Knowledge Repository Concerning Architectural Styles For building Component-Based Systems

A. Ramdane-Cherif, L. Hazem and N. Levy

PRiSM, Université de Versailles St.-Quentin,
45, Avenue des Etats-Unis,
78035 Versailles Cedex, France
rca@prism.uvsq.fr

Abstract. Component-based systems built from existing software components are being used in wide rang of applications that have high quality attribute requirements. In order to achieve the required levels of reliability, availability reusability, and so forth, it is necessary to provide solutions at the architectural level that are able to guide the structuring of components into an architecture using the architectural styles. The concept of architectural styles (patterns) has had a large impact on component-oriented programming. It has greatly helped specifying and organizing components, and integrating sets of components in a reuse system. In this paper, we propose to design a tool support for architectural styles which take into account some of their problems (properties, constraints, quality attributes, etc.). Mechanisms of modeling the architectural styles in knowledge data base and the automated search of the specific architectural style guided by some quality attributes are presented. This paper describes work in progress which will lead in the future to provide a complete knowledge repository which groups all the metadata concerning the software architecture. This knowledge repository will constitute the baseline of some tools related to the architectural styles. Theses tools will guide the less experienced architect to build easily his applications.

1 Introduction

A critical aspect of any complex software system is its architecture. The "architecture" term conveys several meanings sometimes contradictory. We consider that an architecture deals with the structure of the components of a system, their interrelationships and guidelines governing their design and evolution over time [1][2]. So, a description of an architecture is composed of identifiable components of various distinct types. Components interact in identifiable distinct ways. Connectors mediate interactions among components, they establish the rules that govern components interactions and specify any auxiliary mechanisms required. A configuration defines topologies of components and connectors. Both components

and connectors have interfaces. A component interface is defined by a set of ports, which determines the component's points of interaction with its environment. A connector interface is defined as a set of roles which identifies the participants of an interaction. A binding defines the correspondence between elements of an internal configuration and the external interface of a component or a connector. Bindings identify equivalence between two interface points. Moreover, connector always associates a role with a port, while a binding associates a port with another port, or a role with another role as show in (Figure-1-).

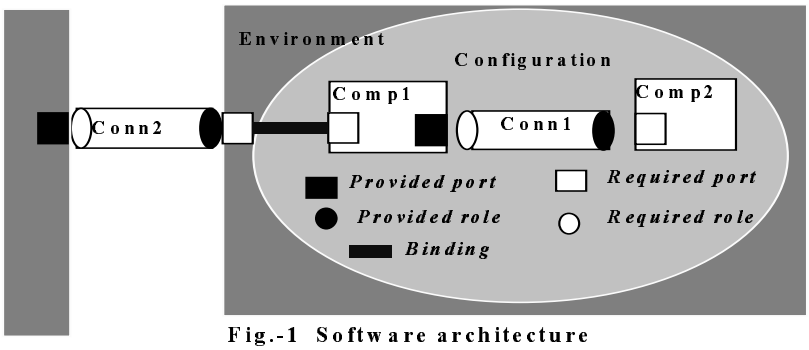

Fig.-1 Software architecture

In order to be able to evaluate the quality attributes of an architecture, In our previous work [3], we have introduced a set of variables representing them within the architecture specification. These variables are defined by functional expressions. The attributes are constrained by predicate expressions. Therefore, it becomes possible to measure the impact in terms of a quality attribute on an architecture by applying some operation presented in the architecture specification. It remains to describe and formalize the modifications strategies allowing the enhancement of one specific quality attribute. Fortunately, it is possible to make quality predictions about a system. These will be based solely on an evaluation of its architecture.

The architectural model of a system provides a high level description that enables compositional design and analysis of components-based systems. The architecture then becomes the basis of systematic development and evolution of software systems. Furthermore, the development of complex software systems is demanding well-established approaches that guarantee the robustness and others qualities of products. This need is becoming more and more relevant as the requirements of customers and the potential of computer telecommunication networks grow. A software architecture-driven development process based on architectural styles (Figure-2-) consists of a requirement analysis phase, a software architecture phase, a design phase and maintenance and modifications phase. During the software architecture phase, one models the system architecture. To do so, a modeling technique must be chosen, then a software architectural style must be selected and instantiated for the concrete problem to be solved. The architecture obtained is then refined either by adding some

details or by decomposing components or connectors (recursively going through modeling, choice of a style, instantiation and refinement). This process should result in an architecture that is defined abstract and reusable. The refinement produces a concrete architecture meeting the environments, the functional and non-functional requirements and all the constraints on dynamics aspect besides the static ones.

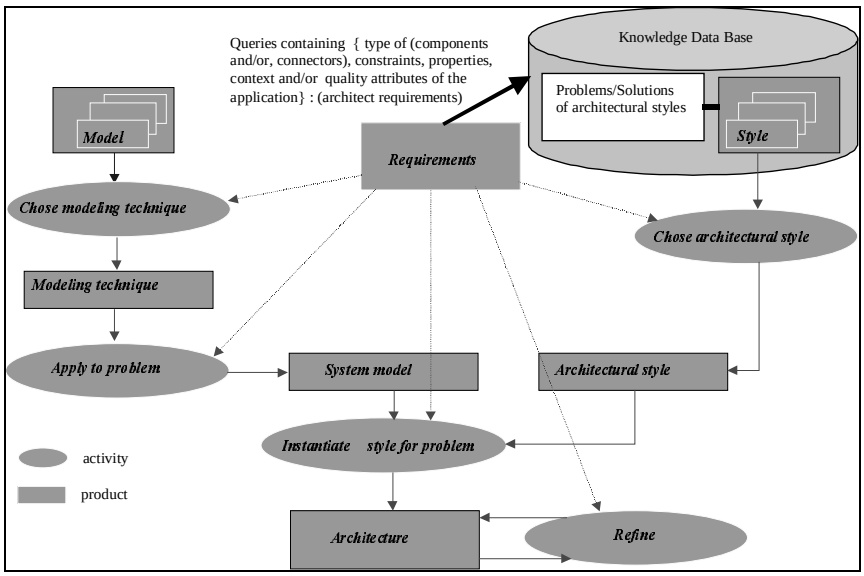

Fig.2. Software architecture phase in development process

Architectural organizing principles for software systems are called architectural styles. In what follows we will consider an architectural style [1] or architectural pattern [5] as a general description of the pattern of data and interaction among the components. An architectural style is seen as a high level view of the architecture, constituted by components, connectors and holding a behavior. Components and connectors have specific functionality and their own quality requirements, which can influence the quality requirements of the software components using style. Architectural patterns may be components of the style [4] and design patterns [6] may be components of the architectural patterns and/or styles. Idioms are considered lower level patterns used in general at implementation stage. We are concerned here mostly with architectural styles. Architectural styles describe solutions to recurrent architectural problems. Their contribution covers the definition, the design, and

documentation. Architectural patterns are meant to improve certain software quality characteristics. For example they are defined to improve the flexibility, understandability, performance, reliability, security ….and other software qualities.

Architectural styles have enjoyed widespread popularity in the past few years, and for good reason: they represent the distilled wisdom of many experienced architects and guide less experienced architects in designing their architectures. However, architectural styles employ qualitative reasoning to motivate when and under what conditions they should be used. An example of the type of description that is a portion of the definition of the pipe and filter style is "Use the pipe and filter style when reuse is desired and performance is not a top priority". The software architect can choose a style based on an understanding of the desired quality goals of the system under construction. Since the architectural styles are also useful in analysis. When analyzing a system, the recognition of the use of pipe and filter, for example, leads to questions about how performance is handled and about the assumptions that the filters make that might impact their reuse. Focusing on particular quality attributes leads to the ability to attach known analytic models for these attributes to the architecture being analyzed. This in turn, leads to the ability to predict the effect of particular architectural decisions and changes to the architecture. Thus, instead of the designer having vague guidance about a particular style having an effect on performance, the designer is given a model, its analysis, and its explicit connection to aspects of the architectural style so that the designer can answer questions such as "what is the effect on performance of moving a particular piece of functionality from one component to another within a pipe and filter based architectural design. Indeed we are beginning to see a proliferation of environments oriented around specific architectural styles. These environments typically provide tools to support particular architectural design paradigms and their associated development methods [7][8][9][10]. Unfortunately, these tools are concentrated on solutions, not paying attention to the problems that the patterns solve. This leads to superficial support by tools.

We believe that methodologies and tools can help developers in automatically detecting and applying the architectural style related to their requirements needs. When developing large system, the developer need to abstract the functional and non-functional requirements of the application at the architectural level. The developer must be particularly careful with the architecture. This paper is organized as follows. In the next section, we will introduce our Objectives to provide a framework at the architectural to help the architect designer. Then our approach and some reflections about its methodology will be presented. After, we will briefly describe the tool used to specify the problem of the application at architectural and to detect the corresponding solution (finding the architecture style which its problem part match the desired application problem). In the fifth section, we describe an application which is highly simplified for presentation purpose. Finally, the paper concludes with a discussion of future directions for this work.

2 Objectives

Good architectural design has always been a major factor in determining the success of a software system. A critical challenge faced by the developer of a software architecture is to understand whether the components of an architecture correctly integrate and satisfy all the architectural constraints and quality attributes. However, it is important to provide a framework at the architectural level that will provide substantial help in (Figure-3-):

- proposing guidelines to automate the detection of architectural patterns based on user refinement problem and constraints (software rules, quality attributes,….).
- storage of different kind of architectural styles under the format: problem and solution.
- building large applications using architectural patterns (development guided by the abstract architectural design using the architectural styles toward the implementation through some problem refinements.
- detecting, early in development, the architectural style which will be the more adapted to the desired application (matching the problem of the style).
- using constraints (software rules, quality attributes) to detect into data base of patterns groups of entities similar to a modeled application and to comply with the specification given by the pattern architecture model.
- choosing the different components and connectors that satisfy the desired structural and behavioral properties (depending on some quality attributes)
- combining decomposition and refinement which offers a very powerful tool to build a general abstract specification which can be gradually made more precise and concrete (towards the concrete solution by refinements of the initial application problem).
- proposing an automatic tool guided by the user requirements which decide when and under what conditions the architectural styles should be used.

3 Support for architectural design

Pattern occur in all phases of design. In this paper we discuss two kinds of patterns: architectural styles and design patterns. A third kind of pattern, code patterns, will not be treated in depth because it is not by and large architectural in nature. What all patterns have in common is that they are pre-designed “chunks” that can be tailored to fit a given situation and about which certain characteristics are known. Each pattern represents a package of design decisions that has already been made and can be reused, as a set. What is most different about them is their scale and hence the time of development when each is applied. Architectural patterns (Architectural style) tend to

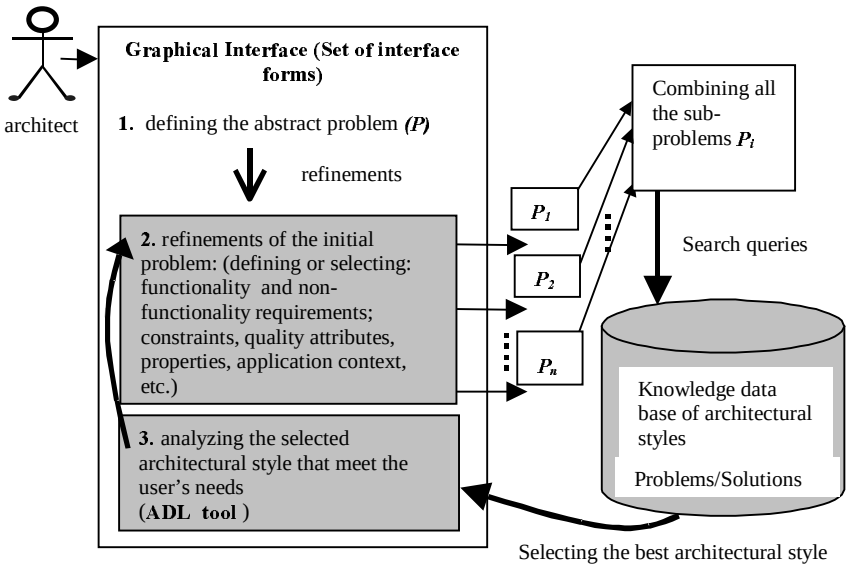

Fig.3. The process of defining the architectural problem and detecting the corresponding solution (architectural style)

be applied by an architect, design patterns usually occur within the confines of an architectural component, and code patterns live in the implementor's tool kit.

Thus, a style is not an architecture. Rather, a style defines a class of architectures; equivalently, it is an abstraction for a set of architectures that meet it. Styles are usually ambiguous (intentionally so) about the number of components involved. For example, a pipe and filter stream may have two filters connected by a pipe or 20 filters connected by 19 pipes. Style may be ambiguous about the mechanism(s) by which the components interact, although some styles (main-program-and-subroutine, for example) bind this explicitly. Styles are always ambiguous about the function of the system: one of the component may be a database, for example, but the kind of data may vary. Style are categorized into related groups. For example, an event system is a substyle of independent components. Event Systems themselves have two substyles: implicit and explicit invocation. Style catalogs also tell us the circumstances in which it is appropriate to apply a style.

The architecture expressed at a high-level by architectural styles, must satisfy some quality attributes. A subset of the architectural decisions that a designer will make is based on properties. The architectural properties in conjunction with quality attributes lead to characterize the actual behaviors model of the system which is unknowable without constructing the system. Our approach follows the three following steps:

1. The purpose of the first step is to describe the architectural design problem being addressed or in other words the goals of the architecture. The problem description consist in:
    - a description of the design: the intend to solve, including the quality attribute of interest, the context of use, constraints, and relevant attribute-specific requirements.
    - a description of the architectural style in terms of component, connections, properties of the components and connections, and patterns of data and control interactions.
    - a description of how the quality attribute models are formally related to the elements of the architectural style and the conclusions about the architectural behavior that are drawn via the models.
2. The second step consist to:
    - associate quality attributes to the architectural design
    - define the requirements needed for any style (constraints): each style require a category assignment (attribute values).
    - matching the architect problem definition to the problem underlying the architectural style (functionality, constraints, properties, quality attributes, etc.)
3. The Third step : this step describe the solution in the format of architectural style which is determined by the following:
    - a set component types (e.g., data repository, a process, a procedure) that perform some function at runtime.
    - a topological layout of these components indicating their runtime interrelationships
    - a set of semantic constraints (for example, a data repository is not allowed to change the values stored in it)
    - a set of connectors (e.g., subroutine call, remote procedure call, data streams, sockets) that mediate communication, coordination, or cooperation among components.
    - primitive component and composite component

4 The proposed tool

To construct our tool, we can distinguish four steps. The first is related to the knowledge acquisition which is done during user requirements analysis. This step concerns the collection of all the knowledge necessary of the conceptual modeling of the data base. The second is related to the data abstraction and structuring. This step concerns the organization of the knowledge acquired during the design phase and the definition of the main entities and relationships that best capture the views of the users. This corresponds to the effective conceptual modeling phase. The third step is related to the establishment of the data base queries which represent the main activity

of the data base. The last step is related to the graphical interfaces constitution which integrate design alternatives and high level interaction with human architect or designer. In our data base design, we distinguish between four abstraction levels: external, conceptual, logical and physical design. Based on these levels, we used the relational normalization, scheme mapping between entity-relationship model and relational model.

Following these steps, we obtained our meta model software architecture which makes it possible to store the catalog of available independent entities (software architecture components) but also to store the architectural styles.

We have carried out the following tables:

Architecture (Number-Architecture, Name, functionality)
Configuration (Number-Configuration, Name, Functionality)
Component (Number-Component, Name, Type, Functionality, constraints)
Port (Number-Port, Nom, Type, Polarity)
Connector (Number-Connector, Nom, Type, Functionality, Constraints)
Binding (Number-Binding, Name)
Role (Number-Role, Name, Type, Polarity)
Quality(Number-Quality, performance, security, maintainability, portability, reusability, availability, reliability, usability, modifiability, testability, integrability)
Specialized-Component(Number- Specialized -Component, Name, Type, Functionality, Constraints)
Specialized-Connector(Number- Specialized -Connector, Name, Type, Functionality, Constraints)

Specialized components (or specialized connectors) can be constituted starting from several simple (primitives) components (or connectors). These components allow directly, for example, to obtain the characteristics of a combination of components and connectors. An architecture can contain one or more configurations and a configuration can belong to one or more architectures. Configurations can contain several components and connectors, a component or (a connector can belong to one or more configurations. Configurations can have one or more binding to connect them to other configurations within an architecture in a completely hierarchical way. A specialized component can be composed of several simple components and a simple component can belong to one or more specialized components. A specialized connector can be composed of several simple connectors and a simple connector can belong to one or more specialized connectors. Each component (simple or specialized) can have one or more ports and each port can belong to several components. Each connector (simple or specialized) can have one or more roles and each role can belong to several connectors. Each simple or specialized component (or connector) can have several quality attributes and several constraints on the context of its use, its environment or more on its functionality.

While passing from the realization of the dependence functional diagram and the entity-relationship model diagram, we obtained the relational model which we have then normalized. Thereafter, this model makes it possible to store elementary basic entities of an architecture like: components, connectors, ports, roles, configurations etc. One can thus carry out the SQL joint queries between these various entities to

constitute architectures or precisely architecture styles. There will be thus an evolutionary base knowledge of architectural styles which contains:

- diagrams of these architectural styles,
- components of these architectural styles
- constraints, context, properties, quality attributes, etc. of these architectural styles through their components.

Each entity constituting a style of architecture is backed up in a table bearing the name of this entity and containing all its properties. But the joint relation between the various tables of several entities through a simple SQL query makes possible and very easy the constitution of the architectural style. In this data base, one can store implicitly the couples (problems-solutions) concerning the architectural styles.

The quality attributes are regarded as percentages compared to an ideal value (100%) (example: if the measurement of the security quality attribute value is 30% for a component this means that the security is satisfied to 30% of the ideal security). The constraints, the context and the properties are formalized in the form of key words which refer to specific cases that one finds in the study of software architectures of several systems.

We elaborated a set of SQL queries which make possible a search in the data base to find such a component, such an architecture, etc. Other more elaborated queries make possible to find the architectural styles based on the architect requirements. These needs will be collected in the form of quality attributes, constraints, context, etc. All these needs which are nonfunctional requirements will be combined with the functional requirements of the system in the refinement process towards the solution (the specific architectural style). A graphical user interface will facilitate the task of the architect designer, little experienced in the field of architectures, to find easily the adapted style to its application. In addition, it is possible to the experimental designers to enrich the base by other architectural styles to put their expertise at the profit of the less experienced architects.

5. Application

For simplicity, each conj (connection between two components via a connector) is represented like in (Figure-4-).

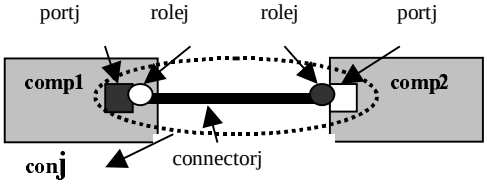

Fig.4. Example of a connection j (conj)

The user designer has a problem consisting to implement a client sever system. He will describe his problem with all constraints, quality attributes, etc. concerning this system via the graphical interfaces. Then, our tool will find from these descriptions the architectural styles corresponding to his needs. This is done by matching these designer description problems to the problems of all architectural styles existing in the data base. For this application, Several architectural styles are already analyzed and stored in the knowledge data base under the format (problems/solutions). These styles are for example: Client-Server style, Mediator style, Client-Dispatcher-Server style and Broker style.

The designer do not specify any detail on the type of communication between client and server, the tool will detect the style of the most abstract level. This style is the Client-Server style whose communication between clients and servers around a service appears, not specifying any detail on this communication (Figure-5-).

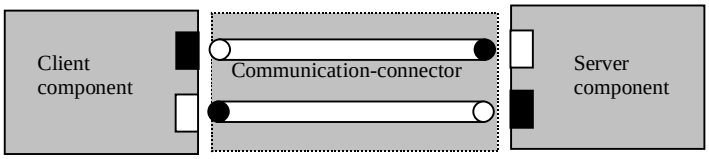

Fig.5. Client-Server Style

The designer will want introduce in his application a mediator component, or he will want the security quality attribute value be more significant than that of client-server style. Therefore, our tool will select the Mediator style whose communication is ensured by a component playing the role of intermediary between several components which cannot communicate directly between them (Figure-6-).

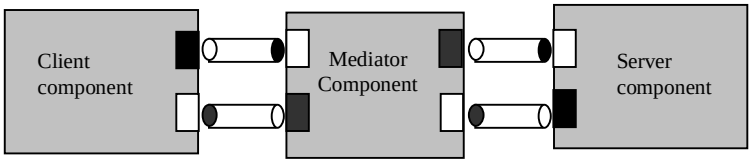

Fig.6. Mediator Style

The designer will prefer that the client must express a request for a connection or he will want the security quality attribute value be more significant than that of mediator style. Therefore, the tool detect the Client-Dispatcher-Server style. In this case the communication between clients and servers is established by the dispatcher component. The relation between the dispatcher and the client expresses the request for a connection whereas the relation between the dispatcher and the server expresses that the dispatcher seeks to establish a link of communication with the server. If this

request for connection would be accepted by the server, the dispatcher communicates it to the client and establishes direct connection between the client and the server. In the contrary case it return an error message to the client (Figure-7-).

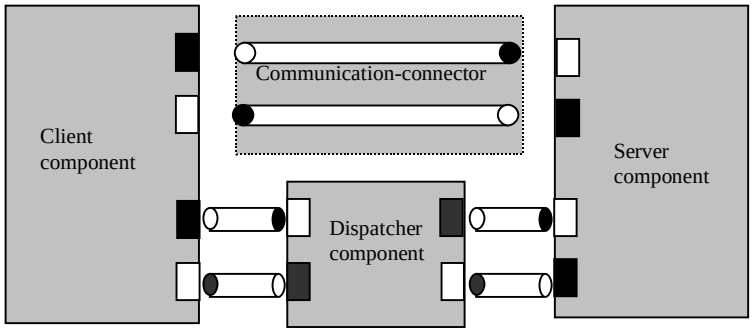

Fig.7. Client-Dispatcher-Server Style

The designer will want to achieve a complete decoupling between the clients and the servers and he will want the security quality attribute value be the most significant. Then the Broker style will be selected. In this case, the servers will register at broker component and propose their services to the clients. The clients reach the functionalities of the servers by sending the commands to the broker component. The tasks of the broker consist to locate the suitable server for each request of the clients, to send the request and to transmit in return to the clients the results or the error messages (Figure-8-).

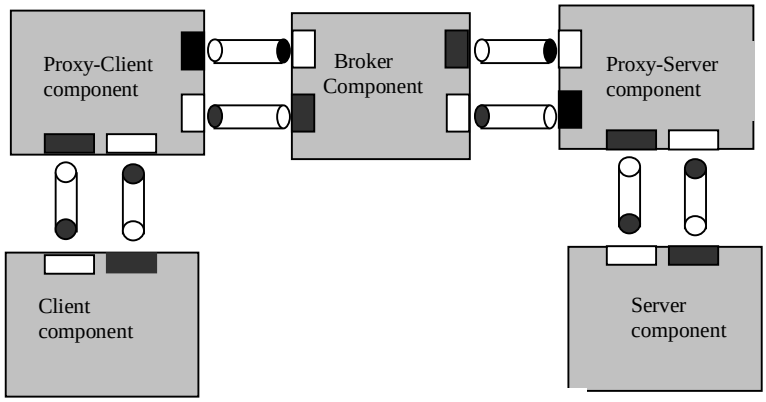

Fig.8. Broker Style

The mechanisms of modeling the architectural styles in knowledge data base and the automated search of the specific architectural style guided by some quality attributes, constraints, properties, etc. can be presented as a refinement process. This process starts from the abstract definition of the problem by the designer. This problem can after be refined gradually by adding some designer's details toward the specific solution. To do so, at each detail level of the designer specification, our tool proposes the corresponding architectural style. If another detail is added to designer specification another architectural style will be proposed until the best architectural style that meet all the designer requirements is detected.

5. Conclusion

The design patterns can be applied at many different levels of abstraction in the software development life-cycle, and can focus on reuse within architectural design as well as detailed design and implementation. In fact, a system of patterns for software development should include patterns covering various ranges of scale, beginning with patterns for defining the basic architectural structure of an application (architectural styles) and ending with patterns describing how to implement a particular design mechanism in a concrete programming language. In this paper, we are focused especially on architectural styles which have a large impact on component-based software development. It should be possible to describe a system as a composition of impendent components and connections and then to combine independent reuse elements into large system based on architectural styles. In this paper a knowledge data base tool which integrate architectural styles (their problems and solutions) and use high level interaction with architect designer is presented. We have given our ideas about the storage and the automated search of architectural styles guided by some quality attributes at the architectural level. We plan to further investigate the possibilities of describing a complete system for developing architectural design that exploit architectural styles to guide software architects in producing specific systems by using a complete knowledge repository. This repository groups all the metadata concerning the software architecture. Another direction of research consists in combining this approach with another approach based on development guided by the lower conception design using the design patterns toward the implementation through problem refinements.

References

1. M. Shaw, D. Garlan, Software Architecture, Perspectives on Emerging Discipline, Prentice-Hall, Inc. , Upper Saddle River, New Jersey, 1996.

2. D. E. Perry, A. L. Wolf, Foundations for the Study of Software Architecture, Software Engineering Notes, 17(4):40, Oct. 1992.
3. A. Ramdane-Cherif, N. Levy and Francisca Losavio, Adaptability of the Software Architecture Guided by the Improvement of Some Desired Software Quality Attributes. In SERP'02: International Conference on Software Engineering Research and Practice. Monte Carlo Resort, Las Vegas, Nevada, USA, June 24-27, 2002.
4. S. J. Mellor, R. Johnson, Why Explore Object Methods, Patterns, and Architectures, IEEE Software Jan/Feb 1997.
5. F. Buschman et al, " Pattern-Oriented Architecture. A System of Patters", John Wiley & Sons Inc. 1996.
6. E. Gamma, R. Helm, R. Johnson and J. Vlissides " Design Patterns -Element of Reusable Object-Oriented Software" Addison Wesley, New York 1995.
7. W. Mak. Victor, Connection: An inter-component communication paradigm for configurable distributed systems. In proceedings of the international workshop on configurable distributed systems. London, UK, March 92.
8. P. Binns and S. Vestal. Formal real-time architecture specification and analysis. In Tenth IEEE workshop on real-time operating systems and software. New York. May 1993.
9. A. H. Eden , Y. Hirshfeld (2001). "Principles in Formal Specification of Object- Oriented Design and Architecture." CASCON 2001, November 5-8, 2001, Toronto, Canada.
10. A. H. Eden, "LePUS: A Visual Formalism for Object-Oriented Architectures". The 6th World Conference on Integrated Design and Process Technology, Pasadena, California, June 26-30, 2002.