A Framework for Classifying and Comparing Software Architecture Tools for Quality Evaluation

Eudisley Anjos, Mário Zenha-Rela

CISUC, Centre for Informatics and Systems
University of Coimbra, Portugal
{eudis,mzrela}@dei.uc.pt

Abstract: Software quality is a crucial factor for system success. Several tools have been proposed to support the evaluation and comparison of software architecture designs. However, the diversity in focus, approaches, interfaces and results leaves the researcher and practitioner wondering what would be the most appropriate solution for their specific goals. This paper presents a comparison framework that identifies the most relevant features for categorizing different architecture evaluation tools according to six different dimensions. The results show the attributes that a comprehensive tool should support include: the ability to handle multiple modelling approaches, integration with the industry standard UML or specific ADL, support for trade-off analysis of competing quality attributes and, the reuse of knowledge through the build-up of new architectural patterns. This comparison is able to, not only guide the choice of evaluation, but also promote the development of more powerful tools for modeling and analysis of software architectures.

Keywords: software quality challenges; trends for achieving quality objectives; architecture evaluation process; **s**oftware architecture; Affidavit.

1. Introduction

In the last years we have witnessed a growing interest in software architecture design driven by the unavoidable need to apply it in the development of large and complex software systems [1], [2]. The design of Software Architecture (SwA) is one of the earliest stages of system development. Ensuring that all requirements are fulfilled leads to a better implementation of the system [3]. However, ensuring the correctness of software architecture is not simple. Several tools and methods have been proposed to help the architects design and evaluate the system requirements right from an early architecture model.

System requirements are collected from stakeholders (e.g. end-users, management, marketing, third-parties, and developers) and include functional and nonfunctional attributes. Functional requirements typically describe the anticipated behavior of a solution or solution components, and a huge body of knowledge and literature on how to collect, model, evaluate and verify them exist [4], [5]. Non-functional attributes, on the other hand describe properties of the system e.g. maintainability, usability, usability, efficiency, portability and reliability, etc [6]. There is no consensus about a standard methodology on how to assess quality in SwA, and the lack of a common taxonomy (despite the IEEE std. 1061-1998) also constrains the development of widespread evaluation methods and tools. Moreover, architects can define specific attributes based on their context, which implies that the supporting methodologies are abstract enough to fit into their specific architecture design. So, the architect is expected to understand the requirements of stakeholders, assess the quality attributes, and choose the best evaluation method and tool to fit in the design process.

There are several methods in literature to evaluate software architecture quality namely: SAAM, Software Architecture Analysis Method [1], [7], with a focus on modifiability; ATAM, Architecture Trade-off Analysis Method [1], [8], mainly used to modifiability but also applied to other quality attributes evaluation and trade-off verification; ARID, Active Reviews for Intermediate Design [9], CBAM, Cost Benefits Analysis Method [10], FAAM, Family Architecture Assessment Method [11]. Most of these methodologies describe a set of steps that should be followed in order to check and evaluate the quality of the architectural design.

There are an increasing number of alternative approaches all of which make the architect's choice more difficult. Some researchers have proposed a taxonomy framework to classify and compare software architecture evaluation techniques, selecting key features of each methodology to categorize them. In [12] is proposed a framework to compare and assess eight SA evaluation methods (most of the scenario-based) and demonstrate the use of the framework to discover the similarities and differences among these methods. In [13] the authors describe the main activities in model checking techniques defining a classification and comparison framework and, in [14], the focus is on evaluation of ten models to assess performance, maintainability, testability and portability. These works mentioned before focus essentially on model checking, simulation-based or scenario-based approaches.

The architect can use evaluation tools (such as the tools evaluated in section 4 below) to support the application of methods, reducing the time spent on analysis and improving the results of the evaluation process. These tools perform different types of evaluation depending on the method used and which design features are the focus of assessment. Many architecture evaluation tools have been proposed but most of them are limited to a specific purpose (e.g. for deployment only) or propose a generic approach but developing only a subset of functionalities for demonstration purposes. Still, there is a lack of knowledge about what the relevant features should be addressed (e.g. description language used by the tool, quality parameters assessed, evaluation method, etc.). This makes difficult to choose the tool that best suits the architectural design process.

Some researchers have tried to assess architecture tools based on the evaluation techniques adopted and comparing them across different tools. However, little work has been done to classify and compare evaluation tools from a generic perspective, describing the main characteristics and assisting the perception of which are the relevant features these tools should provide.

The purpose of this paper is to present a comparison framework that enlights the most relevant attributes to help categorize the different architecture evaluation tools. We think this goal is relevant to the researchers and practitioners in the area as there is a growing number of methods and tools and it is easy to get overwhelmed by the seemingly diversity of options, such as ArchKriti [15], used to support crucial steps in architecture-based software development and, Attribute Driven Design method (ADD) [16], used to design software architecture of a system or collection of systems based on an explicit articulation of the quality attribute goals for the system. Moreover, since we are involved in a major international effort (EU/USA) to build a comprehensive architectural development support tool (AFFIDAVIT) [17], it is relevant to organize the current approaches to identify their major strengths and learn from their weaknesses. We do think that proper tools based on sound methodologies will bring us closer to the widespread adoption of tools that support architectural design, thus leading to an increase in the quality of software.

The reminder of the paper is organized as follows. In section 2 we present the fundamental role of quality attributes in the definition of software architecture and in section 3 the current software architecture evaluation categories. In the following section we discuss the selected architecture evaluation tools and which attributes were considered most relevant. In section 5 we compare the selected tools according to the identified relevant attributes and present the major insights and the current work on the Affidavit tool. Section 6 closes the paper by presenting some future work.

2. Quality Attributes and Software Architecture

The non-functional requirements of a Software Architecture play a key role in terms of the specifying the design patterns and deployment architectures of the solution being built [18]. For example, poor network connectivity will dictate requirements in terms of a centralized vs. decentralized deployment, requirements for resilience may dictate a requirement for clustered hardware, etc. To implement these architectural decisions and thus to design the whole architecture it is essential that the architect knows its most relevant quality attributes (e.g. modifiability, performance, security, availability.).

It is significant to consider that the two most important sources of quality attributes and requirements are the stakeholder's needs and the application domain [19], [20]. The architect needs to have a good understanding of these sources and their influence on the system to model the correct architecture to support the requirements. However, depending on the type of software and its context, different kinds of systems have different relevant quality attributes. In a complex long-lived application such as data-mining and network analysis, maintainability might be the most significant quality attribute while in a real-time sophisticated network, performance may drive most of the architectural decisions

It is necessary to ensure the fulfillment of the quality parameters, although they are contradictory quite often (e.g. security versus usability, performance versus maintainability) [21]. According to Barbacci [22] designers need to analyze trade-offs among multiple conflicting attributes to attend the user requirements. The goal is to evaluate the interrelationships among quality attributes to get a better overall system (the best compromise for a given context). In order to help this kind of analysis, most of the evaluation methods include some sort of trade-off analysis.

So, evaluating the quality of a software architecture is a critical step to ensure that the software meets its design objectives in a reliable and predictable fashion. However the architect needs support to perform this activity. It is almost impossible for an architect validate manually the whole system model, verifying if the required quality attributes are effectively supported by the architecture and assuring that there are no conflicts between them. That is why the adoption of several methods, tools and ADLs are fundamental to help the architect in designing, specification and analysis of SwA.

3. Architecture Evaluation Methods

Any serious software architecture evaluation process needs to consider and categorize several different architectural aspects of the system's requirements (e.g. kind of requirements, architectural description, etc.). Depending on how these aspects are addressed by the evaluation methods, it is possible to identify different evaluation methods categories. Regardless of category, the evaluation methods can be used in isolation, but it is also possible and common to combine different methods to improve the insight and confidence in the architectural solution to evaluate different aspects of the software architecture, if needed. After deciding for a specific evaluation method, the architect has to select the Architecture Description Language (ADL), tool and the best suited technique to her or his specific project.

In this work is adopted the architecture evaluation methods' categorization carried in [14] and [23]. The authors consider four evaluation categories: *scenario-based*, *formal-modeling*, *experience-based* and *simulation* based. Other authors (e.g. [13]), use model-checking to address techniques which verify whether architectural specifications conform to the expected properties. Since in this paper we consider that all presented categories are used to in assess the architectural model, model-checking is not inserted as a different evaluation category.

Below we present a brief characterization of each category:

- **Scenario-Based:** Methods in this category use operational scenarios that describe the requirements to evaluate the system quality. The scenarios are used to validate the software architecture using architectural tactics and the results are documented for later analysis including system evolution, maintenance and the creation of a product line. There are several scenario-based evaluation methods namely SAAM [1], [7], ATAM [1], [8], CBAM [10], FAAM [11].
- **Formal-Modeling:** Uses mathematical proofs for evaluating the quality attributes. The main focus of this category is the evaluation of operational parameters such as performance. An example of formal-modeling is NOAM (Net-based and Object-based Architectural Model) [24]. Usually, the use of formal-modeling and simulation-based methods can be joined to estimate the fulfillment of specific qualities.
- **Experience-based:** The methods in this category use previous experience of architects and stakeholders to evaluate the software architecture [25]. The knowledge obtained of previous evaluations is maintained as successful examples to design new similar solutions and drive further architecture refinements.
- **Simulation-based:** Uses architectural component implementations to simulate the system requirements and evaluate the quality attributes. The methods in this category can be combined with prototyping to validate the

architecture in the same environment of the final system. Examples include LQN[26] and RAPIDE [27].

It is important to notice that the evaluation categories are not directly linked to the evaluation tools. They specify how to apply the theory behind the tools and commonly define steps to assess the SwA quality. Some tools support different methods to get a better insight. In fact, very few ADLs, like Aesop [28], Unicon [29] and ACME [30], provide support for different evaluation processes. They are closely serving as an evaluation tool themselves and assist in modeling specific concepts of architectural patterns, although unfortunately in most cases these are applicable to restricted purposes only.

4. Architecture Evaluation Tools

The diversity of techniques focusing on restricted contexts and attributes turns the selection of an architectural method into a complex task. Architects typically need to adapt several models and languages depending on the attributes they want to evaluate. Thus, it is necessary to know different description languages, scenarios specification, simulation process, application contexts and others method's features to make the best choice and perform the intended evaluation process. While generic tools do not become widespread, we have observed that architects tend to adopt the methods, tools and ADL that they have previously been in contact with.

Many evaluation methods (e.g. ATAM), describe a sequence of manual activities that the architect should perform to identify the main issues concerning quality assessment. Based on these descriptions or instructions, software tools are used to automate only parts of the evaluation process, such as: architectural scenario definition, analysis of architectural components relationships and others. Automating the whole validation of architectural quality requires the mediation of the architect to tailor the model according to the requirements and to solve errors and conflicts detected by the tools.

ADLs also evolved and new features were assembled to aid the architect. According to [31] the tools provided by an ADL are the canonical components (referred also as *ADL toolkit* [32]). Among these components we can mention *active specification,* a tool that guides the architect or even suggests wrong behaviors in the design and, *architectural analysis,* which is the evaluation of the system properties detecting errors in the architectural phase and reducing costs of corrections during the development process. We have take into account that most tools have requirements that hamper the use of ADL already known by the architect, forcing the learning of new architecture description languages or requiring design the model directly at the interface of the tool.

Many characteristics have to be managed and balanced by the architect, thus selecting the best tool is not a simple task. Sometimes, it is necessary that the architect knows how the tool works to decide whether is useful in a specific project. The system requirements guide the architect about the type of tool to be used but still lack of specific information about methods and features or this information is dispersed, preventing a sound and informed *a-priori* evaluation by the architect.

There are few studies to assist the architect in the selection of the best tool to support the architectural quality evaluation. In [33] for example, the authors compare different knowledge management tools to understand SwA knowledge engineering and future research trends in this field. The knowledge about how to assess a particular kind of system requires that the architect knows what characteristics the tool should have.

In this paper we present a survey of different types of architecture evaluation tools and classify them according to the six dimensions presented in Table 1 below. The goal is to identify significant dimensions to analyze the applicability of an evaluation tool in a particular context to a specific goal.

Table 1. Dimensions used for the evaluation of tools.

Feature	Description
Method	The evaluation method used. One tool can support several methods.
ADL	Assess if the tool has its own ADL, use another know ADL or if the architect must describe the architecture manually using the tool interface.
Qualities	Indicate which quality attributes are covered by the tool.
Trade-off	The ability of the tool to understand and measure the trade-offs between two or more quality attributes.
Stakeholders	Indicate if the tool supports the participation of stakeholders during the architecture evaluation or somewhere during the architecture design.
Knowledge	Evaluate if the tool preserves the knowledge (e.g. architectural patterns) since the last architectural evaluation for performing new designs.

5. Assessment of tools

During this work we carried out an extensive survey and analysis of the literature to identify the most relevant tools and understand their focus. Afterward we selected five tools as being more representative and identified six dimensions to present what we think is the essence of the current results. In this section is presented an assessment of these tools using the selected dimensions.

We tried to cover the different methods of evaluation supported by the selected tools. However, it is important to understand that several tools use multiple methods to achieve the proposed objectives. This is especially true when the tool is more generic and supports different techniques according to the attribute which is under evaluation. The tools selected as most relevant due to their maturity and impact in the literature (most are based on research work, even if used in industry) are ArchE design assistant [34], Architecture Evaluation Tool (AET) [35], Acme Simulator [36], ArcheOpterix [37] and DeSi [38]. We shall now present a brief description of each tool, as well as a table summarizing their profile according with the dimensions selected.

5.1. ArchE Design Assistant

This tool is an Eclipse plug-in that manages reasoning frameworks (RF) to evaluate software architectures. The evaluation models are the knowledge sources and the Architecture Expert (ArchE) baseline tool manages their interaction. A relevant point of this *assistant* is that a researcher can concentrate on the modeling and implementation of a reasoning framework for the quality attribute of interest. The tool has no semantic knowledge and consequently supports any reasoning framework. So, ArchE is an assistant to explore quality-driven architecture solutions.

The ArchE receives from each RF a *manifesto*. The *manifesto* is a XML file that lists the element types, scenarios and structural information handled by the reasoning framework. In ArchE many actors collaborate to produce the solution of a problem. In this case, the actors are the RF and every other actor can read information provided by other RFs. This communication can produce new information useful to some of them.

The flexibility of ArchE is its major strength but also its major weakness: researchers and practitioners are able/forced to develop or adapt their own quality-attribute model if not already supported. An input conversion for non supported ADLs might also be necessary.

As the ArchE authors state:

"*ArchE is not intended to perform an exhaustive or optimal search in the design space; rather, it is an assistant to the architect that can point out "good directions" in that space.*"

Note that ArchE is most useful during the assessment phase of architectural development.

Table 2. Dimensions evaluated for ArchE

Dimensions	ArchE behaviour
Method	This tool uses scenarios-based method but each RF can have its own evaluation method.
ADL	There is no specific ADL linked with this tool, only the XML file used as *manifesto.*
Qualities	All quality attributes can be evaluated, depending which RF is used
Trade-Off	The trade-off among different RFs uses a 'traffic-light' metaphor to indicate potential scenario improvements when applying different tactics.
Stakeholders	It is an architect-focused tool; other stakeholders are only involved when scenarios are identified.
Knowledge	ArchE does not build knowledge from past evaluation (architectural patterns) to apply in new projects.

5.2.Architecture Evaluation Tool (AET)

AET is a research tool developed at BOSCH to support the evaluation team in documenting results and managing information during an architecture review. AET has two databases to assist the architect with the information management: General, (i.e. static data) and Project (i.e. dynamic data) databases. This tool is present during the gathering of quality attributes information and architectural scenarios. So, the information obtained from stakeholders is stored in AET databases. The information is used in the current project and storage in the general database for new architectural projects. Although this tool was initially developed to evaluate performance and security, the project is still under development to include more attributes. This tool is focused in the initial phases of requirements gathering and quality attributes trade-off analysis.

Table 3. Dimensions evaluated for AET.

Dimensions	AET behaviour
Method	This tool use scenarios as main method for evaluation. It uses both dynamic and static (experience-based) evaluation types.
ADL	There is no ADL linked with this tool. The data is inserted directly using the tool interface.
Qualities	All quality attributes can be evaluated as it is a mostly manual processing.
Trade-Off	The trade-off is performed based on the data introduced by the architect and stakeholders during the achievement of quality attributes and scenarios. The tool combines this information to guide the architect showing the risks and the impact of changes.
Stakeholders	In the initial steps of data gathering, the tool requires that stakeholders participate to fill the quality requirements (scenarios).
Knowledge	The knowledge (experience repository) is stored in database as input to new evaluations.

5.3.Acme Simulator

This tool is an extension of AcmeStudio (a plug-in of the Eclipse Framework) and uses its existing features for defining architectural models. Also provides specific architectural styles to specify relevant properties and topology to the kind of analysis. The Acme simulator as originally developed provides security and performance analysis using Monte Carlo simulation to evaluate the properties under certain assumptions about their stochastic behaviour. Since the simulator is written in AcmeStudio framework, the Eclipse allows flexible extensions and the tool uses Acme as design ADL to model the software architectures. This tool is clearly focused on architectural assessment.

5.4.ArcheOpterix

This tool provides a framework to implement evaluation techniques and optimization heuristics for AADL (Architecture Analysis and Description Language) based specifications. The algorithms should follow the principle of model-driven engineering, allowing reason about quality attributes based on an abstract architecture model. The focus of ArcheOpterix is embedded and pervasive systems.

The quality evaluation is represented using AttributeEvaluator modules that implements an evaluation method and provides metrics for a given architecture. This tool can evaluate all quality attributes as long as there are suitable evaluation algorithms. The two initial attribute evaluators use mathematical methods to measure the goodness of a given deployment, data transmission reliability and communication overhead. Thus, this tool was classified as formal, although the AttributeEvaluator may use different methods.

Table 4. Dimensions evaluated for Acme Simulator

Dimensions	**Acme Simulator behaviour**
Method	Essentially a Monte-Carlo simulation using specifically designed scenarios (behavior model trees) for evaluation.
ADL	Acme Simulator is linked with AcmeStudio and Acme ADL is necessary to model the architecture design before the evaluation.
Qualities	Initially performance and security were evaluated using a Monte Carlo simulation. The approach is general enough to be extended to other quality attributes.
Trade-Off	The trade-off is presented as a table; the comparison is realized manually with the information provided by the tool. The authors plan to support the comparison directly.
Stakeholders	The stakeholders do not participate during the use of this tool.
Knowledge	No knowledge is explicitly preserved to new projects.

Table 5. Dimensions evaluated for ArcheOpterix

Dimensions	**ArcheOpterix behaviour**
Method	Although the tool uses formal modeling to evaluate the attributes, the *AttributeEvaluator* which contains the algorithm can adopt other methods.
ADL	Uses the AADL standard to describe the architecture to be evaluated.
Qualities	All quality attributes can be evaluated as long an algorithm exists. Currently two quality attributes have been evaluated: data transmission reliability and communication overhead.
Trade-Off	The tool uses an architecture optimization module to solve multi-objective optimization problems using evolutionary algorithms.
Stakeholders	The stakeholders do not participate using this tool.
Knowledge	ArcheOpterix does not preserve the knowledge of evaluations to be used in new projects.

5.5. DeSi

DeSi is an environment that supports flexible and tailored specification, manipulation visualization and re-estimation of deployment architectures for large-scale and highly distributed systems. Using this tool it is possible to integrate, evaluate and compare different algorithms improving system availability in terms of feasibility, efficiency and precision.

This tool was implemented in the Eclipse platform. Its architecture is flexible enough to allow exploration to other system characteristics (e.g., security, fault-tolerance, etc.). DeSi was inspired in tools for visualizing system deployment as UML improving support to specifying, visualizing and analyzing different factors that influence the quality of a deployment.

Table 6. Dimensions evaluated for DeSi.

Dimensions	DeSi behaviour
Method	Formal, DeSi uses algorithms to improve system availability
ADL	The data is input directly in DeSi interface and the tool does not use any ADL.
Qualities	Availability. Although depending on the taxonomy, some features could be classified as security and performance. Moreover, the tool allows the use of new algorithms to explore different attributes.
Trade-Off	There is no trade-off function. DeSi simply provides a benchmarking capability to compare the performance of various algorithms.
Stakeholders	Stakeholders participation is not present.
Knowledge	No knowledge is preserved for future evaluations.

5.6. General overview

In the Table 7, we assembled a summary of results for the selected tools. A rapid analysis provides a synthesis of the most relevant dimensions that are required to assess its nature, applicability and usefulness for a specific goal.

As can be seen, although every tool claims to be devoted to architectural assessment their focus can be very distinct. While ArchE, AET and ArcheOpterix goals cover every quality attribute (depending on whether the corresponding models are implemented), AcmeSimulator and DeSi have a specific focus. On the other hand, while ArchE, AET and AcmeSimulator support human guided trade-off analysis, Archepterix and DeSi provide absolute assessment metrics. As such there is no 'best' tool, as it depends on the usage envised by the architect to face a specific problem in a specific context.

6. Conclusions and further work

It is difficult to classify and compare architectural assessment tools that use different methods and techniques. The large amount of features and attributes that can be questioned hampers this process and demonstrate that generic tools that can be tailored to the architect needs are required. The more flexibility and features available the more it is customizable to the specific architect needs.

Using the assessment insight provided by our approach (Table 7) is possible not only improve the selection of the more appropriate tool for a specific context, but also serve as a powerful driver for building an advanced integrated tool to support architectural modeling and analysis as it shows the attributes that such a comprehensive tool should support. The features presented here are essentially: the ability to handle multiple modelling approaches, integration with the industry standard UML, but also specific ADL whenever needed, support of trade-off analysis of multiple competing quality attributes and, of utmost importance, the possibility to reuse knowledge through the reuse and build-up of new architectural solutions.

This is part of our contribution to a major international effort (USA/EU) in developing a tool (AFFIDAVIT) to promote the widespread adoption of sound architectural practices in the software engineering community, consequently, aiming to increase global software quality. Our specific focus is on maintainability addressing methods to assess this attribute in software architectures, and metrics to assess the level of maintenance during the architectural design.

As future work we intend to improve this analysis including with more dimensions (e.g. application domain, flexibility, open-source, etc.) and publish this framework on Web to be used by researchers and practitioners. Furthermore, in the context of our work, an evaluation of maintainability properties assessed by these tools should be done.

Table 7. Overview of tools according to the relevant dimensions

	ArchE	AET	Acme Simulator	ArcheOpterix	DeSi	*Affidavit*
Method	Scenarios	Scenarios	Simulation	Formal	Formal	All
ADL			✓	✓		✓
Q.A.	All	All	Performance Security	All	Availability	All
Trade-off	✓	✓	✓			✓
Stakeholders		✓				✓
Knowledge		✓				✓

References

[1] P. Clements, R. Kazman, and M. Klein, Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley Professional, 2001.

[2] H. Yang and D. M. Ward, Successful evolution of software systems. Artech House, 2003.

[3] R. Pressman and R. Pressman, Software Engineering: A Practitioner's Approach, 6th ed. McGraw-Hill Science/Engineering/Math, 2004.

[4] J. H. Hausmann and R. Heckel, Detection of Conflicting Functional Requirements in a Use Case-Driven Approach - A static analysis technique based on graph transformation, ICSE

2002, pág. 105--115, 2002.

[5] H. Post, C. Sinz, F. Merz, T. Gorges, and T. Kropf, Linking Functional Requirements and Software Verification, in Requirements Engineering, IEEE International Conference on, Los Alamitos, CA, USA, 2009, pages. 295-302.

[6] L. Chung and J. do Prado Leite, On Non-Functional Requirements in Software Engineering», in Conceptual Modeling: Foundations and Applications, vol. 5600, Springer Berlin / Heidelberg, 2009, pages. 363-379-379.

[7] R. Kazman, L. Bass, G. Abowd, and M. Webb, SAAM: A Method for Analyzing the Properties of Software Architectures, 1994.

[8] M. R. Barbacci et al., Steps in an Architecture Tradeoff Analysis Method: Quality Attribute Models and Analysis, Software Engineering Institute, Carnegie Mellon University, pages. 219--230, 1998.

[9] P. C. Clements, Active Reviews for IntermediateDesigns. SEI: Carnegie Mellon University, 2000.

[10] M. Klein, with R. Kazman, Jai Asundi, Quantifying the costs and benefits of architectural decisions, in Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on, 2001, pages. 297-306.

[11] Dolan, Wortmann and Hammer, and , Technische Universiteit Eindhoven, Architecture assessment of information-system families : a practical perspective», Technische Universiteit Eindhoven, 2001.

[12] M. A. Babar, L. Zhu, and R. Jeffery, A Framework for Classifying and Comparing Software Architecture Evaluation, In: Proceedings Australian Software Engineering Conference (ASWEC). vol. 2004, pages. 309--318, 2004.

[13] P. Zhang, H. Muccini, and B. Li, A classification and comparison of model checking software architecture techniques, Journal of Systems and Software, vol. 83, nº. 5, pages. 723-744, May. 2010.

[14] M. Mattsson, H. Grahn, and F. Mårtensson, Software Architecture Evaluation Methods for Performance, Maintainability, Testability, and Portability, presented at the Second International Conference on the Quality of Software Architectures (QoSA 2006), 2006.

[15] V. Vallieswaran and B. Menezes, ArchKriti : A Software Architecture Based Design and Evaluation Tool Suite, in Information Technology, 2007. ITNG '07. Fourth International Conference on, 2007, pages. 701-706.

[16] F. Bachmann and L. Bass, Introduction to the attribute driven design method, in Proceedings of the 23rd International Conference on Software Engineering, Washington, DC, USA, 2001, pages. 745–746.

[17] «Affidavit Project», Feb-2011. [Online]. Available: http://affidavit.dei.uc.pt/.

[18] A. J. Department and A. Jansen, Software Architecture as a Set of Architectural Design Decisions», In Proceedings Of WICSA 5, pages. 109--120, 2005.

[19] T. Al-Naeem, I. Gorton, M. A. Babar, F. Rabhi, and B. Benatallah, A quality-driven systematic approach for architecting distributed software applications, in Proceedings of the 27th international conference on Software engineering, St. Louis, MO, USA, 2005, pages. 244-253, 2005.

[20] E. Niemelä, Strategies of Product Family Architecture Development, in Software Product Lines, 2005, pages. 186-197.

[21] K. Henningsson and C. Wohlin, Understanding the Relations between Software Quality Attributes - A Survey Approach, presented at the 12th International Conference for Software Quality, Ottawa - Canada, 2002.

[22] M. Barbacci, M. Klein, T. Longstaff, and C. Weinstock, Quality Attributes: Technical Report. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA., 1995.

[23] J. Bosch, Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach. Addison-Wesley Professional, 2000.

[24] Y. Deng, S. Lu, and M. Evangelist, A Formal Approach for Architectural Modeling and Prototyping of Distributed Real-Time Systems, presented at the Thirtieth Hawaii International Conference Digital on System Sciences, 1997.

[25] C. Rosso, Continuous evolution through software architecture evaluation: a case study, Journal of Software Maintenance and Evolution: Research and Practice, vol. 18, nº. 5, 2006.

[26] F. Aquilani, S. Balsamo, and P. Inverardi, Performance analysis at the software architectural design level, Performance Evaluation, vol. 45, nº. 2-3, pages. 147-178, Jul. 2001.

[27] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann, «Specification and analysis of system architecture using Rapide, IEEE Transactions on Software Engineering, vol. 21, nº. 4, pages. 336-354, 1995.

[28] D. Garlan, R. Allen, and J. Ockerbloom, Exploiting Style in Architectural Design Environments, 1994.

[29] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik, Abstractions for Software Architecture and Tools to Support Them, IEEE Transactions On Software Engineering, vol. 21, pages. 314--335, 1995.

[30] D. Garlan, R. Monroe, and D. Wile, Acme: an architecture description interchange language, in CASCON First Decade High Impact Papers, New York, NY, USA, 2010, pages. 159–173.

[31] N. Medvidovic and R. N. Taylor, A Classification and Comparison Framework for Software Architecture Description Languages, IEEE Trans. Softw. Eng., vol. 26, nº. 1, pages. 70-93, 2000.

[32] D. Garlan, J. Ockerbloom, and D. Wile, Towards an ADL Toolkit, presented at the EDCS Architecture and Generation Cluster, 2008.

[33] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. Ali Babar, A comparative study of architecture knowledge management tools, Journal of Systems and Software, vol. 83, nº. 3, pages. 352-370, Mar. 2010.

[34] A. Diaz-Pace, H. Kim, L. Bass, P. Bianco, and F. Bachmann, Integrating Quality-Attribute Reasoning Frameworks in the ArchE Design Assistant, Proceedings of the 4th International Conference on Quality of Software-Architectures: Models and Architectures, pages. 171–188, 2008.

[35] Steffan Thiel, Andreas Hein and Heiner Engelhardt and Robert Bosch, Tool Support for Scenario-Based Architecture Evaluation, presented at the 25 th International Conference on Software Engineering, 2003.

[36] Bradley Schmerl, Shawn Butler and David Garlan, Architecture-based Simulation for Security and Performance, 2006.

[37] A. Aleti, S. Bjornander, L. Grunske, and I. Meedeniya, ArcheOpterix: An extendable tool for architecture optimization of AADL models, in Model-Based Methodologies for Pervasive and Embedded Software, International Workshop on, Los Alamitos, CA, USA, 2009, vol. 0, pages. 61-71.

[38] Marija Mikic-rakic, Sam Malek and Nels Beckman and Nenad Medvidovic, A Tailorable Environment for Assessing the Quality of Deployment Architectures in Highly Distributed Settings, presented at the Working Conf. on Component Deployment, 2004, pages. 1-17.