

Reusing Families Design

Virginia C. de Paula
Computer Science Department
Long Island University - Brooklyn Campus
1 University Plaza
Brooklyn, NY, 11201, USA
Telephone No.: 1 718 488 1269
Fax No.: 1 718 780 4145
virginia.depaula@liu.edu

ABSTRACT

Designing the detailed architecture of a style or of an architectural family is a difficult task. However, it offers great benefits for the product architectures detailed design based on this style or family. The present work proposes a meta-model for the development of applications with layered architectural style and distributed with the middleware CORBA (Common Object Request Broker Architecture). The meta-model offers all the intrinsic information to this family of systems, from the architectural to the detailed level. It has been specified in UML (Unified Modeling Language) and the formal language Z was used to specify some constraints and allow the integration of the meta-model to ZCL, a formal framework to specify and reason about dynamic software architectures. This initiative provides the guidelines that can be followed to detail other styles and families.

Keywords

Software Architectures, Layered Architectural Style, UML, Software Reuse, Software documentation

1. Introduction

The reuse of components provides several benefits for software engineering, such as, gains in productivity; in quality; and in development schedule; besides decrease in development costs [21]. However, the reuse of components embraces not only the code and punctual solutions reuse, but also, solution models, design and tests [1], considering not only the component but the whole engineering of it.

A reusable component should be accompanied by a documentation that presents, besides its functionality, all the artifacts elaborated during its engineering. Offering a component for reuse can indicate much more than only sharing a solution, it usually indicates sharing an experience in the resolution of a certain problem and the whole intrinsic knowledge to this problem or solution.

The availability of those artifacts offers the components better chances to be reused, because it supplies the designer with warranty that the problem was appropriately solved. It also allows the designer to adapt or to extend them so that they can be better adjusted to the new context.

Several patterns that depict solutions for specific problems during the system design or implementation have been reported, in order to make ready and reusable solutions more popular [2, 3]. During the design, patterns can have high or low level of generality. We can call those levels respectively architectural pattern (or style) and design pattern.

The availability of patterns depicts the reuse of the acquired experience for resolution of a problem at a specific generality level. The use of patterns in system development requires methodology and perspicacity. It is not an easy task to deal with the development starting from a style to reach its detailed design, considering the product architecture. The gap between these design tasks is extensive, and it is difficult to guarantee consistence among them (Figure 1a). This task will be equally arduous every time that a different product has to be developed.

Egyed et. al [4] proposes a solution for this problem that consists of detailing the architecture of a style (Figure 1b). This task is also difficult. However, the artifacts of this detailed architecture will turn the detailed design of the product into a less difficult task. Therefore, the detailed design of a style can be reused multiple times during the development of different products, guaranteeing more easiness in the development of these products and better consistence between the style and the detailed architecture of a product based on it.

The layered architectural style organizes software in hierarchical layers, in which each layer is developed over a more general one [1]. One layer can be defined as a set of (sub)systems of equal generalization. Higher layers are application specific and lower layers are of general purpose, i.e. can be used by different applications.

Interactions are usually only between adjacent layers. Each layer provides services to the immediate above layer and works as a client to the immediate below layer. Each layer is composed by systems that can also be specified using the layered architectural style or any other existing style. This decision depends on each layer's complexity and global characteristics. The layered architectural style enriches the systems with several desirable properties [14]:

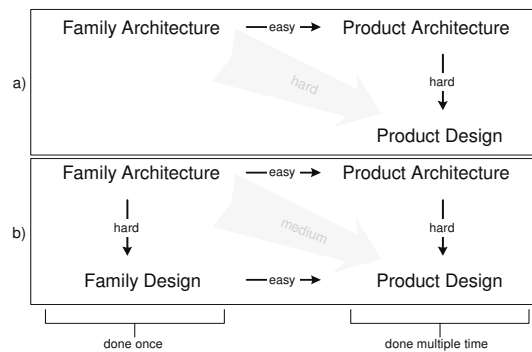


Figure 1 – Two Refinement Approaches [4].

- It supports *design* in crescent levels of abstraction, allowing implementers organize a complex problem as a sequence of incremental steps;
- It supports changes, because most of the layers interact only with two adjacent layers. Therefore, changes in a specific layer will only affect other two;
- It supports reuse. Different implementation of a layer can be used with the only restriction of offering the same interface to the adjacent layers;
- Security can be offered as services by specific layers.

The main goal of the current work is to present a meta-model to specify the detailed design of applications with layered architecture, distributed with the middleware CORBA. In Section 2, we present the architecture conceptual view defined by Hofmeister et. al [5] and used as basis of our work. In Section 3, our proposal - the meta-model to specify layered architecture – is presented. In Section 4, we describe some related work. The conclusions and future work that can be accomplished from our work are presented in Section 5.

2. Architectural views according to Hofmeister et. al

As defined by IEEE Draft Standard 1471, a view addresses one or more interests of the system stakeholders (developer, user, customer, etc) [6]. In [28], a view is defined as “a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders”. Therefore, views help to manage the complexity by disconnecting different interests in different views.

Having as starting point the necessary properties to create the architecture of a system, Hofmeister et. al [5] created four meta-models (called *Siemens four views* by Clements et al in [22]), or architectural views, depicting some characteristics. Each one of them is closed coupled and addresses different interests of engineering. They are:

- Conceptual view – the functionality of the system is mapped to architecture elements called conceptual components, with coordination and data exchange handled by elements called connectors. We can then say that it describes the system in terms of its architectural elements (configuration, components, ports, connectors, roles and protocol);
- Module view – the components and connectors of the conceptual view are mapped for subsystems and modules. It is addressed how the conceptual solution can be realized with today’s software platforms and technologies;
- Execution view – it describes how the modules are mapped for the elements provided by the runtime platform, and how those are mapped for the hardware architecture;
- Code view – it maps and organizes the elements of the Module and Execution views for source code components (object code, libraries and binary).

The four views together work as a guideline to architecture design, because they describe entirely a specific architecture at a certain abstraction level. They are generic models which can be instantiated to specify a product’s architecture or can be refined to specify other models which can be equally instantiated or refined.

The Conceptual View describes the architectural elements of a system (Figure 2). This view prescribes characteristics broadly wanted in architectures such as [14]: composition of components and connectors; abstraction; and reusability of components, connectors and architectural patterns.

In [5], UML (Unified Modeling Language) [7] is used to specify these models. Although UML is not considered an ADL (Architectural Description Language) [8], because it does not have all the desirable characteristics of an ADL, it possesses characteristics which benefit the development of software architectures [9, 23]. For instance: it is easy to use and to learn; it intends to give support to all phases of the design, reducing the gap among the several phases of development; it is flexible and popular.

The great potential of UML for description of architectures has originated researches with focus in extending UML and/or integrating UML to existing ADLs [10, 23], as well as endowing UML with more features to describe architectures and architectural patterns [11]. Due to the great UML’s flexibility and its capability to describe any model in any abstraction level it is possible to describe meta-models (models that define models) in UML.

The meta-model currently proposed has been defined having Hofmeister et. al's conceptual view [5] presented above as starting point. However, Hofmeister et al's model is not appropriate to specify layered architectures, because the components are not organized in layers and the communication between the layers cannot be specified.

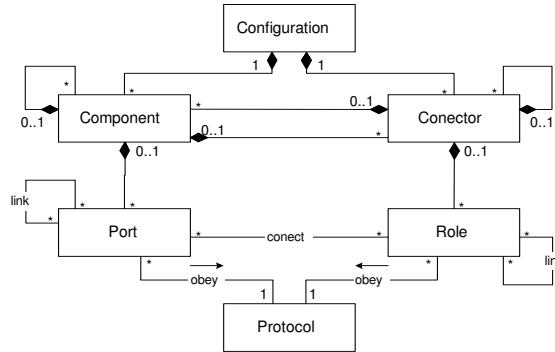


Figure 2 – Conceptual View of Architecture [5]

Therefore, our meta-model can also be seen as a major specialization of Hofmeister's model to specify layered architectures. Our meta-model's static structure was specified through class diagram and its dynamic structure through sequence and states diagrams. We have also used the formal specification language Z [12] to specify constraints of the style which could not be specified through the diagrams. In the following section, we present more details of the proposed meta-model.

3. A Meta-model to specify layered architectures, distributed with middleware CORBA

The meta-model presented here facilitates the design of applications structured in layers and distributed with middleware CORBA. The proposed meta-model groups together two different set of features: the set of the intrinsic layer style characteristics [2, 13, 14], and the one of the intrinsic characteristics of the applications relying on CORBA [15, 16] as communication platform.

One important characteristic of the layer architectural style is to organize the components of a specific software architecture in agreement with their level of generality. This style can be seen as a multi-level client-server, because each layer is either a server providing service to another layer or a client consuming service from another layer. Although usually the interactions happen only among adjacent layers, in the direction Top-Down, Bottom-Up or both directions, there are six interaction types [1]: among adjacent layers in the directions Top-Down, Bottom-Up or both; and among non-adjacent layers in the same direction. The most usual interaction type of the layered style is the one that occurs between adjacent layers in the Top-Down direction, in which a top layer requires service from a bottom layer.

The applications distributed with middleware CORBA represent a family of products which basically imposes the dynamic client-server model and that the components servers register their services in ORB (Object Request Broker). They also require that the components offer services with one of the three synchronization types: synchronous, deferred synchronous and asynchronous.

Each one of these sets of characteristics can be specified by different meta-models [17, 24] which help in the architectural design with layer style or with CORBA issues. However, they are not able to help in the design of applications with both characteristics, because the structuring imposed by the layer style is not foreseen by the applications that use CORBA and vice-versa. Our goal is to provide a meta-model to specify systems with both characteristics.

This is a typical case where implementation decisions, like the use of CORBA, affect the global architecture of the system [18]. In other words, if this decision is not taken in consideration in initial phases of the architecture design, probably, the implemented model will not be consistent with the designed model. Therefore, problems such as the non satisfaction of the functional and non-functional requirements of the system and difficulties in maintaining the architecture can happen.

For the elaboration of a meta-model that satisfies both sets of characteristics, we began by depicting the characteristics of the layer style and step by step we include the communication characteristics imposed by CORBA. The static structure of this meta-model is presented in Figure 3.

The elements of the meta-model have strong relationship with the elements defined in the Conceptual View presented in Figure 2. Besides the elements, the attributes and basic operations were supplied and little summary of each of them is presented by Notes in the diagram.

Some global characteristics of the model proposed by Hofmeister et. al [5] were modified in order to allow our meta-model to support other characteristics not specified by their original conceptual view. For instance, heterogeneity and more flexibility to link ports of composed components.

With regard to heterogeneity, an aggregation was added between the classes *Layered Configuration* and *Layers* and another one between *Layered Configuration* and *Connectors* (in gray in Figure 3). Besides that modification, we needed to specify some constraints in order to guarantee that the aggregation is indeed respected. We also needed to specify how the aggregations between components and connectors happen, the connection between ports and roles and the link among roles.

The specification of these constraints was accomplished using the Z formal language [12]. Z was chosen in order to allow our meta-model to be prepared to extend ZCL [25, 27]. ZCL is a formal framework, written in Z, to describe and reason about dynamic distributed software architectures. ZCL also allows the formal specification of architectural styles [26]. It focuses on the operations necessary for the construction of dynamic software architectures. The framework provides an ADL, also called ZCL. With the extension proposed in the current paper, the ADL ZCL is able, for example, to specify a component-based application organized in layers being executed in a distributed platform based on CORBA. Some issues concerning the integration between ZCL and LuaSpace, a distributed platform based on CORBA, were discussed in [20]. This is an additional contribution of our work. Previous work describes the integration between ZCL and UML [10].

In the proposed meta-model, any interaction among components will only happen through connectors, which are protocols of layer interaction. This gives better flexibility to the interactions: a composed component can interact with its components in a more elaborated way that, for instance, message passage; a composed component will not have to access directly the ports of the internal components with which it interacts.

In Figure 4, we present a sample of a Z schema we have used to specify constraints to Configurations. The first part of a schema expresses declarations and the second one expresses the predicates or constraints upon the objects introduced in the declaration part. The Layered Configuration can be described as: a set of Layers and Connectors; a set of relations Layer-Layer; a set of functions, called level, designating the relation between each layer and a natural number (this number represents a level) and; interaction type (TopDown, BottomUp or Both between adjacency layers, or same directions between any layers). One of the constraints upon objects is: if *interaction_type* equal TopDown-Adjacency then two layers will be related (they must belong to the *Lay-Lay* set) whether the level of the first layer is equal to the level of second layer minus one (expressed through the sixth predicate in schema).

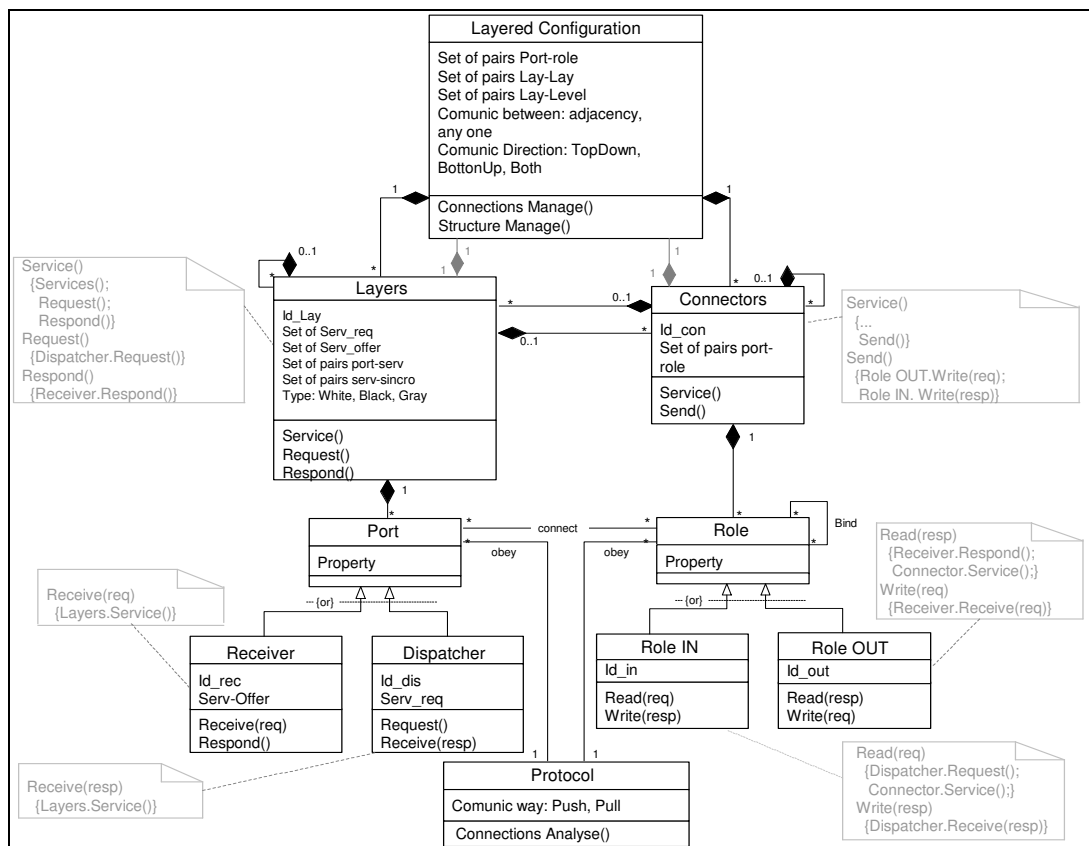


Figure 3 – Static Structure of the Meta-model for Layered Architectural Style

The dynamic structure of the meta-model was specified using sequence and state diagrams, which can also be integrated to ZCL, because ZCL keeps a state machine to control the reconfigurable (dynamic) system at run-time, assuring the system will be in a valid state. One sequence diagram and one state diagram are presented in Figure 5 to illustrate the dynamic structure. These diagrams just represent one possibility of interaction among components of a software architecture: the synchronous interaction. In this interaction, the component waits for answers to its requisition before continuing its internal work. For space reasons, we do not present here the deferred synchronous and asynchronous interactions, but the reader can find them at [24].

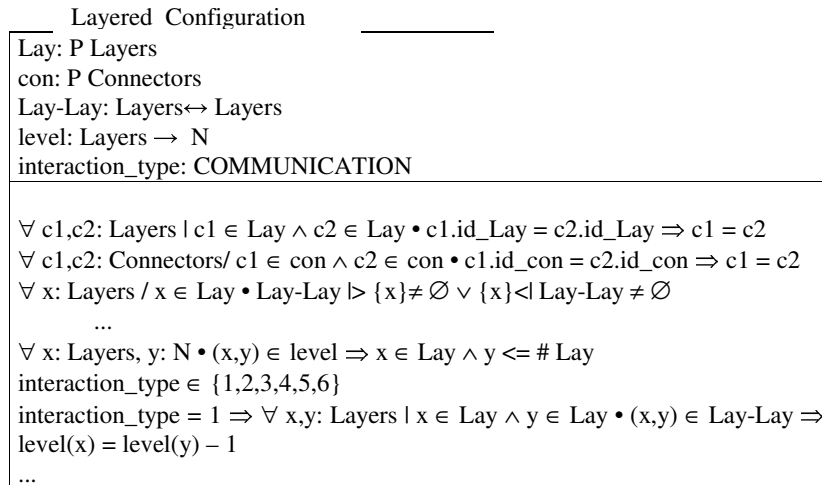


Figure 4: Constraints of Layered Style

In the sequence diagram (Figure 5) a component requests a service through one of its ports. The connector's role linked to that port is responsible for reading the requisition and other role of the same connector is responsible for writing in the receiver port with which it is linked. When the component server finishes the service execution it will send an answer and the connection occur in a similar way.

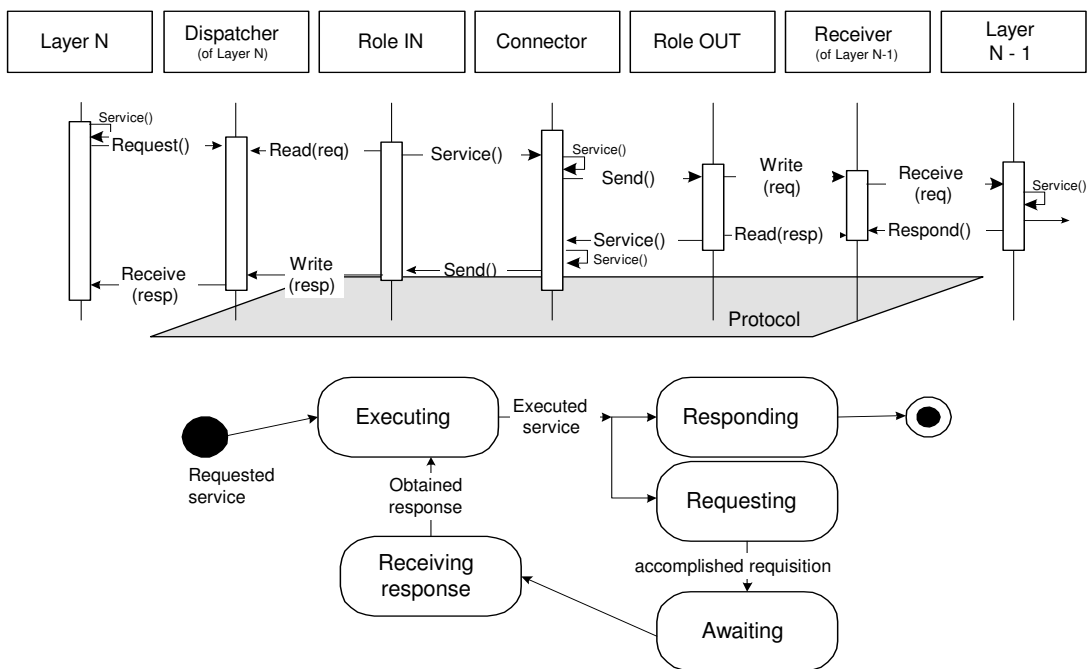


Figure 5 – Dynamic Structure of the Meta-model for Synchronous Interactions

Our meta-model is composed by a static and a dynamic structure, a specification of constraints, and rules of how and when to use it. These artifacts will guide an architect when he/she wants to project a layered architecture and use CORBA as middleware. It will also help in the design of other generic architectures. Therefore, we are providing means of reusing the design of a specific family of architectures.

We have used a case study to validate our meta-model. The case study chosen was the ATM Machine application, which has the following functionalities: withdraw cash, get balance, transfer, pay bills, request checks, etc. This application is typically distributed: the components can be executed in different hardware and software platforms and they have to interact to provide the requested functionality.

It is worth saying that some functionalities in this application clearly present some common activities such as: clients' password verification and analysis; receipt emission; fees charges discounts, etc. Therefore, each of those activities can be associated to several components in the application, allowing the activities to be organized according with their generalization level and more easily reused. We have used the meta-model proposed in the current work to design the

application. We used the CORBA middleware to deal with distribution and the layered architectural style to hierarchically organize the components. All the instances of the meta-model classes and the object diagram were created for the example, but are not presented here due to space reasons. In Figure 6, we illustrate the layered architecture of the case study.

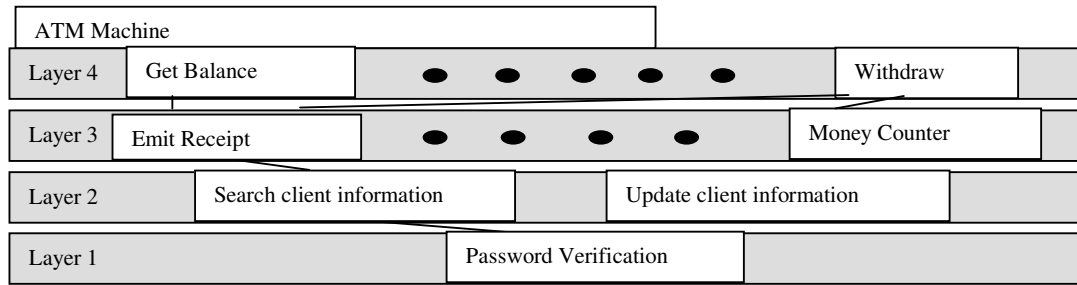


Figure 6 – ATM Machine Layered Software Architecture

4. Related work

A work related to ours is the one of Rosenblum et al. [19], which demonstrates the concern in guiding the designers in the definition of consistent architectures with an infrastructure of pre-selected middleware. They use C2 as architectural description language. We propose the use of UML which is a more flexible and popular language.

The work of Buschmann et al. [3] and Gamma et al. [2] classify architectural and design patterns so that they are popular and easily consulted. Our work does not intend to be a catalog, but it can be considered the specification of a combination of patterns. The artifacts of the meta-model also offer subsidies to guarantee more precisely the consistence between family architecture and product design.

5. Conclusions and Future work

This work presents a meta-model to design distributed layered applications. Our meta-model gives support to the stage of architecture design of a distributed and heterogeneous system. To achieve this goal we offer all the necessary information in a popular and easy language, UML. The artifacts built during the elaboration of our meta-model offer subsidies for: UML to describe styles; to help in the specification of other styles or architectural families; to improve the communication between architects and designers and; to improve the reuse, understanding and maintenance of the developed systems or subsystems. Besides proposing a meta-model to specify layered architecture that use CORBA as middleware, our meta-model extends the ZCL framework, allowing it to be integrated with a component-based development environment.

Our work is part of a major project to specify dynamic and distributed software architecture using formal methods. One of the goals of our project is to map a formal specification into an execution platform. To achieve this, we have been working on integrating ZCL to UML [10], and on integrating ZCL to existing distributed platforms [20]. Therefore, although we are using UML, we specified the constraints of our meta-model in Z, to integrate the meta-model to ZCL, and we are not using the UML Profile for CORBA Specification [29].

Promising works that can be accomplished in this same study line are: to describe other styles as the one defined here; to define a methodology for their use in the engineering and in the reengineering; to apply them and to test them in several contexts; to define other styles, reference architectures and architectures of product line and; to develop a support tool to the use of these meta-models and styles. These are indispensable tasks to obtain a library of styles and architectures entirely modeled and ready for us to be instanced.

6. Acknowledgments

I would like to thank Lyrene Fernandes Silva, a former MSc student from UFRN (Brazil), whose work I supervised, and now a PhD student from PUC-Rio (Brazil). It was really helpful having her to discuss and to clarify ideas.

I also have a lot to thank to Professor Thais Vasconcelos Batista, from UFRN (Brazil). My research would not be so pleasant if she was not always there to discuss and to contribute with her great ideas.

7. References

- [1] JACOBSON, I., GRISS, M., and JONSSON P.: Software Reuse, Addison Wesley, 1997.
- [2] BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., and STAL, M.: A system of patterns – Pattern-Oriented Software Architecture, Wiley, 1996.
- [3] GAMMA, E., HELM, R., JOHNSON, R., and VLISSIDES, J.: Design Patterns – Elements of Reusable Object-Oriented Software, Addison Wesley, 1995.
- [4] EGYED, A., MEHTA, N., and MEDVIDOVIC, N.: Software Connectors and Refinement in Family Architectures, 3° International Workshop – Software Architectures for Product Families, Las Palmas de Gran Canaria, Spain, March 15-17, 2000.
- [5] HOFMEISTER, C., NORD, R., and SONI, D.: Applied Software Architecture, Addison Wesley, 2000.
- [6] EGYED, A.: Integrating Architectural Views in UML, Technical report, USC/CSE-99-TR-514, march 1999.

- [7] BOOCH, G., RUMBAUGH, J., and JACOBSON, I.: The Unified Modeling Language User Guide, Addison Wesley, 1999.
- [8] MEDVIDOVIC, N., and TAYLOR, R.N.: A Classification and Comparison Framework for Software Architecture Description Languages, IEEE Transactions on Software Engineering (TSE), Volume 26, Number 1, pp. 70-93, January 2000. Reprinted in *Rational Developer Network: Seminal Papers on Software Architecture*. Rational Software Corporation (July 2001).
- [9] SILVA, L.F., and de PAULA, V.C.C.: Software Architecture and UML, I Technical-Scientific Workshop – DIMAp 15 years, UFRN, Natal-RN, Brazil, 2000. (In Portuguese)
- [10] MARANHÃO, D.: Integrating ZCL Architecture Description Language with UML, Undergraduate Report, UFRN, Natal-RN, Brazil, June 2000. (In Portuguese)
- [11] ROBBINS, J., REDMILES, D., and ROSEMBLUM, D.: Integrating C2 with the Unified Modeling Language, From the Proceedings of the 1997 California Software Symposium, Irvine, CA, November 1997.
- [12] WOODCOCK, J., and DAVIES, J.: Using Z – Specification, Refinement, and Proof, Prentice Hall, 1996.
- [13] BOSCH, J.: Design & Use of Software Architectures, Addison Wesley, 2000.
- [14] SHAW, M., and GARLAN, D.: Software Architecture – Perspectives on an Emerging Discipline, Prentice-Hall, 1996.
- [15] OMG: The object model, <http://www.omg.org>
- [16] ORFALLI, R., and HARKEY, D.: Client/Server Programming with Java and CORBA, 2nd, Wiley, 1998.
- [17] SILVA, L.F., and de PAULA, V.C.C.: A Meta-model for Specifying Layer Software Architectures, XV Brazilian Symposium on Software Engineering (SBES2001), pp. 132-144, Rio de Janeiro-RJ, Brazil, October 03-05, 2001. (In Portuguese)
- [18] MEHTA, N., MEDVIDOVIC, N., and RAKIC, M.: Why Consider Implementation-Level Decisions in Software Architectures?, <http://sunset.usc.edu/~mehta/research/>
- [19] ROSENBLUM, D., and NITTO, E.Di: Exploiting ADLs to Specify Architectural Styles Induced by Middleware Infrastructures, 21st International Conference on Software Engineering (ICSE'99), Los Angeles, CA, pp. 13-22, 1999.
- [20] de PAULA, V.C.C., and BATISTA, T.V.: Mapping an ADL to a Component-Based Application Development Environment, 5th International Conference on Fundamental Approaches to Software Engineering (FASE 2002), Grenoble, France, pp. 128-142, 2002.
- [21] MILI, H., MILI, A., YACOUB, S., and ADDY E.: Reuse-Based Software Engineering – Techniques, Organization, and Controls, Wiley Inter-Science, 2002.
- [22] CLEMENTS, P., BACHMANN, F., BASS, L., GARLAN D., IVERS, J., LITTLE, R., NORD, R., STAFFORD, J.: Documenting Software Architectures: Views and Beyond, Addison-Wesley Professional, 1st edition, 2002.
- [23] MEDVIDOVIC, N., ROSEBLUM, D.S., REDMILES, D.F., ROBBINS, J.E.: Modeling Software Architectures in the Unified Modeling Language. *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 1, pages 2-57 (January 2002).
- [24] SILVA, L.F.: Using Meta-models to Integrate Patterns (In Portuguese), M.Sc. Thesis, Computer Science Department, Federal University of Rio Grande do Norte, Brazil, 2002.
- [25] de PAULA, V.C.C.: ZCL: A Formal Framework for Specifying Dynamic Distributed Software Architectures, PhD Thesis, Department of Informatics, Federal University of Pernambuco, Recife, Brazil, 1999.
- [26] de PAULA, V.C.C., JUSTO, G.R.R., CUNHA, P.R.F.: Formal Specification of Dynamic Architectural Styles. In: PDPTA 1999 – 1999 International Conference on Parallel and Distributed Processing Techniques and Applications, 1999, Las Vegas, USA. Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99). , 1999.
- [27] de PAULA, V.C.C., JUSTO, G.R.R., CUNHA, P.R.F.: Specifying and Verifying Reconfigurable Software Architectures In: PDSE 2000 - International Symposium on Software Engineering for Parallel and Distributed Systems, 2000, Limerick. Proceedings of International Symposium on Software Engineering for Parallel and Distributed Systems - PDSE 2000. , 2000. p.21 – 31
- [28] ROZANSKI, N., WOODS, E.: Software Systems Architecture, Addison-Wesley, 1st Edition, 2005.
- [29] UML Profile for CORBA: http://www.omg.org/technology/documents/formal/profile_corba.htm