediator is coupled with its colleagues

*2. Stimuli/Responses for Modifiability*

Stimuli: attach/detach a colleague

Responses: increase/decrease the time used to distribute the message from the Mediator to its colleagues

*3. Architectural style*

Table 3 shows the values of the architectural parameters corresponding to the architectural decisions for Mediator:

<i>Modifiability Architectural parameters</i>	
<i>Name</i>	<i>Value</i>
Topology	star
Size	number of colleagues

**Table 3: Architectural Decisions for the Mediator style**

*4. Analysis*

The measure of the complexity for modifiability is a function of the coupling. In this case it is defined as the required effort to attach or detach a colleague, which only implies to modify the intermediary or Mediator. The unit of measure is *time*.

*4.1 Reasoning*

It seems that to attach or detach a colleague requires almost the same effort. Nevertheless, to attach a new colleague should imply a greater effort, in the sense that the data corresponding to the new colleague must be registered with the Mediator (Mediator knows its colleagues).

*4.2 Heuristics of analysis and design*

Modifiability problems related to the style with respect to the overall performance of the system:

- To allow direct communication among colleagues would affect:
  - Usability: performance improves
  - Modifiability: coupling increases; complexity increases
  - Extensibility: coupling increases; complexity increases

Notice that the considerations about modifiability and extensibility are the same. In this sense, extensibility could be considered a sub-characteristic of modifiability in the quality model shown in Figure 2. Nevertheless we have preferred to leave them separate to increase the model's flexibility.

## The PAC Sub-ABAS

### 1. Description of the problem

The PAC style has been described in the previous section on architectural styles for interactive systems.

#### 1.1 Criteria for Choosing this ABAS

PAC can be selected if:

- The agents follow the Mediator style
- The agents are hierarchically organized (tree-like organization).

#### 2. Stimuli/Responses for Modifiability

*Stimuli:* The same as for Sub-ABAS Mediator: attach/detach a new agent

*Response:* Number of affected agents

#### 3. Architectural style

Table 4 shows the values of the architectural parameters corresponding to the architectural decisions for the PAC style.

<i>Modifiability Architectural parameters</i>	
<i>Name</i>	<i>Value</i>
Topology	tree
Size	depth of the tree

**Table 4: Architectural Decisions for the PAC style**

#### 4. Analysis

The measure of the complexity for modifiability is a function of the depth of the tree (number of levels). It is similar to the Layering ABAS [15]. Coupling is a function of the number of connections among the PAC agents (Fan in/out).

#### 4.1 Reasoning

Complexity is measured in two ways:

- If an agent is attached, the measure will be the effort, measured as a function of the time spent in the communication (as in Mediator)
- If an agent is detached, the measure will be the number of affected agents. To detach an agent, the agents belonging to the sub-tree containing the agent to be detached will be also detached, according to usual tree properties. If the depth of the sub-tree is great, the complexity will increase, so the modifiability will be affected.

The detachment of an agent implies that the agents of its sub-tree are also detached. If the tree levels or depth increases, the complexity of detaching an agent will also increase, hence modifiability decreases.

#### 4.2 *Heuristics of analysis and design*

The detachment of an agent can be realized only if its sub-agents are retired. The same reasoning for the Mediator sub-ABAS can be applied in this case. The tree depth is a crucial parameter. As we have pointed out for Mediator, if direct communication among the agents is established, performance improves, but modifiability will be affected, because coupling increases. Every agent in the tree must be notified of the detachment of the agent. In this case, the PAC agent's control should behave like a Publisher/Subscriber to improve modifiability.

### **CONCLUSION**

In this work we have used the ABAS information structure, and we found it very useful to have at hand a record of the specialized knowledge of a software designer to reason on the quality properties of an architectural style, in a particular domain. However, the reasoning used in ABAS is in general a qualitative reasoning and it is limited to one attribute at a time. In our approach, the ISO 9126 model, defined before using the ABAS structure, has facilitated and accelerated the ABAS analysis stage, enriching also the ABAS information structure. The ISO 9126 model offers a global view of quality, including the attribute of interest, which is analyzed using the ABAS structure. This quality model, customized for the domain of interactive systems [7], helps to elicitate non-functional requirements or external characteristics. These quality requirements are referred in the ABAS approach as predictable behavior. The three external characteristics, Reliability, Usability and Maintainability, are refined into sub-characteristics, until the measurable entities are reached. In this way, the dependencies or tradeoffs among these internal characteristics and the global quality goals can be studied. Moreover, it can be established qualitatively and quantitatively if the final system meets the expected quality requirements, allowing a holistic vision of the quality aspects related to a particular domain. In particular, we have used the information gathered by the ISO quality model to enrich the ABAS structure, to characterize the PAC architectural style. For concluding we strongly recommend to construct an ISO 9126-based quality model on the domain of interest, before applying the ABAS structure, in order to have a global and better view of the quality attributes of an architectural style and improve the analysis part. The use of notational standards, as the UML (Unified Modeling Language) [23] for characterizing architectural styles for particular domains using the extended ABAS technique proposed in this paper, is an ongoing research. Moreover, we are now studying the possibility of applying an ISO-based approach, such as [4], to complement the analysis step of the ABAS structure, to quantitatively establish the links between the quality characteristics required and the measured properties affecting them. This approach could be easily included as a step in object-oriented software process development frameworks, such as the Rational Unified Process [16], towards a quality architectural design process.

### **REFERENCES**

1. Bass L., Coutaz J, "Developing Software for the User Interface", Addison Wesley, SEI series, 1991.
2. Bass L., Clements P., Kazman R. "Software Architecture in Practice", Addison Wesley, 1998.
3. Bansija J., Davis C. "An Object-Oriented Design Assessment Model", PHD Dissertation, University of Alabama, September 1997.
4. Bøegh J., DePanfilis S., Kitchenham B., Pasquini A. "A Method for Software Quality Planning, Control and Evaluation". IEEE Software, 69-77, March/April 1999
5. Bosch J. "Design and Use of Software Architecture", Addison Wesley, Harlow, England, 2000

6. Buschmann F., Meunier R., Rhonert H., Sommerlad P., Stal M. "Pattern-Oriented Software Architecture. A System of Patterns", John Wiley & Sons, 1996.
7. Chirinos L., "Evaluación de la calidad del software en un proceso orientado a objetos", Magister Thesis, Posgrado en Ciencias de la Computación, Universidad Central de Venezuela, July 1999.
8. Collins D., "Designing Object-Oriented User Interfaces", Benjamin/Cummings Publishing Company, Inc. 1995.
9. Dromey, R., "Cornering the Chimera", 33-43, IEEE Software Vol. 13, No.1, January 1996.
10. Gamma E., Helm R., Johnson R., Vlissides J. "Design Patterns. Elements of Reusable Object-Oriented Software" Addison Wesley Publishing Co., 1995.
11. Goldberg A., "Smalltalk-80 The Interactive Programming Environment", Addison-Wesley 1984.
12. ISO/IEC FCD 9126-1.2: Information Technology - Software Product Quality.Part 1: Quality Model, 1998, draft.
13. Kazman R., Klein M., Barbacci M., Longstaff T., Lipson H., Carriere J., "The Architecture Tradeoff Analysis Method",CMU/SEI-98-TR-008, ESC-TR-98-008, July 1998
14. Kazman R., Abowd, G, Bass L., Clemens P. "Scenario-Based Analysis of Software Architecture", IEEE Software, Nov. 1996, 47-55
15. Klein M., Kazman R., "Attribute-Based Architectural Styles", CMU/SEI-99-TR-022, ESC-TR-99-022, October 1999.
16. Krutchen P. "The Rational Unified Process", Addison Wesley, 1999.
17. Levy N., Losavio F. "Analyzing and Comparing Architectural Styles" Proceedings of the XIX Int. Conference of the Chilean Computer Science Society, (87-95), November 11-13, Talca, Chile, 1999.
18. Losavio F., Chirinos L."Evaluación de la calidad en el desarrollo de sistemas interactivos", (92-108) Proceedings X CITS, Curitiba, Brazil, 17-21, May 1999.
19. Losavio F, Matteo A. "A Method for User-Interface Development", Journal of Object Oriented Programming, Vol. 10, 5 (22-27) 1997
20. Losavio F., Matteo A., Pérez de Ovalles M..A. "Architecture for an Object-Oriented CASE Environment", Journal of Object-Oriented Programming, Vol. 12, No. 6 (49-60) 1999
21. Losavio F., Matteo A. "Multiagent Models for Designing Object-Oriented Distributed Systems ", Journal of Object Oriented Programming, Vol. 13, No. 3 (8-12), June 2000.
22. Show G., Garlan D. "Software Architecture. Perspectives on an Emerging Discipline", Prentice Hall, New Yersey, 1996
23. Rational Software Corporation "UML: Unified Modeling Language', V1.0, Rational Software Corporation, 1997