1 MULTIPLE VIEWS IN SOFTWARE ARCHITECTURE: CONSISTENCY AND CONFORMANCE

POSITION PAPER

D. Le Métayer*
and M. Périn*

MULTIPLE VIEWS: SIGNIFICANCE AND PROBLEMS

The study of software architectures has grown as an autonomous discipline recently. A widely accepted, and very general, characterisation of software architectures can be sketched as follows [Shaw and Garlan, 1995]: "Software architecture is the level of software design that addresses the overall structure and properties of software systems". The success of this trend of work is based on two major assumptions:

- Much benefit has to be gained from the description of software systems at a high level of abstraction: pay-offs are expected in terms of quality of design, maintenance, analysis, verification, testing, communication between developers, etc.

- Software architectures involve specific concepts which require specific languages, techniques and tools.

*Irisa/Inria, Campus de Beaulieu, 35042 Rennes, France. Email: lemetayer/mperin@irisa.fr

The first assumption is hardly arguable in principle even if better tools and architecture development environments are still needed to really assess the practical significance of the whole approach.

The second assumption is sometimes questioned outside the emerging "software architecture community" because a number of issues related to software architectures have been studied in different contexts in the past. Abstract (and even finite state) models of real systems have been used for a long time and the notions of abstract interpretation [Cousot and Cousot, 1979] and property preserving abstraction [Loiseaux et al., 1995] have been formalised and studied extensively for various classes of programming languages (concurrent, sequential, functional, logic, etc.). Furthermore, as far as formalisms are concerned, process calculi such as CSP or the π-calculus provide concurrency and synchronisation features that can also be used to specify the communication protocols between components in an architecture [Allen and Garlan, 1994, Radestock and Eisenbach, 1996]; sophisticated facilities for modularity are offered by functional languages of the ML family (SML, CAML, etc.) [MacQueen, 1985]; also synchronous languages provide formal semantics of box and line drawings used for describing real-time systems [Benveniste and Berry, 1991]. So it is legitimate at this stage to put the specificity of software architectures under scrutiny and discuss whether it justifies the development of a new research area.

First, as far as abstract interpretation is concerned, it should be noticed that it has mainly been used in a bottom-up rather than a top-down fashion so far. In other words, it has mostly been concerned with (a posteriori) analysis rather than design and refinement. So one interesting issue in the context of software architecture would be the application of the abstract interpretation framework to the refinement of global properties. This would lead, strictly speaking, to a "concrete interpretation" process.

Second, and more importantly, we believe that the doubts mentioned above have been fuelled by the fact that the range of works published in this area so far does not reflect the generality of the approach. It turns out that most of them are concerned with communication and synchronisation which is just one of the aspects of a software architecture. In particular, non functional properties such as security, performance, reliability and extensibility have not received enough attention. But the study of each of these properties may require a different kind of decomposition of the software. For this reason, we believe that the notion of "view" is crucial and deserves more consideration in the context of software architectures. A number of variants of the notion of view have been proposed in different areas of computer science:

- In development methods: for example RM-ODP descriptions [Bourgois et al., 1998] include five viewpoints (enterprise, information, computational, engineering and technology); [Finkelstein et al., 1992] proposes a framework supporting the definition and use of multiple viewpoints in system development; UML also promotes the use of multiple views (as collections of diagrams such as static structure, statechart, component and deployment diagrams) to describe complex systems.

- In programming languages: Aspect Oriented Programming [Kiczales, 1996] puts forward a technique for the development of programs as collections of aspects (such as synchronisation, communication, failure handling, memory allocation) which isolate important design decisions that would otherwise be scattered throughout the code.

- In programming language semantics: the "action semantics" framework [Mosses, 1996] defines the meaning of a program using a number of "facets" (such as control flow, functional, declarative, imperative, communicative, etc.), each of them focussing on one specific kind of information (control flow, data flow, scoping, memory, communications, etc.).

- In database systems: most database management systems provide a facility for defining and querying views which can be seen as abstractions of the actual database [Ullman, 1988]. Views are typically used to ensure that specific fields of the database are not accessible to certain classes of users or to provide information that is not explicit in the actual database (but can be derived from it).

The relevance of the notion of view for software architectures has already been advocated in [Kruchten, 1995] but, to our best knowledge, no existing ADL really supports the notion of multiple views. A major issue that must be addressed before introducing multiple views in an ADL is the specification of the correspondences between the different views. Ideally, a form of consistency should be defined and automatic tools should be provided to help in the design of an overall consistent architecture. The conformance of a system with respect to a given architecture is another significant issue. Although not specific to the notion of view, this problem is compounded by the introduction of multiple views. There may be a great diversity of views and it is unlikely that one single technique can be appropriate to ensure conformance with respect to each of them. None of the works on the variants of views mentioned above provides solutions to the consistency and the conformance problems for software architecture views, either because they do not rely on firm theoretical foundations or because they cannot be transposed to the general notion of view that is required in the context of software architectures. We believe that the study of these problems is a research direction of prime importance for software architectures. In the rest of this position paper, we mainly focus on consistency and summarise a preliminary approach that we are currently experimenting through two case studies: the specification of security properties of an information system and the design of a train control system.

CONSISTENCY AND HETEROGENEITY OF MULTIPLE VIEWS

In order to get the greatest benefits of the approach, the decomposition into multiple views should be chosen in such a way that it minimises the dependencies between views (otherwise little has to be gained from their separation). However all the views ultimately relate to the implemented system and they

can rarely be completely independent. For example, the distribution view can have some impact on the security view and the fault-tolerance view. Different views may even correspond to conflicting requirements: for instance adding a communication link can improve fault tolerance but also have a negative effect on security (because it may give rise to unwanted information flows).

The consistency problem is compounded by the inherent heterogeneity of multiple view architectures: since there can be a great diversity of views, each of them should be expressed in the framework (model, specification language, notation, etc.) that is best suited to its specific purpose. It is difficult to imagine a general and semantics based solution to the consistency problem in this context. Previous work on consistency across specification languages [Zave and Jackson, 1993, Bowman et al., 1996] has considered either an intermediate language or a common underlying implementation. For example [Bowman et al., 1996] states that "n specifications are consistent if and only if there exists a physical implementation which is a realisation of all the specifications, ie. all the specifications can be implemented in a single system". This implementation consistency is pictured in Figure 1.1.

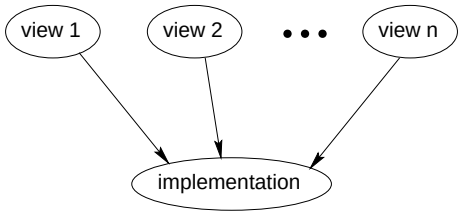

Figure 1.1 Implementation consistency

So in all cases mappings have to be defined from each view specification language to a single formalism. This solution may be appropriate when the formalisms used to describe the views are not of a too different nature (like Z and LOTOS considered in [Bowman et al., 1996]), which is precisely not the case for software architectures. So we believe that this "single formalism mapping approach" would defeat the very purpose of the introduction of views in software architecture. In other words, little would be gained, in terms of verification, if, to decide upon their consistency, all the views had to be merged into a single all embracing specification. We believe that the issues of consistency and conformance should better be decoupled for a better separation of concerns and an increased tractability. The approach that we are currently experimenting can be sketched as follows:

- **Consistency**: each view is represented as an uninterpreted labelled graph. Graphs are widely used representations of software architectures and all the views that we have considered so far (functional, distribution, physical, security, etc.) can be expressed naturally in terms of graphs. The fact that a view defined in a specific framework has first to be ex-

pressed as a graph may be seen as a limitation of the approach. Note however that the versatile nature of graphs and the fact that they are uninterpreted makes this translation easier. The uninterpreted nature of our graphs is a major departure with respect to previous work on consistency across specification languages: even if all the views are expressed as graphs, each of these graphs may convey very different kind of semantic information. The only restrictions on graphs are expressed through a collection of structural constraints (also called internal consistency rules), which bear, for example on the number of nodes with a given label or the number of edges (or paths) from (and/or to) a given node, etc. We consider inter-view consistency as a local relation between two views[2]. This relation is defined using a correspondence relation between the nodes (and edges) of the two views plus structural constraints similar to the internal consistency rules. Our approach, which introduces a clear separation between consistency and conformance, is depicted in Figure 1.2. The potential shortcoming of this local approach (as opposed to the "single formalism mapping approach") is that the number of correspondence relations may grow quadratically (in the worst case) with the number of views. We envision that the number of views will remain low though and the expected benefit is a lower complexity and a better understandability of correspondence relations.

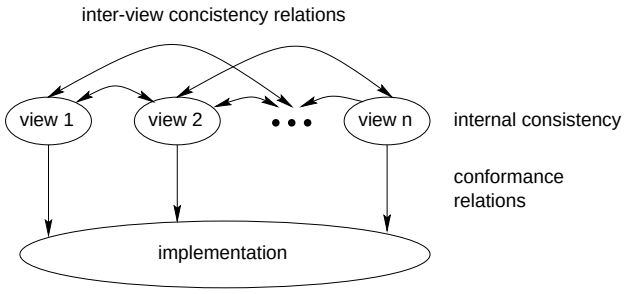

Figure 1.2 Separation of local consistency and conformance

- **Conformance**: in our approach, heterogeneity issues are isolated from the definition of consistency for a better separation of concerns. We believe that heterogeneity is better handled as a conformance problem between each view and the actual system. By definition, there is no general solution to this problem since the set of possible frameworks for the definition of view is open-ended and potentially varied. Just to take an example, there is very little in common between the proof that a program satisfies a given data-flow property and the verification that the

[2]Consistency relations involving more than two views can be defined as well, but they didn't turn out to be useful on the examples that we have treated so far.

bandwidth of a hardware bus is greater than a given threshold. Furthermore, it should be noted that this conformance process can either be bottom-up (recovering aspects of the architecture from the actual code) or top-down and prescriptive (when a view can be compiled into executable code, thus ensuring, by construction, that the actual system conforms to the view). We have followed the latter approach in the work described in [Holzbacher et al., 1997], which shows how the communication view can be implemented as an effective coordinator.

The ideas put forward in this paper are still preliminary and we are currently experimenting them on two case studies:

- The specification of the security aspects (confidentiality, integrity) of the information system of a research center: the different views in this case correspond to the services provided by the system: web server, rlogin server, ftp server, etc. The consistency constraints are used to ensure a global coherency of the security policy: for example, "if there is an edge with security level l between two nodes (or sites) n_1 and n_2, there should not exist any path between n_1 and n_2 in which all the edges have a security level strictly less than l".
- The design of a train control system: the relevant views are the functional view, the distribution view and the physical view. An example of consistency constraint is that the distribution view preserves the data flow of the functional view (which is a path property between the related nodes).

It is too early to draw any conclusion from these ongoing experiences but we feel that the approach sketched here is sufficiently practical to provide useful help in the design of multiple view architectures. Further interesting issues have emerged from these studies; among them let us just mention briefly software architecture analysis. The analysis of a software architecture gets more complicated when the properties of interest are non functional. The reason is that they can usually not be characterised by a predicate (they may even be subjective). Instead of providing a yes/no answer, we must return a concise and understandable piece of information that the user can estimate by himself. For example, in the case of a security property, the output of the analysis can be "for such a security breach to be possible, the intruder must be able to spy network N and have access to machine M".

References

[Allen and Garlan, 1994] Allen, R. and Garlan, D. (1994). Formalizing architectural connection. In *Proceedings of the 16th International Conference on Software Engineering*, pages 71–80. IEEE Computer Society Press.

[Benveniste and Berry, 1991] Benveniste, A. and Berry, G. (1991). The synchronous approach to reactive and real–time systems. *IEEE Trans. Autom. Control*, 9(79):1270–1282.

[Bourgois et al., 1998] Bourgois, M., Franklin, D., and Robinson, P. (1998). Applying RM-ODP to the Air Traffic Management Domain. EATCHiP Technical Document (CSD/architecture), Eurocontrol, Brussels.

[Bowman et al., 1996] Bowman, H., Boiten, E., Derrick, J., and Steen, M. (1996). Viewpoint consistency in ODP, a general interpretation. In Najm, E. and Stefani, J.-B., editors, *First IFIP International Workshop on Formal Methods for Open Object-Based Distributed Systems*, pages 189–204. Chapman & Hall.

[Cousot and Cousot, 1979] Cousot, P. and Cousot, R. (1979). Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 269–282.

[Finkelstein et al., 1992] Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., and Goedicke, M. (1992). Viewpoints: A framework for integrating multiple perspectives in systems development. *International Journal of Software Engineering and Knowledge Engineering*, 1(2):31–58.

[Holzbacher et al., 1997] Holzbacher, A. A., Périn, M., and Südholt, M. (1997). Modeling railway control systems using graph grammars: a case study. In *Proceedings of the Second Conference on Coordination Models, Languages and Applications*, volume 1282 of *LNCS*, pages 172–186, Berlin.

[Kiczales, 1996] Kiczales, G. (1996). Aspect-oriented programming. *ACM Computing Surveys*, 28(4es):154–154.

[Kruchten, 1995] Kruchten, P. B. (1995). The 4 + 1 view model of architecture. *IEEE Software*, 12(6):42–50.

[Loiseaux et al., 1995] Loiseaux, C., Graf, S., Sifakis, J., Bouajjani, A., and Bensale, S. (1995). Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):1–35.

[MacQueen, 1985] MacQueen, D. (1985). *Modules for standard ML.* Dept. Computer Science, University of Edinburgh.

[Mosses, 1996] Mosses, P. D. (1996). A tutorial on action semantics. Technical report, BRICS. Tutorial notes for FME'94 and FME'96.

[Radestock and Eisenbach, 1996] Radestock, M. and Eisenbach, S. (1996). Semantics of a Higher-Order Coordination Language. In Ciancarini, P. and Hankin, C., editors, *Proceedings of the Conference on Coordination Models, Languages and Applications*, volume 1061 of *LNCS*, pages 339–356.

[Shaw and Garlan, 1995] Shaw, M. and Garlan, D. (1995). Formulations and Formalisms in Software Architecture. In *Computer Science Today, Recent Trends and Developments*, volume 1000 of *LNCS*, pages 307–323.

[Ullman, 1988] Ullman, J. D. (1988). *Principles of database and knowledge-base systems, Vol. I.* Computer Science Press, Inc., Rockville, MD, USA.

[Zave and Jackson, 1993] Zave, P. and Jackson, M. (1993). Conjunction as composition. *ACM Transactions of Software Engineering and Methodology*, 2(4):379–411.