QUALITY CHARACTERISTICS TO SELECT AN ARCHITECTURE FOR REAL-TIME INTERNET APPLICATIONS[1]

Losavio F., Matteo A., Ordaz Jr. O.
Centro ISYS, Facultad de Ciencias,
Universidad Central de Venezuela
Apartado 47567, Los Chaguaramos 1041-A, Caracas, Venezuela
{flosavio, amatteo, oordaz}@isys.ciens.ucv.ve

Lévy N., Marcano-Kamenoff R.
Laboratoire PRISM
Université de Versailles St.-Quentin,
78035 Versailles Cedex, France
{Nicole.Levy,Rafael.Marcano}@prism.uvsq.fr

Abstract

Performance, security and availability are important non-functional characteristics that must be present in real-time systems. The selection of a convenient architecture is an important step in achieving these quality goals. The use of an appropriate architectural style can simplify architectural design and subsequent software implementation stage. The overall quality goals are influenced by the structural characteristics or topology of the style. However, the problem on the selection of the right architectural styles according to the desired quality attributes is an open issue. The existing approaches lack of a standard and formal notation. They are limited to an informal description and examples of the application of a style. Quality issues are not explicitly considered. The main goal of this work is to propose an approach for the selection of software architectures based on quality characteristics. We present a process integrating the ABAS technique with the ISO 9126 quality model, taking advantage of their complementary strengths. The B formal language is used to formally describe architectural styles and their quality attributes. We describe and discuss an experience obtained in applying this process for the selection of the architecture of a market stock exchange monitoring system. One of the transformations introduces Internet as a communication medium.
Key-words: software architecture, real-time system, quality attribute, ABAS, architectural style

1. INTRODUCTION

Real-time time systems interact directly with electrical and/or mechanical devices, handling external events usually captured by sensors from the environment. They must be prepared to deal with safety-critical situations, which must be handled with strict timing and ordering constraints. They may vary in time and scope, but performance, security and availability are important quality or non-functional characteristics that must be present in such systems, whose failure may involve high costs, such as loss of human life.

An important step towards achieving the quality goals required by a real-time system is the selection of a convenient architecture for the corresponding software system [BCK 98], [BK 99]. Architectural design identifies the key strategies for the large-scale organization of the system under development [Kru 00], [SR 98], [Dou 99]. These strategies include for example, the mapping of a software package to processors, bus and protocol selection, at a quite low level of abstraction. Quality requirements are generally

[1] This research is sponsored by the CEE INCO SQUAD project EP 962019 and the CDCH ARCAS project No. 03.13.4584.00 of the Universidad Central de Venezuela

dealt with by a rather informal process during architectural design. Conventional object-oriented design methods [Rum et al 96], [Jac et al 92], [Kru 00] tend more on achieving the required system functionality, paying limited attention to quality requirements. Implicitly, the use of the object-oriented modeling approach guarantees to some extent the construction of reusable and flexible systems. Hence maintainability and reusability requirements are incorporated to some extent. However, only these quality characteristics are implicitly considered [Bos 00]. It is also of general agreement that the improvement of one quality attribute may negatively influence another one, so there must be a negotiation or tradeoff before building the final system. Otherwise, the inclusion of different quality requirements once the system is built, will be extremely costly. There are very few approaches to explicitly handle the conflicts in quality requirements during the architectural design stage [Bøe et al 99], [KK 00], [Bos 00], [Kas et al 98]. 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.

A method, proposed by [Bos 00], 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), is similar to the one formulated by [Bos 00]. It is proposed by [Kaz et al 98] 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 Attribute Based Architectural Style (ABAS). A quality model for a particular quality attribute is established to help in the selection of a style. An ABAS considers only one attribute at a time. If several attributes must be considered, the ABAS technique is reapplied.

Both methods are quite similar. However, one of the major differences between these approaches is that [Bos 00] method 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.

For the purpose of this work, we have benefited from both approaches. 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. However, since ABAS considers only one attribute at a time, we have used an extended ABAS [CLP 00], defining a quality model involving all the interesting attributes, according to the ISO 9126 model. In this way we have a global and better picture of all the involved quality attributes. On the other hand, we have used a formal approach based on the B language [Abr 96], similar to the transformation approach followed by [Bos 00], to formally justify the selection of the style and related patterns.

In what follows we will consider an architectural style [GS 96] or architectural pattern [Bus et al 96] 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 [Bus et al 96], [KK 99]. A component of the style may be a design pattern, in the sense of [Gam et al 95].

The main goal of this work is to present and discuss the experience obtained in applying the ABAS (Attribute-Based Architectural Style) technique [KK 99], for the selection of the architecture of a market stock exchange monitoring system. This application is considered a soft real-time problem, in the sense that some of the events may miss their deadline, without affecting the whole system's behavior. The transformation process that undergoes architectural design is formally described by means of the B language. One of the transformations introduces Internet as a communication medium.

The structure of this paper is as follows. The first section introduces real-time monitoring systems. First, the requirements for the stock exchanges monitoring system are described. Then, a quality model is introduced, based on ISO 9126 model. A categorization of architectural styles for real-time systems is subsequently presented. The second section describes the process of selection of the architecture based on quality attributes. The ABAS technique is introduced. The third section illustrates the use of the B language to formally specify architectures with quality attributes. The whole process of applying the presented technique to select the architecture of the stock exchanges system is detailed in section 4. The last section discusses the acquired skills and advantages of the presented approach.

2. REAL-TIME MONITORING SYSTEMS

2.1 Requirements for a real-time stock exchanges monitoring system

The primary goal of a real-time monitoring system is to capture, analyze and broadcast events (data) in real-time. We are interested in *soft* real-time systems, where some of the events may miss their deadline, without affecting the whole system's behavior. The needs of real-time distributed applications running in heterogeneous environments interconnected by wide-area networks, have driven the requirements for an application that will be called CSE (Cyber Stock Exchange). Non-functional requirements for CSE are high availability, platforms heterogeneity, distribution of clients, reliable information with strict deadlines. It is known that these characteristics are not independent, and there must be a tradeoff to determine priorities.

The CSE system, as a real-time *data provider*, will monitor small and medium size Latin American stock exchanges for brokers and independent investors. An antenna (*feed server*) external to the system, provides the data (*feed*) to the CSE data server. A feed contains the relevant information of a stock exchange transaction. The clients (*brokers*), distributed in different geographical locations, are subscribed with the data server. When a change on the feed to which a client is subscribed occurs, the feed is broadcasted to him by the data server, according to a strict time delay. Since one of the requirements for the CSE platform is wide-area networks, the time delay will depend on the network structure used to send the information to the clients. The type of service offered depends on this delay.

Type of services offered

A commercial data provider for stock exchanges can be of different types, according to the average delivery time (adt) offered for the delivery of the data feeds to the clients:

- *end of day* data provider. Data are delivered at the end of the day
- *delayed* data provider. Data are sent periodically and only when there is a modification.
- *real-time* data provider. Data are sent each time there is a modification.

CSE will satisfy one of these services.

Non-functional requirements: quality characteristics

The quality characteristics required for CSE are the following: - *Availability*, because the system must not interrupt the service. In case of interruption, important transactions may be lost involving substantial financial loss. - *Efficiency*, because the data must be delivered within the established average delivery time (*adt*) in order to fulfill the service offered. In consequence, high performance must be assured in data transmission. - *Portability*, because the clients which are distributed in different locations, use different development platforms, minimizing the need for changes and adaptations. The programming language used is also involved in this issue.

Availability and efficiency are the most relevant characteristics for CSE.

Efficiency is measured in terms of the number of transactions served each day. It depends on the number of brokers and/or stock exchanges to be served and on the platform used. If more clients are introduced, a hardware with high performance must be considered. Reliability in our case, depends directly on the network (Internet) and the different communication protocols for data transmission; it may affect the availability of the whole system. If the system is not available, the main goal will not be accomplished, hence the system will not conform functionality, so availability is crucial for failure or success. In order to guarantee availability, redundancy of hardware and software must be taken into account and maintenance can also be affected in terms of cost increase. In what follows, a general model for establishing the quality characteristics of real-time monitoring systems will be presented.

ISO 9126 [ISO 98] proposes a generic model, to specify and evaluate the quality of a software product from different perspectives or views, acquisition, development, maintenance. It considers *internal characteristics*, which are related to the software development process and environment and *external characteristics*, observed by the end-user on the final software product. The view of quality, on these bases, can be internal or external, and it is also affected by the stakeholder view in the particular stage of development. An external characteristic can be measured internally, however its name and measure may be different, according to the stage of development. For example, *portability* is an external characteristic according to ISO 9126: we can speak of a portable system, from the point of view of the end-user of the final system. Moreover, the design can be extensible from the point of view of the system engineer in the design phase, we will then speak in terms of *extensibility*. An important issue on software product quality is that the product internal characteristics determine or influence the external characteristics. In order to establish this influence, internal characteristics must be *linked* or related in some way to external characteristics. ISO 9126 define six characteristics that can be subdivided into sub-characteristic, introducing a refinement notion: *Functionality, Reliability, Usability, Efficiency, Maintainability, Portability*. Attributes in the ISO context are the measurable elements of the high level quality characteristics and sub-characteristics.

The generic ISO 9126 model must be customized according to the system’s non functional requirements. Figure 1 shows the ISO model adapted to the quality requirements of real-time monitoring systems, considering reliability as the relevant external characteristic. It considers two main aspects: the arrival of the data to their final destination and the correctness of these data at the moment of displaying them on the client for satisfying the service. In terms of the CSE system, availability is an external sub-characteristic of reliability. If availability cannot be guaranteed, the system is not reliable. Reliability is measured by the percentage of time that the system functions without failures that represent an interruption of the service. Complexity, as its internal sub-characteristic, can be measured registering the interruptions of the system, as the time that the data server is not transmitting the feeds, and the number of clients requiring the services. A great complexity could affect reliability. Coupling is used to calibrate complexity. It is measured in terms of standard OO metrics.

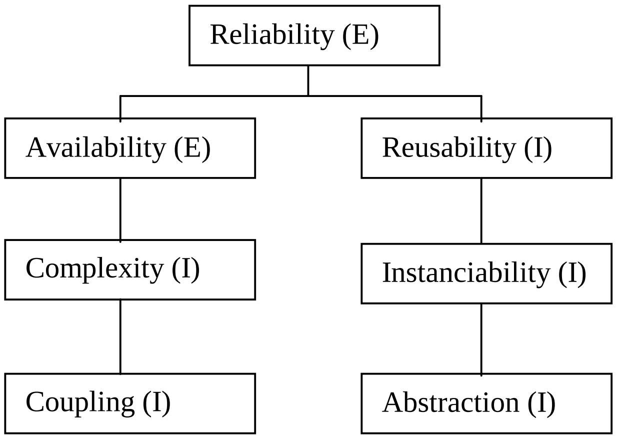


Figure 1. Quality Model for Real-time Monitoring Systems

On the other hand, reusability is an internal sub-characteristic that may also affect reliability. At design level it can be measured using standard OO metrics considering abstraction a sub-characteristics of instanciability.

Efficiency (performance) is an external characteristic measured in terms of the number of transactions served each day. Portability may in turn affect efficiency. They will not be treated here in further details. Usability and Maintainability are not the main concerns for CSE, they neither will be discussed here.

From the above discussion, it can be observed that availability affects directly the functionality or functional conformity of real-time monitoring systems. If the system is not available, the functional requirements will not be fulfilled. In this sense, we have given priority to this characteristic for selecting a convenient architecture for CSE.

2.2 Architectural styles for real-time systems

CSE is a distributed application, so we will be interested only in those architectural styles favoring indirect communication and components decoupling. We will consider the Data Indirection style [KK 99]. This style is characterized by an intermediary (data repository or protocol) between producers and consumers of some shared data. Producers and consumers do not know the data implementation details of the repository and they do not know each other. The design patterns Publisher/Subscriber and Mediator [Bus et al 96] will be studied. The Data Indirection style describes an elemental distributed software system in which producers and consumers communicate through an intermediary component. However, the details on the repository or the protocol associated to the intermediary component remains undefined.

In order to communicate producers and consumers through a specific communication model, we could introduce variants of the intermediary component. As a result, the Publisher/Subscriber pattern is studied. It introduces the synchronization and propagation of changes between the publisher and the subscribers. The Mediator pattern introduces a specialized component (i.e. Mediator) taking in charge the communication between colleagues which differ in their communication protocols.

3. THE ABAS (Attribute-Based Architectural Styles)

The notion of Attribute-Based Architectural Style (ABAS) [KK 99], as we pointed out in the Introduction, 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 the ABAS approach. Notice that the ABAS attribute notion corresponds to the ISO 9126 notion of quality characteristic. 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:

This structure is similar to those proposed in the catalogues of architectural styles [SG 95], [Bus et al 96], 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 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 in our case, if the reliability attribute is required, all the ABAS using different forms of data indirection, which seems to be suitable architecture for distributed systems, could be analyzed according to the framework of Table 1. The complexity of the architecture, expressed by the coupling of the components, has to be taken into account, because we are considering explicitly availability. In this sense, we have extended the ABAS framework [CLP 00], considering the ISO 9126 quality model for a global and better understanding of the quality characteristics of the system. The quality model previously discussed, shows how these characteristics affect the availability of the services offered by the system.

Structure	**Description**
1. Problem description	*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.
2. Stimulus/Response attribute measures	A characterization of the stimuli to which the ABAS is to respond and the quality attribute measures of the response. Construction of an ISO 9126 based *quality model for the attribute.*
3. Architectural style	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 *architectural decisions*.
4. Analysis	Description of how the quality attribute models are formally related to the architectural style and the conclusions about “architectural behavior”. *Establishment of the links* or tradeoff, 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

3.1 Data Indirection

Problem description

This ABAS is characterized by keeping the producers and consumers of shared data from having knowledge of each other's existence and the details of their implementations by interposing an intermediary or protocol between the producer and consumers of shared data items.

Criteria for selecting Data Indirection

It is relevant to anticipate changes in the producers and consumers of data, including the addition of new producers and consumers, if these changes are frequent and it is worth the cost of the modification.

Stimuli/Response for availability

Important stimuli and their measurable controllable responses are:

- Stimuli:
  - add a new producer or consumer of data
  - a modification to an existing producer or consumer of data
  - a modification to the data repository
- Responses:
  - The number of components, interfaces and connections added, deleted and modified, along with the characterization of the complexity of these additions/deletions/modifications

Architectural considerations

The data repository can be a location known by both producers and consumers (e.g. a file or a global data area) or it can be a separate computational component (e.g. a blackboard). The constraint on the repository is that it can hold data. The repository has a data structure, and a set of data types or layout known by all producers and consumers. A single component may be both a producer and a consumer. The producers place their data on the repository because they know the details of the layout; the consumer has a similar behavior for retrieving the data. The management of performance and concurrency control are outside the scope of this style.

Analysis

Redundancy in data producers and data flow channels will increase availability. The dependency on the repository is crucial for availability. In case of failure, a substitute repository must be available.

Architectural parameters for the availability attribute	
Topology	Star
Knowledge of the data layout by client	Complete
Dependency on Repository for producers/consumers	Very high
Redundancy of data producers	High
Redundancy of data flow	High

Table 2. Architectural decisions for Data Indirection

3.2 Mediator

Problem description

Mediator is extensively described in [Gam et al 95], [LL 99]. The intent of the Mediator design pattern is to define an object that encapsulates how a set of objects interacts. 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 if interconnections is reduced.

Criteria for selecting Mediator

Conditions that must be satisfied to select Mediator:

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

Stimuli/Responses for availability

- Stimulus: add a new colleague
- Response: availability of the service increases with time, the number of colleagues (relevant to availability of service)

Architectural considerations

Table 3 presents relevant considerations for Mediator with respect to the availability attribute.

Architectural parameters for the availability attribute	
Topology	Star
Size (Number of colleagues)	High
Dependency on Mediator	Very high
Redundancy of data flows	High

Table 3. Architectural decisions for Mediator

Analysis

Colleagues may be data producers or consumers, indistinctly. Redundancy of colleagues implies the capacity of substituting the mediator for another colleague in case of failure, increasing availability as a function of the time that the service is available.

However, if the number of colleagues increases too much (increase in complexity) the capacity of the Mediator for handling communications could be compromised. In consequence, the availability of the system will be negatively affected, since the direct communication between the mediator and its colleagues could be delayed, increasing the possibility of failures in the data delivery.

In case of CSE, the availability characteristic affects the performance of the system, as a function of the cost of the redundancy mechanisms necessary to provide the required availability level, in a convenient time delay.

3.3 Publisher/Subscriber with Push model

Problem description

It helps to synchronize the state of producers (publishers) and consumers (subscribers) of data. When a producer “publishes” a new data, all the subscribers related to the producer, which require the data, are notified and automatically receive the data. In the case of a push model [Bus et al 96], the producer sends data with the notification only to the interested consumers, reducing the number (complexity) of the communications to the consumers and increasing the performance of the application.

Criteria for selecting Publisher/Subscriber with Push model

Conditions that must be satisfied to select Publisher/Subscriber with Push model:

- The number and identity of data producers and/or consumers are not known or may vary
- The temporal ordering between producers and consumers is not known and undergoes frequent changes

- There are no time constraints related to the amount of data that must be produced and/or consumed. There are no synchronization dependencies between the production and consummation of the data.

Stimuli/Responses for availability

- Stimulus: add a new producer
- Response: increases the availability of the service, measured in terms of the number of transactions executed in a unit of time

Architectural considerations

Table 4 presents relevant considerations for Publisher/Subscriber with push model with respect to the availability attribute.

Architectural parameters for the availability attribute	
Topology	Star
Data persistency	Transitory
Size of the data package	Small
Communication Protocol	Selective broadcasting
Dependencies (from producer)	Very high
Redundancy in data flow	High

Table 4. Architectural decisions for Publisher/Subscriber with push model

Analysis

An adequate redundancy of producers and data flow channels decreases the possibility of failures, increasing availability.

The use of the push model with selective broadcasting communication protocols organized in a star topology, favors performance and availability, considering moderate data packages, as a function of the band width of the communication channel and the number of subscribers.

Availability affects the performance of the system, as a function of the cost of the redundancy mechanisms required. The associated computational infrastructure should have enough capacity and support balanced.

4. FORMALIZING ABAS USING B

The ABAS technique has the advantage of supporting a simple and intuitive description of software architectures. It permits to specify the general structure of a software model at a high level of abstraction and to reason about it. It is useful to understand and document systems, allowing a better communication between developers and customers. However, ABAS lacks of precise semantics and remains inadequate to proof correctness and consistency. Therefore, the resulting specifications can be subject to misinterpretations. ABAS is not sufficient enough to develop rigorous applications that require non-functional properties to be ensured.

The approach presented here integrates the B [Abr 96] formal method with the ABAS technique in a complementary manner. The choice of B offers a perfect opportunity to enhance existing semi-formal descriptions of architectural styles. We use the B formal language in order to balance the semantic weakness of ABAS by a rigorous and precise specification. The B formal specification is used to specify precisely the structure, the behavior and also to measure the non-functional characteristics of an architectural style.

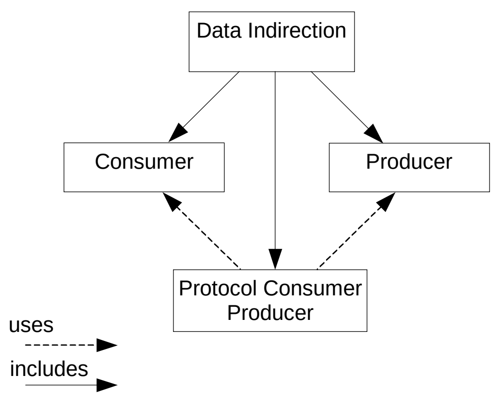
The B formal language is based on the set theory. A B specification is composed of a hierarchy of abstract machines, each one corresponding to a particular component of the specified system. An abstract machine declares a set of state variables describing the abstract state. The machines operations are used to modify the state variables. First order logic is used to express the invariant of a machine, as well as the preconditions of the operations. The post-conditions are defined as generalized substitutions. The consistency between invariants and operations can be proven. The mistakes are consequently removed, ensuring the correctness of the specification. This is a major advantage of the B method. The B method is entirely supported by automated tools such as the Atelier B.

In [MLL 00] we have presented an approach to formally specify architectural patterns using the B language. A complete description of the B method and the B language can be found in [Abr 96].

In the current approach, ABAS technique is used to describe the high-level structure of a style, such as components/classes and association relationships among them. Then, a first B abstract specification is deduced from the ABAS description and used to check consistency. To do so, an abstract machine is associated to each structural component of the style. Subsequently, the B notation is used to describe details of each component that are left unspecified in ABAS, such as the composition and data types of class states, the behavior of class operations and the global invariants. At this level a number of important decisions concerning some unspecified properties must be elucidated by the developer. The quality attributes are included at this point. The resulting specification is then used to determine the quality attribute values of the architecture.

Figure 2 shows the structure of the B specification associated to the Data Indirection style. The left hand side of the picture shows the *uses* and *includes* links between the different machines. On the right hand side, we present the machine *Data Indirection* which specifies the architectural style. It includes the components *Consumer*, *Producer* and *Protocol Consumer Producer* (the intermediary). Because of lack of space, the complete specification of these components is omitted here. In order to measure the non-functional requirements, the quality attributes are associated to the B machines through the *definitions* clauses. Notice that a definition called *availability* is used to associate the availability attribute to the *Data Indirection* machine. The availability of the whole architecture is calculated from the consumer's perspective. For a given consumer (C_j), the availability of the system takes into account five variables :

- the availability of the consumer itself, avail(C_j)
- the availability of the connector, avail(co-I-C_j), between the intermediary and the consumer
- the availability of the intermediary, avail(I)
- the availability of each producer (P_i) communicating with the intermediary, avail(P_i)
- the availability of the connector between each producer and the intermediary, avail(co-P_i-I).



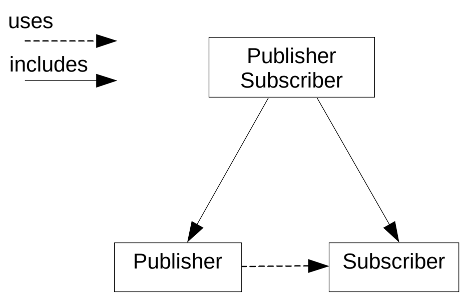
```
MACHINE Data_Indirection
        /* machine specifying the whole architectural style */
INCLUDES Consumer, Producer, Protocol_Consumer_Producer
        /* used machines */
VARIABLES
   message, protocol_consumer_producer, dispatched_messages
        /* state variables of this machine */
INVARIANT
   message ⊆ MESSAGE ∧
   protocol_consumer_producer ∈ consumer ⤔ producer ∧
   dispatched_messages ∈ message → producer ∧
   ran(protocol_consumer_producer) ⊆ ran(dispatched_messages)
        /* the invariant on variables */
DEFINITIONS
  Availability(Cj) ==
        Σ ( avail(Pi) * avail(co-Pi-I) ) * avail(I) * avail(co-I-Cj ) * avail(Cj)
INITIALISATION
   message,protocol_consumer_producer,dispatched_messages := ∅,∅,∅
        /* the initial state of variables */
OPERATIONS
        /* … operations definitions …*/
```

Figure 2. B specification of Data Indirection style

Figure 3 shows the specification of the Publisher/Subscriber architectural style. Notice that, for given subscriber (S_i), the availability of the system takes into account the following variables:

- the availability of the subscriber itself, avail(S_i)
- the availability of the connector, avail(co-Pub-S_j), between the publisher and the subscriber
- the availability of the publisher, avail(Pub).

As for the Data Indirection specification, the availability attribute of Publisher Subscriber is declared as *definition* within the abstract machine.



```
MACHINE Publisher_Subscriber
        /* machine specifying the whole architectural style */
INCLUDES Publisher, Subscriber
        /* used machines */
VARIABLES
   message, protocol_publisher_subscriber
        /* state variables of this machine */
INVARIANT
   message ⊆ MESSAGE ∧
   protocol_publisher_subscriber ∈ publisher ⇸ subscriber
        …
        /* the link invariant between this machine and the refined */
DEFINITIONS
   Availability(Si) == avail(Pub) * avail(co-Pub-Si) * avail(Si)

        …
```

Figure 3. B specification of Publisher Subscriber style

5. Case study: selection of the architecture for CSE.

5.1 Selection of the Publisher/Subscriber with push model for Stock Exchanges Monitoring Systems.

In the previous sections, we have studied different characterizations of architectural models for real-time distributed systems, in particular for stock exchanges monitoring systems. The extended ABAS framework formulated for each candidate architecture has provided useful guidelines for helping in the selection criteria.

- Mediator is not adequate because it favors encapsulation (abstraction) of components (see Figure 1), communicating colleagues that do not know each other by means of an intermediary (Mediator); even if it favors low coupling, it is better adapted for achieving modifiability and reusability, instead of availability.

- Publisher/Subscriber with push model is adequate because it offers a selective broadcasting of the data by the publisher, maintaining at the same time a low level of coupling. The costs of redundancy may be paid, because the structure of the Publisher/Subscriber is not complex. Notice that since all the architectures studied derive from the Data Indirection style, they have in common a high dependence from the intermediary component. Then redundancy is crucial for availability. But if the involved structure of the pattern is simple, complexity will decrease and so will decrease cost.

5.2 Evaluation of the availability attribute

We applied the Publisher/Subscriber style with push model to design the architecture of the CSE. The publisher receives directly the *feeds* from the antenna and broadcasts them to the subscribed brokers via a connector. The brokers are provided with a component subscriber, as shown in figure 4.

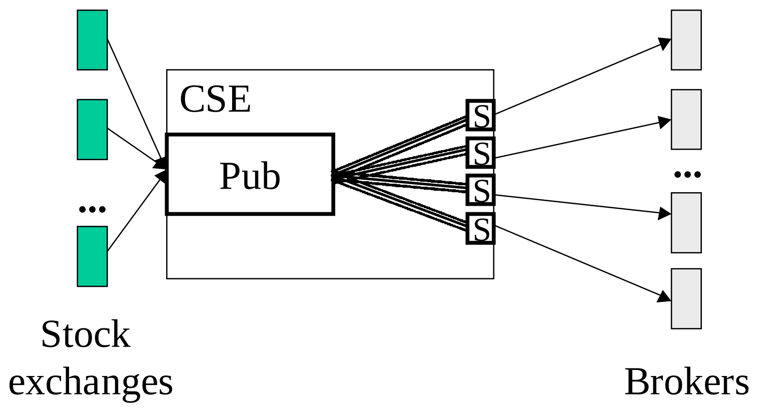


Figure 4. Application of Publisher/Subscriber style

The availability attribute for this architecture and for the ith broker is the following:

$$avail_i = avail(\ Pub\ ) * avail(co\text{-}Pub\text{-}s_i\ ) * avail(\ S_i)$$

where avail(Pub) is the value of the availability attribute associated to the publisher machine, avail(S_i) is the one of the subscriber machine and avail(co-Pub-s_i) the one of the connector between the publisher and the ith broker.

In order to enhance this availability, we choose to use Internet as a connector. Internet can be considered as always available, i.e. its availability is equal to 1. The architecture obtained is shown in Figure 5.

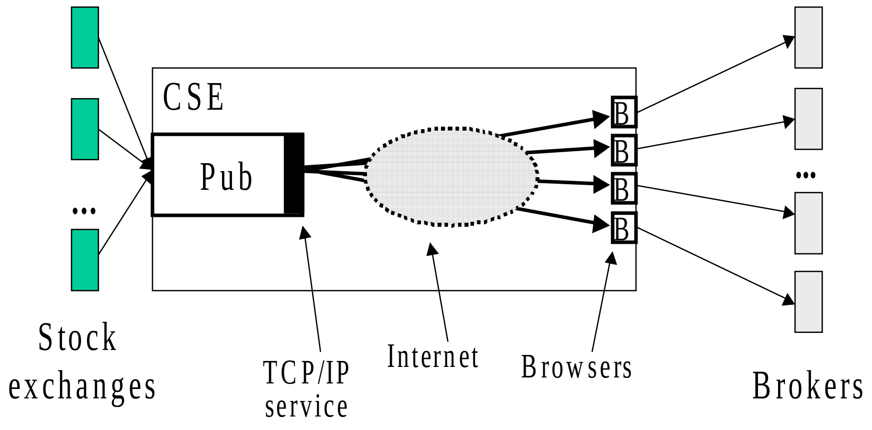


Figure 5. Use of Internet as connector

The availability attribute for this architecture and for the ith broker is now the following:

$$avail_i = avail(\ Pub\ ) * avail(\ B_i)$$

where avail(B_i) is the availability of the browser used by the broker to interact with the publisher.

This availability formula shows that the availability of the publisher is crucial. The introduction of a redundant publisher will double this availability. The architecture obtained is shown in Figure 6.

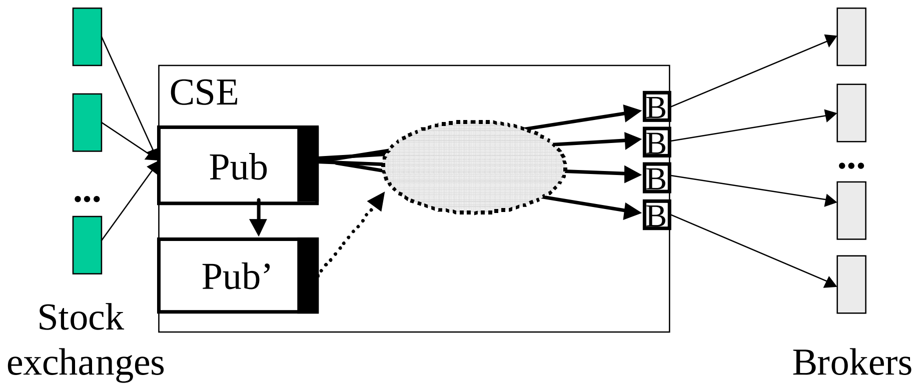


Figure 6. Introduction of a redundant publisher

The availability attribute for this architecture and for the ith broker is now the following:

$$avail_i = 2 * avail(Pub) * avail(B_i)$$

6. CONCLUSION

In this paper we have studied different attribute-based architectural styles (ABAS). The styles have then been formalised using the B language. Each component is specified as an abstract machine in which quality attributes are defined. We have taken the availability attribute as an example. Then, we have applied the proposed technique in order to design the architecture of a Stock Exchanges Monitoring System. The architecture has been developed by stepwise transformations: first we have used the Publisher/Subscriber style with the push model. The formula of the availability attribute showed the importance of the connector's availability. The use of Internet as connector between the publisher and the subscribers had the advantage to offer a very high availability. Then it appears that the availability depend on the availability of the publisher itself. The middleware solution consisting in introducing a redundant publisher was then applied.

The correct selection of a system architecture enhances the subsequent software implementation and the system as a whole. Moreover, the structural characteristics or topology of the chosen styles influences the overall quality goals. However, the applicability of a style, that is to say the selection of the right style for a particular design issue, is yet an open problem. It has been the object of many relevant works [Gam et al 95], [Bus et al 96], trying to describe patterns to be easily retrieved and reused. However, these attempts lack in general of a standard and formal notation, being limited to an informal description and examples of the application of a style. Quality issues are not explicitly considered. Therefore, this descriptions lead to misinterpretations. This makes them an insecure basis for critical software development.

Formal methods are used to specify precisely the structure and the behavior of the entities composing a system and to prove rigorously that these satisfy the desired structural and behavioral properties. Formal methods promise increased reliability of software systems and provide analysis and verification tools. In [MLL 00], we have introduced a formal framework for system development using patterns. This framework integrates the B formal language, describing the transformation from software architecture to system design through successive transformation steps. In this paper we have shown how formal methods can also take into account quality attributes.

REFERENCES

[Abr 96] Abrial J.R. “The B Book - Assigning Programs to Meanings”, Cambridge University Press, 1996. ISBN 0-521-4961-5.

[BCK 98] L. Bass, P. Clements, R. Kazman “Software Architecture in Practice”, Addison Wesley, 1998.

[BK 99] L. Bass, R. Kazman “Architecture-Based Development”, TR CMU/SEI-99-TR-007, ESC-TR-99-007, April 1999.

[Bosh] J. Bosh “Design and Use of Software Architecture”, ACM Press, 2000.

[Bøe et al 99] Bøegh J., DePanfilis S., Kitchenham B., Pasquini A. “A Method for Software Quality Planning, Control and Evaluation”. IEEE Software, 69-77, March/April 1999

[Bus et al 96] F. Buschman et al “Pattern-Oriented Software Architecture. A System of Patterns”, John Wiley & Sons Inc., 1996.

[CLP 00] Chirinos L., Losavio F., Pérez M.A. “Attribute-Based Techniques to Evaluate Architectural Styles for Interactive Systems”, Centro ISYS, Universidad Central de Venezuela, Caracas, May 2000, Draft.

[Dou 99] Douglass B. P. “Real-Time UML” Second Edition, Addison-Wesley, 1999.

[Gam et al 95] E. Gamma, R. Helm, R. Johnson and J.Vlissides “Design Patterns – Element of Reusable Object-Oriented Software”. Addison Wesley, New York 1995.

[ISO 98] ISO/IEC FCD 9126-1.2: “Information Technology - Software Product Quality.Part 1”: Quality Model, 1998.

[KK 99] Klein M., Kazman R., “Attribute-Based Architectural Styles”, CMU/SEI-99-TR-022, ESC-TR-99-022, October 1999.

[Kru 00] P. Krutchen “The Rational Unified Process. An Introduction”, 2nd. Edition, Addison Wesley, Reading, Massachussets, 2000.

[Kaz et al 98] 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.

[LC 99] 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.

[MLL 00] Marcano R., Lévy N., Losavio F. “Spécification et Spécialisation de Patterns en UML et B”. Proceedings LMO’2000 – Langages et Modèles à Objets, Ed. Hermès, Montréal (Ca), janvier 2000.

[SG 96] Shaw M., Garlan D. “Software Architecture – Pperspective of an Emerging Discipline”, Perentice Hall, 1996.

[SR 98] B. Selic, J. Rumbaugh “Using UML for Modelling Complex Real Time Systems”, RSC, OTL, March 1998.