TOWARDS AN AUTOMATED EVALUATION PROCESS FOR SOFTWARE ARCHITECTURES

R. Bashroush, I. Spence, P. Kilpatrick, T.J. Brown
Queen's University Belfast
School of Computer Science
18 Malone Road, Belfast BT7 1NN, UK
{r.bashroush, i.spence, p.kilpatrick, tj.brown}@qub.ac.uk

Abstract

Optimizing and editing enterprise software systems, after the implementation process has started, is widely recognized to be an expensive process. This has led to increasing emphasis on locating mistakes within software systems at the design stage, to help minimize development costs. There is increasing interest in the field of architecture evaluation techniques that can identify problems at the design stage, either within complete, or partially complete architectures. Most current techniques rely on manual review-based evaluation methods that require advanced skills from architects and evaluators. We are currently considering what a formal Architecture Description Language (ADL) can contribute to the process of architecture evaluation and validation. Our investigation is considering the inter-relationships between the activities performed during the architecture evaluation process, the characteristics an ADL should possess to support these activities, and the tools needed to provide convenient access to, and presentation of architectural information.

Key Words

Software Architecture, Evaluation, Tools, ATAM

1. Introduction

Current software architecture evaluation techniques do not depend on any specific form of architecture description. Although their lack of dependence on any particular formal description framework makes them widely applicable, the lack of standardization can introduce problems. There may be gaps between the information contained in an architecture description and the information required by the evaluation process. This can lead to time-consuming effort being needed, to extract information from the architecture description in an appropriate form, or to provide additional information in order to facilitate the evaluation process. If, on the other hand, the architecture is described using an architecture description language, it should be possible to introduce a degree of standardization into the description process, which is aligned with the requirements of the architecture assessment process. To achieve this we must design the ADL to take account of the evaluation process, so that when an architecture is fully described in the language, we can be confident of having all the information we expect to need at the evaluation phase.
Our research targets this issue and studies the properties an ADL should have in order to bring the architecture description and the evaluation techniques together, in an attempt to alleviate the human effort in the overall process of evaluation.

In the course of this work we are using an ADL called ADLARS [1] that is being developed within our group, and was targeted originally at real-time software families. The reason for choosing ADLARS is that with our own ADL we can modify the language to incorporate characteristics suggested by our investigation of the evaluation process. As for the evaluation technique, we have adopted the "Architecture Tradeoff Analysis Method" (ATAM)[2] a technique that was developed at CMU within one of the leading teams in the domain of software evaluation, and is considered a good example of a formal evaluation process.

In this paper we report on work-in-progress on combining ATAM with ADLARS, and at identifying the most suitable tool support to provide partial automation for architecture evaluation in the context of software product lines. In what follows, we will give a brief description of the general steps of ATAM, highlighting the steps that could make use of a formal description language like ADLARS, given the necessary set of tools. Section 3 presents our proposals for the properties an architecture description language should have to support the evaluation process. Section 4 discusses the general need for tool development and summarizes a recommended set of tools that could be of value in the context of evaluation showing a direct correlation with ATAM. Section 5 contains the conclusion and the anticipated future work.

2. Information Flows within the ATAM Evaluation Process

Product-line architectures are designed to serve as the basis for a whole family of intended products. They must also satisfy a set of quality attributes and stakeholder requirements. They must be described in a way that satisfies a variety of requirements. Implementers will need to extract enough technical detail to facilitate implementation, while other stakeholders may need a broader overview of the architecture. After the architecture development stage, the architecture description forms the principal input to the evaluation process. Within this process the architecture will be evaluated (using ATAM for example), against the set of required attributes, with results being feed back into the process (figure 1). We can think of the process as a flow of information from one entity to the other. We used the term Information Bottle-neck to describe points of large-scale information flow within the process. These are points where inadequacies in the information due to human failure could introduce crucial time delay to the overall process. Identifying these potential bottle-necks in the overall process can help us to determine where best to target tool support.

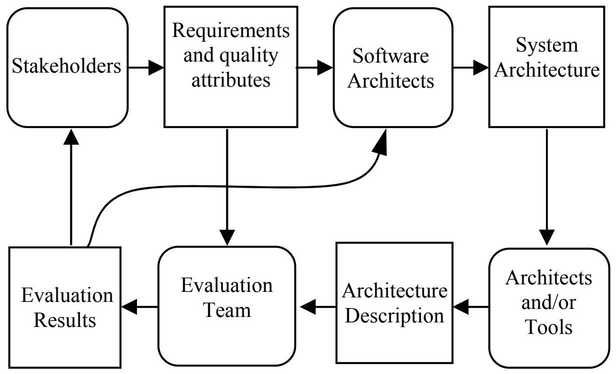

Figure 2. The Software Architecture Development and Evaluation Lifecycle

There has been a wide variety of approaches for evaluating and benchmarking the performance of software designs; however, they all share the same aim which is predicting problems in design prior to implementation. We have chosen ATAM [2] as being a widely accepted evaluation process that can constitute a good test-bed for measuring the contribution of a formal ADL to the evaluation process. ATAM emphasises the interaction between the quality attributes in a system, and hence the term "tradeoff" in the title. ATAM comprises nine steps divided into four groups as summarized in [3, pp. 44]: (for more details on ATAM read [2][3][4])

A. Presentation
  1. Present ATAM
  2. Present the business drivers
  3. Present the architecture

B. Investigation and Analysis
  4. Identify the architectural approaches
  5. Generate the quality attribute utility tree
  6. Analyze the architectural approaches

C. Testing
  7. Brainstorm and prioritise scenarios
  8. Analyze the architecture approaches

D. Reporting
  9. Present the results

The first phase in our work was to study the steps in ATAM and locate the potential bottle-necks in the process. As is shown above, the first three steps are the presentation of the architecture and business drivers by architects and stakeholders to the evaluation team who will analyze the data in steps 4, 5 and 6. Here we notice data flow from the architecture team to the evaluation team. This flow is generally not governed by any standard protocol or data format. This could cause a problem if the first team (architects) does not pass all the required information in the appropriate format (as the evaluation team might not contain domain experts) to the second team (evaluators). This is stated in [3, pp. 105], confirming the importance of the clarity and completeness of the documentation of the architecture to the evaluation process: this constitutes the first potential bottle-neck.

Applying scenarios to architectures is the core of most evaluation processes, and ATAM is no different. In ATAM's testing phase, use cases are employed to assess whether the architecture meets its non-functional requirements. As human interaction constitutes the main part of this step, this could add a significant delay to the overall process, especially if there is a large number of scenarios, and hence the second potential bottle-neck.

We have looked at possible ways in which a formal language with an appropriate set of tools could contribute to the above process, especially with reference to alleviating the bottle-necks. Here we note that some work has already been done to this end, including UCM [5][6] for capturing scenarios, and ABAS [7] (and lately the work on "tactics"[14]) that integrates quality attributes within standard architectural styles [8]. However, our research is to investigate the properties of a formal ADL and its corresponding set of tools that could facilitate the usage and integration of these different concepts (ABAS, UCM, etc.) in one development environment. Our first task is to find the properties an ADL should have to be capable of capturing all the necessary information about a candidate architecture (section 3). The second is to specify a suitable set of tools that would extract the information from the architecture definition and output it in a desired format (section 4).

3. ADL Properties to Support Evaluation

From analysis of the ATAM process, we identified the following characteristics [15] of an ADL as being desirable in the context of the evaluation process:

1. The ability to contain all the necessary information that would enable the user to see the different architectural views of a candidate system (Functional view, Concurrent view, Code view, etc.) i.e. to be capable of distinguishing among the different views.

2. A facility for adding textual descriptions to various architecture components and tasks that can serve as notes or comments that would be helpful for documentation purposes.

3. The ability to capture information about the architectural styles [8] (and sub-styles) used and correlating styles with attributes. Important work has been conducted in this area, and one of the important outcomes is the Attribute-Based Architectural Styles ABAS [7].

4. The capability of allowing the construction and evaluation of Use Cases on the architecture described. This is not a new idea, as the usage of Use Case Maps UCM [5] for simulating scenarios has attracted much research interest [6][9].

The first two properties above support architecture presentation and documentation, corresponding to the first three steps in the ATAM process. The third property supports analysis of the architecture approaches used and their corresponding quality attributes, and this would help with the fourth, fifth and sixth steps in ATAM: Investigation and Analysis. The fourth property in our list will contribute to the automation of scenario analysis, although automation may not apply to all possible types of scenarios.

Now considering these requirements in the context of ADLARS, the language as described in [1] already covers the first two properties. ADLARS views software architectures to be existing in a three dimensional space: concurrency, structure and behaviour. Concurrency is conveyed in Tasks, structure is described by Components and behaviour is captured by Interaction Themes. This satisfies the first property. Task and Component definitions both have a field called Description that allows the addition of textual notes. This serves the second property in our list. For more information about ADLARS please read [1].

Current work is being conducted to enable ADLARS to support the notion of Attribute Based Architecture Styles (ABAS) instead of regular architectural styles [8]. This will serve the third property of the list. And finally, a UCM extension tool is in the process of construction. The tool would help in compiling UCM defined scenarios over ADLARS described architectures. Tools are discussed in more detail in the next section.

4. Recommended Set of Tools

The role of tools in the cycle of any software engineering process has always been of great importance. The relationships between tools, notations and process activities are often succinctly portrayed in terms of the so-called Triangle of Success [10] shown in figure 2.

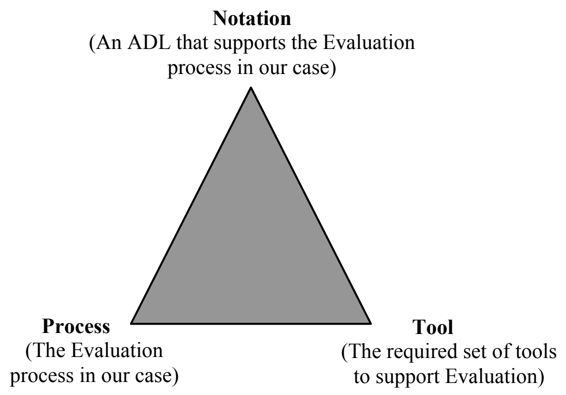
Notation
(An ADL that supports the Evaluation process in our case)
Process
(The Evaluation process in our case)
Tool
(The required set of tools to support Evaluation)

Figure 3. The Triangle of Success [10]

As suggested by Quatrani [10], for successful software engineering, you need all three facets – a notation, a process, and a tool. "*You can learn a notation, but if you don't know how to use it (process), you will probably fail. You may have a great process, but if you can't communicate the process (notation), you will probably fail. And lastly, if you cannot document the artifacts of your work (tool), you will probably fail*" [10]. Moreover, in SPLC2002 [11] it was pointed out that there is a need for tools and that not much has been done in tools development.

In the context of software architectures, the ability to retrieve information from an architecture definition for documentation or presentation purposes is as important as the ability to build the architecture. Imagine someone writing poetry in a language that no one can read! Extracting information from an architecture definition, especially when considering enterprise software systems that constitute thousands of lines of code, would be a very time consuming process when conducted by a human being, and here we sense the importance of the tools in the software development life cycle.

We have spent some time considering the possible tools that could be useful in our work and we arrived at the set of proposed tools (the ADLARS Development Studio) that is summarized in figure 4.

Each tool would manage the extraction and presentation of part of the ADLARS architecture description throughout the development and evaluation processes. Where appropriate, the tools would offer alternative views of aspects of the architecture. The tools, combined with appropriate ADL language features can ensure that all necessary information is contained within the architecture description, and can be retrieved in the best way to facilitate the evaluation process.

A Table showing where each tool of the development studio could contribute to the ATAM process is presented at the end.

5. Conclusion and Future Work

Concepts and ideas [5][7][12] have been developed that could help with different steps of the architecture evaluation processes, a good overview of which is given in [13]. Our research is concerned with studying the different evaluation techniques to see how these techniques could benefit from a formal language that would capture widely-accepted concepts and ideas such as those of UCM [5] and ABAS [7], in an attempt to drive the evaluation process towards automation.

Architecture evaluation teams do not always comprise domain experts, and the introduction of the team to the domain properties, common problems and design patterns can be a time consuming process. This is what motivates the search for tools that would assist the evaluation team. Our work in this field is still in its early stages and much more remains to be done. This paper proposed a marriage of ATAM and ADLARS; however, the reasoning followed here could be applied to any ADL-Evaluation method combination.

Currently several case studies on the application of ADLARS are nearing completion [16]. These will form the basis for applying the ATAM strategy in the context of an ADLARS-described architecture and allow more detailed definition of the tools of the ADLARS studio. Also, ADLARS itself continues to evolve and current work is focused on providing support for attribute based architectural styles. We sense a great potential in ABAS, especially in the evaluation domain, and so capturing ABAS in our ADL and investigating the proper way of presenting it will be high on our list of future priorities.

References

[1] T.J. Brown, I. Spence, P. Kilpatrick. ADLARS: A Relational Architecture Description Language for Software Families. *Proc. of the 5th International Workshop on Product Family Engineering*, Siena, Italy, 2003.

[2] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere. The architecture tradeoff analysis method. *In Proceedings of the 4th IEEE International Conf. on Engineering of Complex Systems*, CA, 1998, 68-78.

[3] P. Clements, R. Kazman, M. Klein. Evaluating Software Architecture: Methods and Case Studies. SEI series in software engineering. Addison-Wesley 2002.

[4] R. Kazman, M. Klein, and P. Clements, ATAM: Method for architecture evaluation. *Technical report, CMU/SEI-2000-TR-004, 2000.*

[5] R.J.A. Buhr, R.S. Casselman, *Use Case Maps for object-oriented systems* (Prentice Hall, 1996).

[6] H. de Bruin, H. van Vliet. *Scenario Based Generation and Evaluation of Software Architecture, Proc. 3rd International Conference*, Erfurt, 2001, 128-139.

[7] M. Klein, R. Kazman, L. Bass, J. Carriere, M. Barbacci, H. Lipson. Attribute-Based Architecture Styles. *Proc.of the First Working IFIP Conference on Software Architecture*, TX, 1999, 225-243.

[8] M. Shaw, D. Garlan. *Software Architecture: Perspectives on an emerging discipline* (Prentice Hall, 1996).

[9] D. Petriu, M. Woodside. Software Performance Models from System Scenarios in Use Case Maps. *Proc. 12th International Conference, Performance TOOLS 2002*, London, 2002.

[10] T. Quatrani. *Visual Modeling with Rational Rose and UML* (Addison-Wesley 1998).

[11] Software Product Lines, Second International Conference, SPLC 2, San Diego, CA, USA. August 19-22, 2002, Proceedings. LNCS 2379 Springer 2002.

[12] K. Kang, S. Cohen, J. Hess, W. Novak, A.S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. *Technical Report CMU/SEI-90-TR-21, ESD-90-TR-222.* November 1990.

[13] M. Ionita1, D. Hammer, H. Obbink. *Scenario-Based Software Architecture Evaluation Methods: An Overview. Proc. ICSE 2002 Workshop Methods and Techniques for Software Architecture Review and Assessment*, Florida, 2002.

[14] F. Bechmann, L. Bass, M. Klein, *Deriving Architectural Tactics: A step Toward Methodical Architectural Design. Technical Report CMU/SEI-2003-TR-004.*

[15] R. Bashroush: *The Contribution of Architecture Description Languages to the Evaluation of Software Architectures.* Submitted to the Doctoral Symposium at the 26th international conf. on Software Engineering, Scotland, UK, 2004.

[16] R. Bashroush, I. Spence, P. Kilpatrick and T.J. Brown. A Real-time Network Emulator: ADLARS Case Study. *Proc. of the 3rd Asia Pacific International Symposium*, Istanbul, 2004.

ATAM Step	**ADLARS conjugate**
1	-
2	*Feature Model Tree* [12][1]
3	*ArchView*- Visual tool capable of extracting different architectural views (Functional, Concurrency, Code etc.) from the architecture description and displaying them graphically
4	*StyleView*- extracts information from the *Arrangements* section in the *Component* that specifies the style/pattern used
5	Attribute-Based Architectural Styles ABAS[2]
6	Attribute-Based Architectural Styles ABAS[2]
7	*UCM extension*- Capturing scenarios in the form of UCM and mapping them to ADLARS definition files
8 [3]	Using *UCM extension* tool to run UCMs over ABASs
9	-

[1] The Feature Model Tree [12] captures the business drivers in the form of features. It is the first step conducted before building ADLARS Tasks and Components, please refer to [1] for more details. Here its worth mentioning that currently not all business goals can be captured by the Feature Model Tree.

[2] To be included in the next version of ADLARS

[3] Even though the 6th and the 8th steps in ATAM are the same, their ADLAR's conjugates differ as step 6 is in the analysis phase and step 8 is in the testing phase

Table 1. Contribution of *ADLARS development studio* to the ATAM process

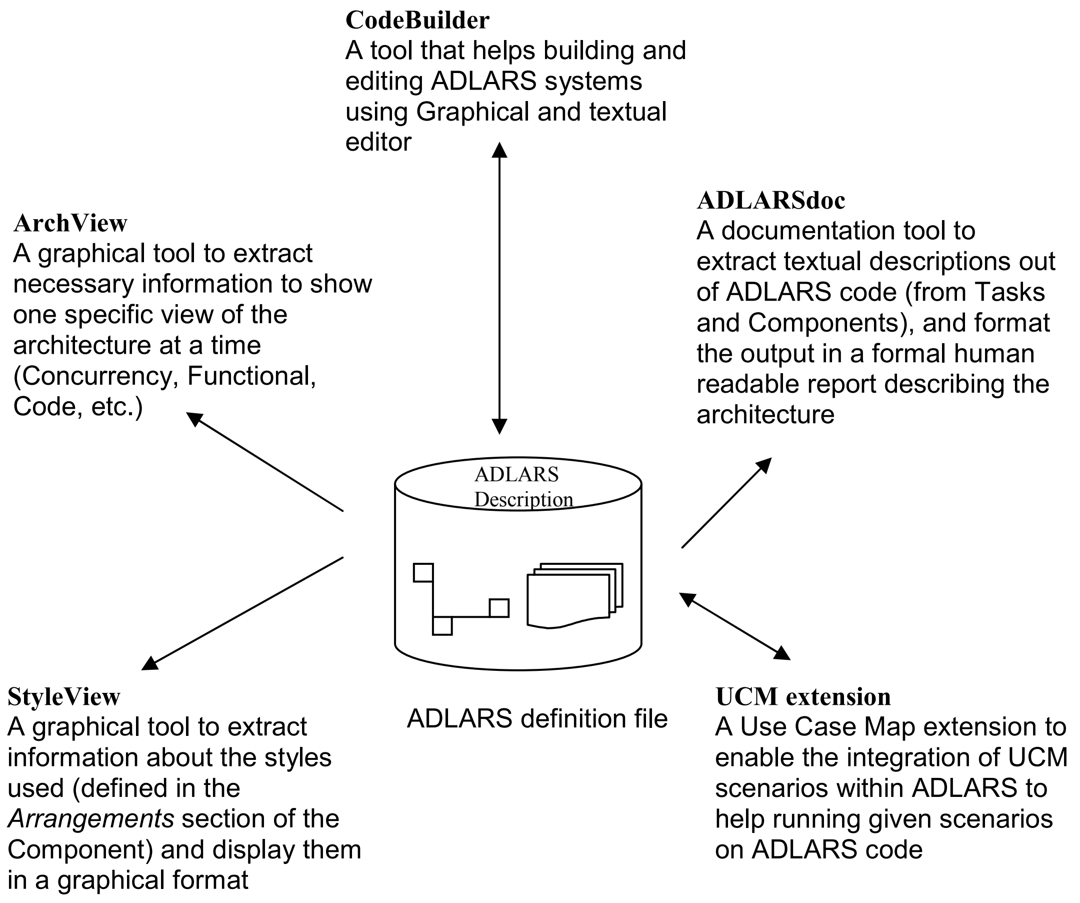
CodeBuilder
A tool that helps building and editing ADLARS systems using Graphical and textual editor
ADLARSdoc
A documentation tool to extract textual descriptions out of ADLARS code (from Tasks and Components), and format the output in a formal human readable report describing the architecture
ArchView
A graphical tool to extract necessary information to show one specific view of the architecture at a time (Concurrency, Functional, Code, etc.)
ADLARS Description
ADLARS definition file
StyleView
A graphical tool to extract information about the styles used (defined in the *Arrangements* section of the Component) and display them in a graphical format
UCM extension
A Use Case Map extension to enable the integration of UCM scenarios within ADLARS to help running given scenarios on ADLARS code

Figure 4. ADLARS Development Studio [15]