Multi-View Description of Software Architectures

Valérie Issarny, Titos Saridakis, Apostolos Zarras

INRIA - IRISA
Campus de Beaulieu, 35042 Rennes Cédex, FRANCE
email: {issarny,saridaki,zarras}@irisa.fr

Abstract

The specification of a software architecture using different ADLs allows system designers to carry out a number of complementary analyses. In this position paper, we go one step further in this direction by advocating the need for specifying distinct views of a software architecture, each characterizing a specific type of properties (i.e. functional, interaction, and quality properties). Multi-view description of a software architecture raises the issue of combining a set of architectural views so as to derive the resulting overall architecture. We propose some hints on how this can be handled.

1 Introduction

It is now recognized that the construction of complex software systems can greatly benefit from the software architecture paradigm [9]. The software architecture of a system describes the system's gross organization using an ADL, which provides notations for the abstract specification of its architectural elements. Practically, the construction of a system from its software architecture requires taking into account the following properties of components and connectors: (*i*) the functionality offered by a component must match the ones expected by the components connected to it; (*ii*) the connectors used for handling interactions among components must provide communication protocols that conform to those expected by components; (*iii*) the overall architecture must provide the desired *quality* in terms of criteria as diverse as dependability, efficiency, scalability, security, timeliness, and usability. In this position paper, we propose to use multi-view architectural descriptions to cope with the above types of properties. The next section discusses the different views of a software architecture. Section 3 then addresses the issue of architectural consistency that is the validity of an overall software architecture with regards to its different descriptions. We conclude in Section 4.

ISAW3 Orlando Florida USA
1998 1-58113-081-3/98/11

2 Architectural Views

The construction of a software system requires to take into account the behavior of components with respect to their functional, interaction, and quality properties. Based on this premise, we claim that the software architecture of a system should be subdivided into a set of complementary architectural views where each view corresponds to one type of properties. A pragmatic justification is that some of the aforementioned properties are already handled by existing ADLs (e.g. see [5] for an overview of ADLs), which may be conveniently exploited through the ACME framework [2]. An additional justification relates to the resulting benefit from the standpoint of design reuse and software evolution. For illustration, let us consider the example of a basic Distributed File System (DFS) structure. Figure 1 depicts different views of the DFS software architecture where boxes denote components, ellipses denote connectors, and directed arrows denote flows of control. Furthermore, white boxes denote functional components (i.e. components implementing some algorithmic aspect of the system), and grey boxes denote components enforcing some quality property. Base constituents of the DFS are the client and the file service functional components. Notice that although we depict a single client component, there may be several instances of such. Specifically, the views that we consider for illustration are: *a*) the *functional view*, which gives the operations that may be called by any client of the file service, together with the file service operations, e.g. read and write; *b*) the *interaction view*, which gives the interaction protocol between the client and the service, e.g. a client-server interaction; *c*) two *efficiency views*, which give the DFS structure aimed at increasing its efficiency by employing respectively a prefetching technique and a distributed implementation of the service coupled with load balancing; and *d*) the *dependability view*, which gives the DFS structure with respect to the fault tolerant mechanisms that are used, e.g. replication of the file service whose management relies on some broadcast protocol.

In parallel to the multi-view description of the DFS

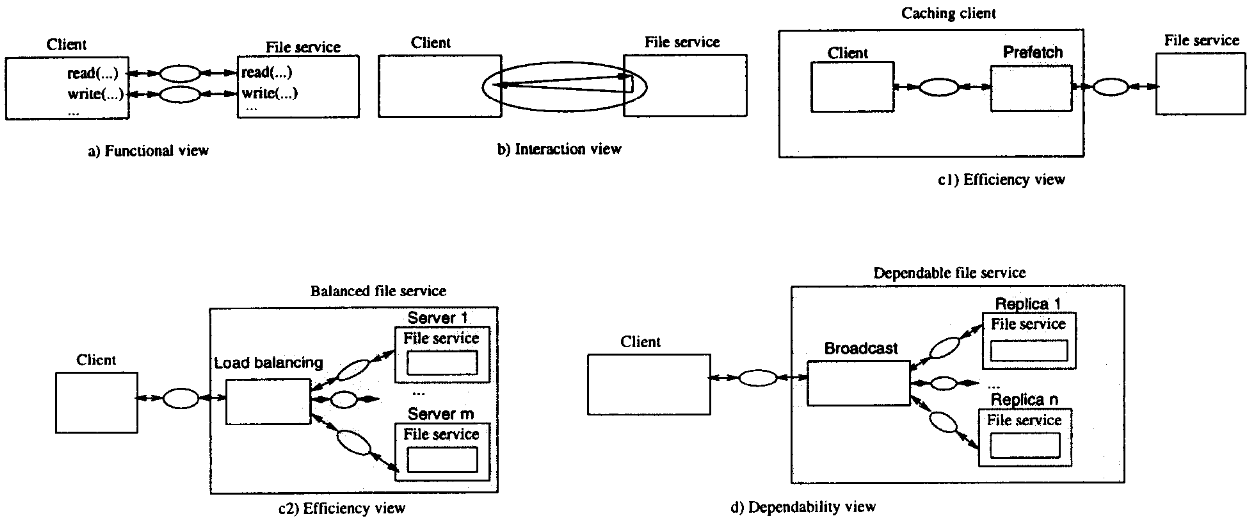

Figure 1: Architectural views of the DFS

architecture, let us consider a single view aggregating all the aforementioned architectural features, as depicted in Figure 2. Considering design reuse, the multi-view description allows developers to exploit the design decisions made for the specific architecture for other application domains; only the functional view is tightly coupled with the DFS design. All the other design aspects are at least eligible for any client-server architectural style provided the adequate substitution of functional components (white boxes). From the perspective of software evolution, the multi-view architecture facilitates the revision of each software property, independently of the others.

So far, we have given a rough informal picture of architectural view descriptions. A question that raises is to identify which ADL should be used for formal descriptions. As already identified in the software architecture community, existing ADLs together provide distinct useful capabilities. Hence, each architectural view may be described from different perspectives using different ADLs, so as to carry out complementary analyses. However, a number of important issues remain open, related to the composition of a set of architectural views so as to produce the overall system architecture (e.g. producing the architecture of Figure 2 from those depicted in Figure 1).

3 Combining Views

The combination of architectural views raises two complementary issues: (*i*) architectural consistency, i.e. verifying whether the composition of the views' behavior produces a correct behavior, and (*ii*) architectural structure, i.e. producing the structure of the overall architecture from the structure of its composing views.

Architectural consistency: One way to handle architectural consistency is to derive the overall architecture first and then to carry out adequate analyses. However, we would like *compositionality of views* so as to exploit the results of the analyses performed on individual views. This issue is similar to the horizontal composition of refinement patterns discussed in [6], which uses a syntactic criterion so as to avoid a case-by-case proof of correctness. Such a solution cannot be undertaken for the composition of views. For illustration, let us consider the quality views of the DFS example: depending on replication management used in the dependability view, this may impact negatively on the system's performance and hence lower the actual benefit of the efficiency views. Another example is the combination of fault tolerance with security concerns where the former requires replication while handling the latter implies minimizing replication. Let us examine view compositionality in more detail, by considering the various pairs of distinct types of views:

- **Functional-Interaction**: The only aspect to be considered for architectural consistency is to ensure that the synchronization among functional operations is consistent with the synchronization achieved by the underlying interaction protocol. Compositionality of functional and interaction views may then be handled in a way similar to the treatment of port-role compatibility in Wright [1]. The synchronization protocols associated to component operations are declared in the functional view and are checked for compatibility with the protocols declared in the corresponding components of the interaction view.

- **Interaction-Quality**: The combination of interaction with quality views requires to ensure that the interaction properties achieved among functional components remain unchanged. Hence, we may reason on compositionality of interaction with quality views in a way similar to the treatment of composition between interaction and functional views. A quality view must declare the synchronization protocols enforced for its composing functional components, which are checked for compatibility with the interaction protocols of the corresponding components in the interaction view.

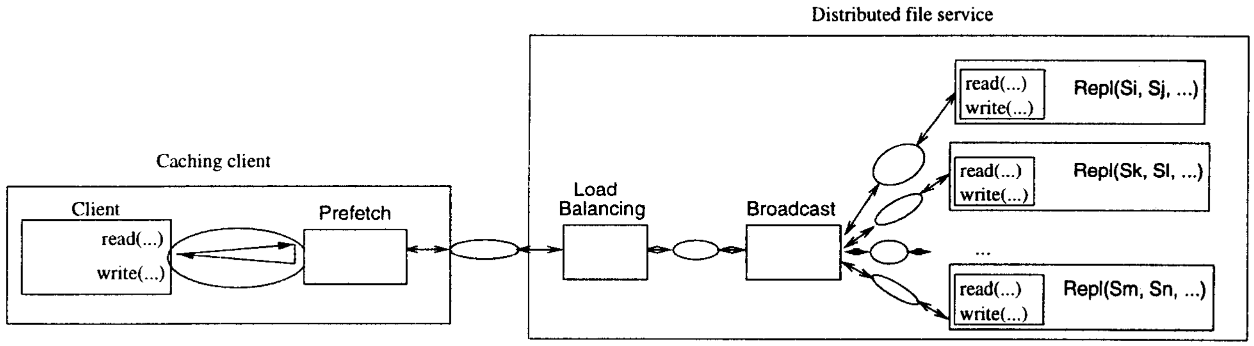

Figure 2: The complete DFS

- **Quality-Quality**: As previously suggested, reasoning about the compositionality of quality views is the most difficult part. We must ensure that for any two views $\mathcal{V}_1$ and $\mathcal{V}_2$ that are respectively targeted for quality properties of types $\mathcal{T}_1$ and $\mathcal{T}_2$, the quality property of type $\mathcal{T}_2$ (resp. $\mathcal{T}_1$) of the view $\mathcal{V}_1$ (resp. $\mathcal{V}_2$) is consistent with the quality property of $\mathcal{V}_2$ (resp. $\mathcal{V}_1$). In the framework of the Aster project[1], we have been working on the abstract specification of architectural views enforcing some quality properties (i.e. dependability, security, and concurrency control) using temporal first-order logic so as to allow the analysis of architectures with respect to provided quality properties as well as quality view refinement (e.g. see [4]). However, these properties are addressed independently and we are currently examining a solution to the compositionality of associated views. A direct solution consists in requiring to associate to each quality view, its properties with respect to the other types of qualities. Reasoning about compositionality of quality views then amounts to verify consistency among the quality properties relating to the same quality type (i.e. the conjunction of the quality properties must not evaluate to false). The practicality of the above approach still needs to be devised.
- **Functional-Quality**: A quality view may make assumptions on the quality properties of its composing functional components. Thus, a functional view may be composed with a quality view if the quality behavior of components in the functional view is consistent with the one of the corresponding components in the quality view. One way to handle this aspect is to have the specification of the quality properties provided by the functional components in the functional and quality views. These properties must then be checked for consistency at composition time in a way similar to the composition of quality views.

Although the treatment of view compositionality is not completely solved and still needs to be formally defined, we are confident that it can be handled in a rigorous way and even supported through CASE tools including the ones used by the Wright and Aster projects.

[1]http://www.irisa.fr/solidor/work/aster.html.

Architectural structure: In addition to the view compositionality issue, there is the issue of deriving the overall architecture of a system from its composing views. The main objective here is to minimize the developer interaction by providing the adequate CASE tool. We are developing a special instance of such a tool [10], which produces an architecture from a set of quality views and a functional view, for a specific interaction view (i.e. the view associated to the CORBA ORB). Let us notice that we expect existing solutions to the integration of various interaction protocols (e.g. UniCon [8]) to provide the adequate basis for extending the tool to the treatment of various interaction views.

4 Conclusions

We have advocated the need for multi-view description of software architectures, and discussed the issue of composing architectural views so as to derive the overall architecture of a software system. Although results regarding the composition of architectural views are still at a preliminary stage, we are confident that a practical solution can be provided together with supporting CASE tools. To our knowledge, multi-view description of a software architecture has only been addressed from the perspective of exploiting the distinct useful capabilities of existing ADLs (e.g. see [3]). Our work is complementary in that it addresses the combination of architectural descriptions characterizing different behaviors. Our notion of multi-view architecture may be considered as defining a set of architectural styles for an architecture. This approach has in particular been suggested in [7] to characterize the various aspects of product line architectures. However, the author does not examine this issue in detail, he only points out these potential benefits of architectural styles.

References

[1] R. Allen and D. Garlan. A Formal Basis for Architectural Connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, July 1997.

[2] D. Garlan, R. Monroe, and D. Wile. ACME: An Architecture Interchange Language. In *Procee-*

dings of the CASCON 97, pages 169–183, November 1997. http://www.cs.cmu.edu/afs/cs/project/able/-www/papers_bib.html.

[3] D. Garlan and Z. Wang. A Case Study in Software Architecture Interchange. Technical report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, USA, 1998. http://www.cs.cmu.edu/afs/cs/project/able/-www/papers_bib.html.

[4] V. Issarny, C. Bidan, and T. Saridakis. Characterizing Coordination Architectures According to Their Non-Functional Execution Properties. In *Proceedings of the 31st Hawaii International Conference on System Science*, pages 275–283, January 1998. http://www.irisa.fr/solidor/work/aster.html.

[5] N. Medvidovic and R. N. Taylor. A Framework for Classifying and Comparing Architecture Description Danguages. In *Proceedings of the 5th ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 60–76, September 1997.

[6] M. Moriconi, X. Qian, and R. A. Riemenschneider. Correct Architecture Refinement. *IEEE Transactions on Software Engineering*, 21(4):356–372, April 1995.

[7] D. E. Perry. Generic Architecture Description. In *Proceedings of the International Workshop on the Principles of Software Evolution*, 1998.

[8] M. Shaw, R. DeLine, D. Kelin, T. Ross, D. Young, and G. Zelesnik. Abstraction for Software Architectures and Tools to Support Them. *IEEE Transaction on Software Engineering*, 21(4):314–335, April 1995.

[9] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline.* Prentice Hall, 1996.

[10] A. Zarras and V. Issarny. A Framework for Systematic Synthesis of Transactional Middleware. In *Proceedings of Middleware98, the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, September 1998. To appear. http://www.irisa.fr/solidor/work/aster.html.