Architectural Descriptions as Models of Software*

Position Paper for the 2nd Australasian Workshop on Software Architectures (ASAW'99)

Eyðun Eli Jacobsen Bent Bruun Kristensen Palle Nowack
The Maersk Mc-Kinney Moller Institute for Production Technology
University of Southern Denmark/Odense University, DK-5230 Odense M, Denmark
e-mail: {jacobsen, bbkristensen, nowack}@mip.sdu.dk

1: Introduction

In every software system, some architecture is present, and even though the architecture is not described explicitly in our current programming technologies, the consequences of the architecture are indeed present in terms of the properties and qualities of the software. In the following we present one of the key ideas in our work with software architecture, namely architecture as software models. We introduce conceptual models of software in Section 2 and we characterize architecture models in Section 3. The ideas are further elaborated in [4].

In a brief characterization of current software architecture definitions (e.g. [5], [2], [6], [1]) of software architecture we claim that these describe *organization* only, not *abstraction*. We distinguish between organizing and abstracting: If we organize a certain domain, we only put some kind of order in the domain, among the elements in the given domain. If we abstract from a certain domain, we create new "concepts"/elements in another domain, an abstract domain. By moving to another domain we create knowledge about the former domain, and not just order. With respect to software systems, many such abstract domains are relevant and even useful. Because the architecture is so important with respect to the overall qualities of the software, we are interested in the set of abstractions which capture and describe the software architecture,

2: Conceptual Models of Software

A model is conceptual in the sense that it forms a conceptual understanding of something, and that it is based on concept formation in terms of classification, generalization and aggregation.

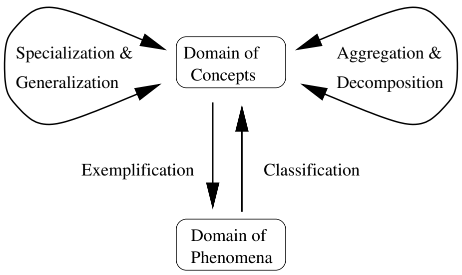

Figure 1. Abstraction Processes

Figure 1 illustrates the abstraction processes in conceptual modeling. Conceptual modeling can be seen as the underlying theoretical understanding of object-oriented programming although existing object-oriented notation only in a limited way supports this theory. The processes are fundamental abstraction processes.

*This research was supported in part by Danish National Centre for IT Research, Project No. 74.

In object-oriented programming an object/class can be seen as a model of a phenomenon/concept — to give the class of an object corresponds to classification — to give an object of a class to exemplification. To let a (specialized) class inherit from a (general) class corresponds to form a specialization of that (general) class. To describe a (whole) class by use of references to (part) objects corresponds to form an aggregation of (part) objects. The modelled phenomena and concepts can be considered to form a *referent system*, and the objects and classes can be perceived as expressing a *model system*.

3: Architecture as Models

Typically we apply object-oriented analysis to assist us in developing an object-oriented model system which can be transformed into a design and finally implemented. Thus, the problem and usage domains of the considered software system are initially considered as the referent system. However, one could just as well choose a software system as a referent system: we can all observe the software, and we can identify various kinds of phenomena in the software.

This means, that two significant and distinct application of abstraction can be identified in the software development process. In Figure 2 both these abstraction processes are shown and related.

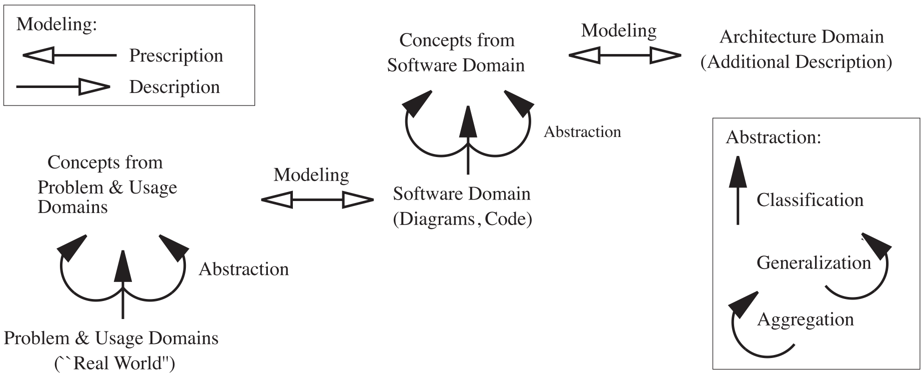

Figure 2. Abstraction & Modeling at Two Levels

During analysis we build models in order to make explicit our understanding of the problem and usage domains. We describe phenomena and concepts from the problem and usage domains with diagrams and code from the software domain. The problem and usage domains form the reference system and the software domain form the model system.

During design we build models in order to make explicit our understanding of the envisioned software. We describe phenomena and concepts from the software domain with diagrams and additional notation from the architecture domain. The software domain forms the reference system and the architecture domain forms the model system. Throughout design (and implementation) we form concepts over the software domain — such concepts do not categorize anything from the problem and usage domains (at least only indirectly). The software domain is the reference system. The model system explicitly describes the phenomena and concepts from the software domain. Concepts formed over the software domain include design patterns [3], architectural patterns [2], components, connectors, and architectural styles [5]. Architectural descriptions seek to describe such concepts in a rigorous way. The descriptions specify representations of architectural phenomena, that facilitate the interpretation and understanding of the software domain. However, there is still no established common and shared conceptual framework for describing such concepts, and there is still a lack of support for describing architectural models.

The illustration should be considered static; we do not impose any ordering of the described processes. Any practical development effort alternates between analysis and design. Many issues constrain the validation of the suitability of the models. A constraint of vital importance in object-oriented development is that the models formed during analysis are reflected in the models formed during design.

To sum up Figure 2 contains two instances of the referent and model system — the model system of the left instance is also the referent system of the right instance.

4: Conclusion

Our thesis is that software architecture is abstractions over the software domain. Some immediate consequences of this include that
(1) Architecture design requires abstraction. Abstraction implies, in addition to organization and delimitation, formation of concepts at another abstraction level together with use of the concepts at that level. We abstract from irrelevant details and concrete matters, given selected perspectives. But we have to find, introduce, and motivate the abstractions (and the perspectives). Enough experience in architecture design and insight in the given application is necessary to be able to come up with suggestions for *the right abstractions*, and to decide which are the correct and promising ones.
(2) The nature of abstractions over the software domain is still partially unexplored and unfamiliar. In our abstractions in object-oriented modeling the descriptions are concrete — abstract matters are related to problem and the usage domains. Abstractions over the software domain are different — probably more abstract and unreal. The challenge is to clarify the *essence of software abstractions*, and specify the quality measure for abstractions.

References

[1] L. Bass, P. Clements, R. Kazman, K. Bass. Software Architecture in Practice. Addison-Wesley, 1998.

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture: A System of Patterns. Wiley & Sons, 1996.

[3] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

[4] E.E. Jacobsen, B.B. Kristensen, P. Nowack: Architecture = Abstractions over Software. Proceedings of International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Pacific'99), Melbourne, Australia, 1999.

[5] M. Shaw, D. Garlan: Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall, 1996.

[6] C. Szyperski: Component Software. Beyond Object-Oriented Programming. ACM Press/Addison Wesley, 1998.