




# Architectural Evolution Style Representation Model

Kadidiatou Djibo<sup>2</sup><sup>a</sup>, Mourad Chabane Oussalah<sup>1</sup><sup>b</sup> and Jacqueline Konate<sup>2</sup><sup>c</sup>

<sup>1</sup>LS2N - UMR CNRS 6004, Nantes University, 110 Bd Michelet, Nantes, France

<sup>2</sup>Faculty of Sciences and Techniques, USTTB, Bamako, Mali

**Keywords:** Software Architecture, Evolution Styles, Evolution, Model, Component.

**Abstract:** In this paper, we present a software architecture evolution process representation model following the principles of meta-modeling. The architecture evolution process is modeled as a software architecture evolution style. Then, we introduce an evolution style representation model in square. The model in square allows to represent the evolution process through four main dimensions of which : the evolution actor, the evolving architectural element, The evolution time and the evolution operation signature. Finally, we define a simplified formalism to express these architectural evolution with more convenience.

## 1 INTRODUCTION

The software architecture's objective is to provide an overview and a high level of abstraction in order to be able to understand a software system ((Shaw and Garlan, 1994)). The architecture proposes a system organization abstracted as a collection of software parts ((Perry and Wolf, 1992)). They have favored software mastery which is becoming more and more complex and distributed. These are often large-scale heterogeneous distributed software components, embedded systems, telecommunications, wireless ad hoc systems. As markets and technologies continue to evolve, these systems must also change to meet new market and technology requirements. The architecture offers a support for the control, the systems comprehension. So when a system needs to be changed, the ideal starting point for the evolution team is to understand the system (architecture), in order to try to find a set of suitable modifications. In this context, large-scale evolution planning is a very difficult activity that requires an understanding of the overall system structure, consideration of previous design decisions, principled trade-offs between candidate evolutionary scenarios, and optimal scenario selection (Barnes et al., 2013). Thus, the system architecture must also evolve to remain consistent with the systems documented as a guide.

The role of the software architecture in the software evolution process can be considered from two points of view: as an artifact for evolution, which guides the planning and conduct of the evolution process, and as an evolution artifact, because it must itself evolve in order to be consistent with the change in the system(Cuesta et al. (Cuesta et al., 2013)).


Indeed, the software architecture evolution is a very complex process to plan and, therefore, the person or team that plays this role must have a mix of architectural knowledge from different domains, including: business architecture, enterprise architecture, data architecture, application architecture and infrastructure or technical architecture (Hassan, 2018).


Therefore, architects need tools, methods and techniques that help them avoid the evaporation of knowledge about architecture's evolution (Oussalah et al. 2006 (Seriai et al., 2006)). This is especially true with respect to the sharing and reuse of this knowledge by architects who do not have this experience (Hassan, 2018).


This study is part of the exploration of prediction and planning in the component-oriented software architecture structural evolution.

The need for evolution becomes more necessary in component-oriented systems where the implementation is motivated by the principle of reuse. In a component-oriented system, one or more components or the whole system can evolve in order to be reused. Thus, the evolution can concern the source codes or the system architecture as a support.

The evolution of a software architecture is a com-

<sup>a</sup> <https://orcid.org/0000-0003-1916-7364>

<sup>b</sup> <https://orcid.org/0000-0001-8049-110X>

<sup>c</sup> <https://orcid.org/0000-0002-2599-7658>

plex task and requires several expertises. Many research efforts have aimed at modeling and reusing evolution in software architectures in order to capitalize information and foster knowledge sharing in the architecture community. The evolution styles introduced in (Seriai et al., 2006) are part of this context. Given the complexity of architectural evolution, it is not easy to have all the necessary skills within a company to make a software architecture evolve. It becomes necessary to go beyond the reuse offered by the evolution styles, but to anticipate the future evolutions from the good practices of previous evolution on similar architectures. This would considerably reduce the costs (in terms of skills, time, etc.) related to the evolution of these architectures and make them accessible to all.

We are convinced of the importance of integrating prediction and evolution planning into software architectures. Few works, to our knowledge, are devoted to the prediction and planning of evolution in software architectures.

The objective of this study is to help propose a solution to this lack. We are specifically interested in the problem of structural evolution prediction and planning in software architectures. For this, we use the evolution style key concept introduced by Oussalah et al. 2006, we apply data learning techniques in order to propose possible future evolutionary paths from previous evolution data (Figure 1).

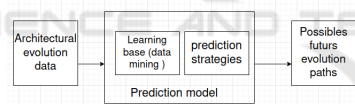


Figure 1: Overall goal.

Thus, this study takes place in three main phases. The first one consists in proposing a meta-model to represent the evolution of software architectures as a process, which we named evolution representation model. It is positioned at the  $M_2$  modeling level of the OMG above the architecture layer to represent its evolution. It has been positioned in relation to the standards defined for modeling the evolution of software architectures. This paper is devoted to the first phase which presents the new meta-model introduced to represent software architecture evolution as a process and the formalism introduced to collect evolution data. Given the number of pages allowed, the other two phases will be published further on.

This paper is organized as follows : Section 2 presents related work on software architecture evolution styles. The new evolution representation model introduced is presented in section 3. Section 4 describes the concepts of the model. In section 5

evolution style relationship is defined, then we give the evolution style representation formalism in section 6. Finally, we conclude the paper in section 7.

## 2 RELATED WORKS

In this study, we are mainly interested in the structural evolution of component-based software architectures. A software architecture structural evolution is reflected by the various changes in its structure and/or in that of its constituent elements. A change in an architecture is always generated by one or more operations applied to this architecture or to one of its elements, such as deletion, addition, modification. We refer to these operations as evolution operations (Sadou et al., 2005). A software architecture's structural evolution can be performed at the specification or design stage of the system it describes (static evolution) or during the execution of this system (dynamic evolution). In each of these cases, it is necessary to consider that any element of the software architecture can be brought to evolve. Thus, it is necessary to identify what can evolve, how to make it evolve, how to guarantee the coherence of the architecture having evolved and then how to pass on these changes in the software architecture to the system it describes. In general, to prepare a system or an architecture for evolution, it is necessary to : Be able to specify the need for evolution; Manage the impact of these changes; Establish the link between the starting model and the ending model. Thus, in (Sadou-Haririche, 2007) the authors introduced the three dimensions of the architectural evolution namely : the evolution object, the evolution type and the evolution support. The evolution's object makes it possible to specify the architectural elements to which the evolution relates. It can be any element reified and considered as a first class entity by the ADL of the software architecture to be evolved. Thus, according to the basic concepts of description of a software architecture, an evolution can concern a configuration, a component, a connector, and/or an interface. An evolution can concern the types of these elements and/or their instances, in other words it can concern the Architectural level or the Application level. The evolution type specifies the phase during which the evolution is executed.

According to Garlan et al. (Garlan, 2008), an evolution style expresses the software architecture's evolution as a set of potential evolution paths from the initial architecture to the target architecture. Each path defines a sequence of evolution transitions, each of which is specified by evolution operators. The (Cuesta et al., 2013) team has defined architectural

knowledge-based evolution styles (AKdES), which are also based on the design decisions of the architecture whenever an evolution step is made. Each evolution step is performed because an evolution decision is made following the verification of an evolution decision. From Oussalah et al.'s point of view (Oussalah et al., 2007), the main idea of an evolution style is to model the software architecture evolution activity in order to provide a reusable domain-specific evolution expertise. They consider an architectural evolution as consisting of architectural elements (component, connector, interface, etc.) modifications (addition, update, removal). The authors proposed a meta-evolution style called MES (Meta Evolution-Style)(Hassan and Oussalah, 2018) to unify the modeling concepts that formulate evolution styles.

### 3 EVOLUTION REPRESENTATION MODEL

In order to model, standardize, formalize and remain consistent in the description of evolution styles and to promote the reuse of these styles, an evolution meta-style called MES has been introduced in (Hassan and Oussalah, 2016). MES is positioned at the  $M_3$  modeling level of the OMG and defines a generic meta-model that represents an architectural evolution as a process by specifying the role (actor), the element to be evolved and the evolution operation. Thus, any evolution style language must conform to MES, more precisely must be an instance of MES that defines the normal framework for representing, describing and modeling a reusable architectural evolution. In this paper, the introduced representation model is positioned at the  $M_2$  modeling level of the OMG as an instance of the MES and defines an architectural evolution as a process.

The evolution style meta-model that we propose should allow the transmission of good evolution practices, promote the reuse of software architecture evolution and will provide a normal framework for the definition of evolution styles. Thus, it must answer two main requests : (1) compliance to MES and (2) The evolution expectations through the following four questions, What ?(what is evolving ?), Who ?(who makes it evolve ?), When ?(when to evolve it ?) and How (how to evolve it ?). These four questions describe the evolution activity. In answering them, our meta-model describes an architectural evolution as a process.

*What (what is evolving) ?*

It is the basic element (input/output) of any evolution operation. The evolution concerns all or a targeted

part of the architecture. In the first case, the entire architecture is affected. In the second case, only the designated parts will be affected. Thus it remains important to better reify the architecture, better represent each of its components and their different configurations. Thus, the meta-model answers the question *What?* through the package EvolvingArchitectureElement.

*Who (who makes it evolve) ?*

It is a set of responsibilities, i.e. it describes the skills, tools and techniques required to perform a specific architecture evolution operation. Thus, this question is answered through the Actor concept.

*When (when to evolve it) ?*

The management of the schedule and the chaining, i.e. defining which operation before or after (the schedule of operations) remains necessary in the definition of the process models. The TimeEvolution concept solves this issue. It allows to manage the steps and to plan the evolution operations.

*How (how to evolve it) ?*

It is about defining how to produce visible changes in the state of the architecture. It is solved from the concepts EvolutionStyle, Header, Competence, Action and Impact.

Taking these concerns (questions) into account allows the meta-model to define an evolution style as a process. Through the class diagram of the figure 2, we highlight the concepts and inter-concept relationships of the model.

### 4 DESCRIPTION OF THE EVOLUTION REPRESENTATION MODEL CONCEPTS

An architectural evolution style is described by a process specifying the activity, the role and the product (evolving element) as shown in Figure 2. Thus, the meta model is defined in three main parts. The concepts associated with each part (Fig.2) are defined.

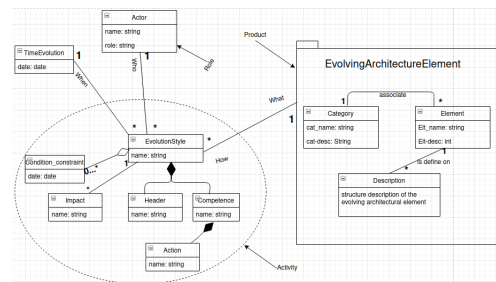


Figure 2: Evolution representation model.

## 4.1 Product

The EvolvingArchitectureElement package allows to represent the architectural element, its category, its ports and its connections. It tries to give a complete representation of the architectural element in order to pass easily to a graphic interpretation of the architecture. Thus, the Category concept allows to group and classify architectural elements of the same type and function. The Element class is used to represent architectural elements and their properties. Through the Description concept, the structure of the architectural element is defined. Thus, the EvolvingArchitectureElement package proposes to offer a complete textual description of the graphical representation of the architecture in order to promote an automatic transition from the graphical representation to the textual or vice versa.

## 4.2 Role

Defined by the Actor concept, it describes the actor of the evolution and the skills needed to perform the evolution operation being defined. An actor can intervene in several evolution operations. It can refer to a physical person or a computer program.

## 4.3 Evolution Operation (Activity)

It is one of the units of the process that produces visible changes in the state of the architecture. It is associated with the roles and architectural elements. It is described through the concepts EvolutionStyle, Evolution\_constraint, Impact, Header, Competence, Action.

### 4.3.1 Evolution Style

An evolution style encapsulates that which allows to describe and apply an evolution to an architectural element. The EvolutionStyle class is a named entity that is composed of two complementary parts : a header defined through the Header concept and a skill defined by the class Competence. The header has an informal description of the purpose and publishes a list of parameters and assertions. The evolving element appears as an implicit parameter named context. The type of the parameters is provided by the set of evolutive elements, plus the usual primitive types (String, int, boolean, float, etc.). The competence represented by the Competence concept describes a unit of implementation corresponding to the header. The implementation unit specifies the data flow and all control logic. We have extended the EvolutionStyle concept by adding the notion of condition constraint

which defines or establishes the conditions of execution of an evolution operation on a given architectural element. Its definition is not mandatory and there can be several condition constraints to check for a given operation. It is described through the concept Condition\_constraint.

An evolution style can cause or invoke another evolution style, the Impact class allows to specify the evolution styles invoked by the execution of the current style.

## Evolution Style Representation

Figure 3 schematizes the representation of a style according to four compartments : the style name, its header, its competence and impact.

Every evolution operation is described as an evolution style. An evolution style is in relation with other evolution styles. This relationship is managed by the Impact concept which specifies the invoked styles and the relationship type.

|            |  |
|------------|--|
| Style name | Proper name or style label   |
| Header     | Context : (refers to the evolving architectural element)<br>Pre-condition : (defines condition constraints and invoked styles) |
| Competence | Description<br>Actions: (All actions necessary for the execution of the style)   |
| Impact     | Post-condition   |

Figure 3: Evolution style representation.

### 4.3.2 TimeEvolution

The TimeEvolution class associates an execution delay to an evolution operation on an evolving architectural element. It allows us to plan and manage the steps in the management of evolutions.

## 5 EVOLUTION STYLES RELATIONSHIP

The operational mechanisms provided by our model are instantiation, specialization, composition, and finally usage. As these mechanisms are largely inspired by those of the object-oriented approach, we can therefore use the notations of the UML class diagram for the illustrations to follow in this section. Figure 4 illustrates the operational mechanisms of the evolution representation model.

In a general way, instantiation is a mechanism that allows you to move from a given modeling level to a lower level. Evolution styles can be instantiated several times in an architecture. The instance of an evolutionary style is a particular process that is created within the structure given by its style. All instances of an evolution style must offer exactly the same skill

as that style. The instantiation mechanism covers the binding of formal parameters to effective parameters (i.e., elements of an architecture), the evaluation of the precondition and postcondition, and the execution of the skill body. Therefore, an abstract evolution style cannot have instances since it does not specify any competence. We can consider that an instance is attached to its evolution style type by the relation "is-a".

Evolution styles can be defined by extension of other styles. The inheritance mechanism associated with the specialization relationship is inspired by the class inheritance mechanism in the object paradigm. With the proposed bipartite structure, on the one hand a sub-style can add and override elements of the header of its super-style, and on the other hand a sub-style can redefine the competence of its super-style. This mechanism can be used to define concrete evolution styles as sub-styles of abstract styles by providing the missing skill.

Composition is necessary to describe developments at different levels of detail. Style composition refers to the "all-part" structuring between two styles. At each level of composition, each style can be seen as having as parts those sub-styles which represent stages entering its competence. This relationship suggests a strong coupling between evolution styles and promotes the encapsulation of complex skills. Our model uses the composition mechanism to define composite evolution styles, increasingly complex, delegating their functionality to component styles. The composition starts from a set of primitive evolution styles, whose competence is elementary. Finally, the communication mode assigned to the composition is necessarily synchronous, because the execution of the component styles is subordinated to that of the composite style.

In the evolution representation model, utilization is a fundamental technique in the perception of an "expert" system as a set of interrelated expertises. Thus, usage is a relationship that allows a style to reference another style to use its functionality. It should not be confused with a compositional relationship, as it has a more momentary character and does not involve strong coupling. Finally, the mode of communication attributed to the use is asynchronous, because the executions of the styles do not necessarily have to be concordant.

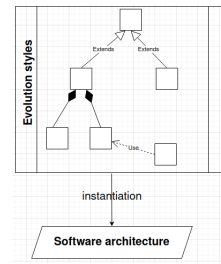


Figure 4: Evolution styles relationship.

## 6 FORMALISM FOR REPRESENTING EVOLUTION STYLES

The specification of evolution styles should be as simple as possible, both in terms of the concepts manipulated and the quantity of elements to be described. Taking into account the functionalities offered by the evolution representation model, the chosen formalism must make it possible to distinguish the competence header of an evolution style, the evolving architectural elements, the actor and the associated evolution date. This keeps the style integrity as a process. In addition, the architect must be able to prioritize headers, skills, and evolving elements and have multiple views of his styles.

The Y-model (MY) is a formalism that naturally incorporates certain functionalities and has also been used in (Smeda et al., 2008) to describe component-based software architectures. It has been used as a support, to describe an evolution style by three aspects (corresponding to the three branches of the Y) : domain, header and competence. These aspects represent successively all that is related to the evolving architectural elements, to the operations of evolutions and to the implementations of these operations (Le Goaer et al., 2008). Given the features offered by the evolution representation model we introduced in this paper, the Y model previously used in (Goaer, 2009) does not allow us to properly represent our model. Thus, we introduce the model in square to represent an evolution style modeled as a process, distinguishing the skill header of an evolution style, the evolving architectural elements, the actor and the associated evolution time. The four elements mentioned correspond to the four vertices of the square (Figure 5).

### 6.1 The Model in Square of Style Representation

Using the square model (Figure 5, Figure 6), an evolution style can be described by four aspects which are: the evolving architectural element, the evolution operation header, the evolution actor and the evolution date. These four elements are linked together to establish an evolutionary style. They are defined for the purpose of reuse to describe different evolution archetypes.

As said before, the three main concepts of the description of an evolution style are the evolving architectural element, the role (actor) and the evolution operation. We add a fourth concept, not explained until now, which is the date and/or the deadline of the evolution operation. The evolving architectural element provides a vocabulary of evolving elements and their relationships, typically in the form of classes and attributes in object representations. The acquisition of this basic conceptual vocabulary is therefore necessary to express the evolution knowledge on this domain. We model these concepts using four aspects (each represented by a vertex of the square): EvolvingArchitectureElement, Header, Actor and TimeEvolution. Each vertex of the square describes an evolution style concept, as shown in the Figure 5.

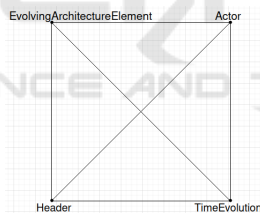


Figure 5: Evolution style concepts representation using the model in square.

The diagonals of the square allow to represent the concepts of the evolution style. The center of the square corresponding to the intersection point of the two diagonals represents the architecture on which the styles are defined. The evolution styles expressed by the four concepts at the vertices are described using different levels of abstraction. Each level of abstraction is represented by a quadrilateral in the square model, from the highest level (perfect square) to the lowest level (other quadrilateral shape). The level of representation of a concept is deduced by its position relative to the center of the square. A concept represented near the center is said to be weakly represented. A concept located near the top is said to be strongly represented. In the middle, it is said to be moderately represented (Figure 6).

When we observe the representations of styles S1

and S2 (Figure 6), the actor A1 is better represented than A2, in other terms A1 is strongly represented than A2 with respect to certain criteria defined on the two elements (competence, experience, etc.) for example A1 is more competent than A2, the style S1 is defined by an actor more competent than the style S2. Similarly, if we take E1 and E2, the criteria can be the age of the element, the previous evolution of the element, etc. E1 has evolved several times compared to E2, the S1 style is defined on an architectural element which is more sensitive because it is much affected by the evolution operations than the S2 style. TimeEvolution defines the earliest and latest evolution time. Thus, T2 is more urgent than T1, in other words the S2 style would be more urgent than S1.

According to the criteria, the model in square makes it possible to draw hypotheses on the behavior of the styles.

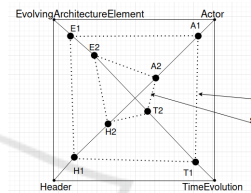


Figure 6: Evolution style concepts representation using the model in square.

Thus, from the square model, we derive the simplified expression of evolution style.

### 6.2 Simplified Expression of Evolutionary Style

The introduced evolution style meta-model defines the style modeling language. In order to provide a formalism to express evolution styles with more convenience and entirely based on the introduced meta-model, we have introduced a simplified expression to express software architecture evolution styles. Thus, the simplified expression provides the framework for expressing an evolution style while remaining consistent with the representation model. It allows to define the process parameters. It is presented as a quadruplet defining the actor, the evolving architectural element, the operation header and the execution date. The evolving architectural element is presented as a couple of elements and its category. The header is the transaction's signature and allows the transaction to be uniquely identified. Figure 7 gives us an overview of the simplified expression. This expression allows to name and express one by one all the evolution styles of an evolving software architecture.

Style-name : < Actor, (Element, Category), Header, TimeEvolution >

Figure 7: Evolution style simplified expression.

### 6.3 Interests and Benefits of Simplified Formalism

The simplified formalism allows to build an evolution style library that can be submitted to analyses in order to infer on the evolution process of software architectures. Indeed, the work done in this paper leads us to develop a model for planning and predicting architectural evolution. For this purpose, the style library obtained through the simplified formalism is subjected to analysis.

Sequential pattern extraction techniques can be applied to the style library in order to discover the sequences of recurrent evolution operations, the architectural elements more or less affected by the evolution operations, the evolution actors more or less solicited and several other types of information. This information can be used to build a learning base to predict and plan future architecture evolution by learning from past facts and data. This would allow to anticipate the evolutions and to reduce considerably the costs (competence, delay).

## 7 CONCLUSION

In this paper we have presented the evolution style meta-model introduced to represent the software architecture evolution process. It models the evolution as a process by specifying the activity, the evolving product, the role and the execution date of the operation. The evolution representation model provides the necessary concepts for specifying and properly managing software architecture evolution independently of the architecture model and any ADL. Thus, it considers the different modeling levels of a software architecture and the need to manage the evolution through these different levels. In addition, we introduced the model in square which, based on the introduced evolution representation model, provides an evolution style representation framework with abstraction levels and a simplified software architecture evolution style expression in order to easily collect evolution data while respecting the model policy. These data collected through the simplified expression can be submitted to studies or analyses in order to infer on evolution styles.

The results obtained lead us to plan and predict the future evolution paths of an evolving software architecture from the previous evolution data collected

according to the presented model. From the previous evolution data of a software architecture evolving in time  $A_1$  towards  $A_n$ , the aim is to elaborate a training base in order to predict the possibilities and the skills required to evolve towards  $A_{n+1}$ . This work will be developed in a future study and will facilitate evolution management in software architectures with a good management of resources and a better capitalization of information in the architect community. These results can also be applied to other artifacts.

## REFERENCES

- Ahmad, A., Pahl, C., Altamimi, A. B., and Alreshidi, A. (2018). Mining patterns from change logs to support reuse-driven evolution of software architectures. *Journal of Computer Science and Technology*, 33(6):1278–1306.
- Barnes, J. M., Pandey, A., and Garlan, D. (2013). Automated planning for software architecture evolution. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 213–223. IEEE.
- Breivold, H. P., Crnkovic, I., and Larsson, M. (2012). A systematic review of software architecture evolution research. *Information and Software Technology*, 54(1):16–40.
- Cuesta, C. E., Navarro, E., Perry, D. E., and Roda, C. (2013). Evolution styles: using architectural knowledge as an evolution driver. *Journal of Software: Evolution and Process*, 25(9):957–980.
- Falcarin, P. and Alonso, G. (2004). Software architecture evolution through dynamic AOP. In Oquendo, F., Warboys, B., and Morrison, R., editors, *Software Architecture, First European Workshop, EWSA 2004, St Andrews, UK, May 21-22, 2004, Proceedings*, volume 3047, pages 57–73. Springer.
- Filho, J. W., de Figueiredo Carneiro, G., and Maciel, R. S. P. (2019). A systematic mapping on visual solutions to support the comprehension of software architecture evolution. In Jr., J. J. P., editor, *The 25th International DMS Conference on Visualization and Visual Languages, DMSVIVA 2019, Hotel Tivoli, Lisbon, Portugal, July 8-9, 2019*, pages 63–82. KSI Research Inc. and Knowledge Systems Institute Graduate School.
- Garlan, D. (2008). Evolution styles-formal foundations and tool support for software architecture evolution. *Computer Science Department, reports-archive.adm.cs.cmu.edu*, page 650.
- Gasmallah, N., Amirat, A., Oussalah, M., and Seridibouchelaghemi, H. (2019). Developing an evolution software architecture framework based on six dimen-

- sions. *International Journal of Simulation and Process Modelling*, 14(4):325–337.
- Goaer, O. L. (2009). *Styles d'évolution dans les architectures logicielles. (Evolution styles within software architectures)*. PhD thesis, University of Nantes, France.
- Hassan, A. (2018). *Style and Meta-Style: Another way to reuse Software Architecture Evolution. (Style et Meta-Style: Une autre façon de réutiliser l'évolution dans les architectures logicielles)*. PhD thesis, University of Nantes, France.
- Hassan, A. and Oussalah, M. (2016). Meta-Evolution Style for Software Architecture Evolution. In *SOFSEM 42th International Conference on Current Trends in Theory and Practice of Computer Science-LNCS*, Harachov, Czech Republic.
- Hassan, A. and Oussalah, M. C. (2018). Evolution styles: Multi-view/multi-level model for software architecture evolution. *JSW*, 13(3):146–154.
- Kouroshfar, E., Mirakhorli, M., Bagheri, H., Xiao, L., Malek, S., and Cai, Y. (2015). A study on the role of software architecture in the evolution and quality of software. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 246–257. IEEE.
- Le Goaer, O., Tamzalit, D., Oussalah, M. C., and Seriai, A.-D. (2008). Evolution styles to the rescue of architectural evolution knowledge. In *Proceedings of the 3rd International Workshop on Sharing and Reusing Architectural Knowledge, SHARK '08*, page 31–36, New York, NY, USA. Association for Computing Machinery.
- Oussalah, M., Goaer, O. L., Tamzalit, D., and Seriai, A. (2007). Evolution styles in practice - refactoring revisited as evolution style. In Filipe, J., Shishkov, B., and Helfert, M., editors, *ICSOF 2007, Proceedings of the Second International Conference on Software and Data Technologies, Volume SE, Barcelona, Spain, July 22-25, 2007*, pages 138–143. INSTICC Press.
- Perry, D. E. and Wolf, A. L. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software engineering notes*, 17(4):40–52.
- Plakidas, K., Schall, D., and Zdun, U. (2018). Model-based support for decision-making in architecture evolution of complex software systems. In *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, pages 1–7.
- Sadou, N., Tamzalit, D., and Oussalah, M. (2005). How to manage uniformly software architecture at different abstraction levels. In *International Conference on Conceptual Modeling*, pages 16–30. Springer.
- Sadou-Harireche, N. (2007). *Evolution Structurelle dans les Architecture Logicielles à base de Composants*. PhD thesis, PhD thesis, Université de Nantes.
- Seriai, A., Oussalah, M. C., Tamzalit, D., and Goaer, O. L. (2006). A reuse-driven approach to update component-based software architectures. In *Proceedings of the 2006 IEEE International Conference on Information Reuse and Integration, IRI - 2006: Heuristic Systems Engineering, September 16-18, 2006, Waikoloa, Hawaii, USA*, pages 313–318. IEEE Systems, Man, and Cybernetics Society.
- Shaw, M. and Garlan, D. (1994). An introduction to software architecture. *School of Computer Science Carnegie Mellon University Pittsburgh, PA*.
- Smeda, A., Oussalah, M., and Khammaci, T. (2008). My architecture: a knowledge representation meta-model for software architecture. *International Journal of Software Engineering and Knowledge Engineering*, 18(07):877–894.