

DIRECTIONS IN ARCHITECTURAL SPECIFICATIONS

Amnon H. Eden¹

Department of Computer Science, Concordia University, Montreal, Canada

Desiderata

Architectural specifications provide software with “*a unifying or coherent form or structure*” [1]. Keeping this purpose in mind, the adequacy of architectural specifications ultimately depends on the language used for phrasing architectural statements. Let us examine the desirable qualities such a language:

- A. **Expressive.** Only naturally, coherent specifications must be based on an appropriate set of underlying abstractions [2]. The commonly accepted ontology (originally set forth in [1] and later developed in [3]) primarily consists of *components*, which are loci of data and computations, and *connectors*, which are means for transferring data and control between components. In the case of object-oriented architectures [4], one observes different abstractions, such as *uniform sets* of (possibly) high orders, *clans* (a set of dynamically-related methods), and *class hierarchies*.
- B. **Well-Defined, Precise.** Despite the intuitive appeal of box-and-lines diagrams, informal sketches may only give an idea on the general layout of a system, but, particularly with large and complex software systems, they cannot be reasoned upon, constrain future evolution cycles, be used by CASE tools, or resolve ambiguities. We conclude that a notation must be well-defined, and that it must be accompanied by an interpretation function that assigns each specification with an unambiguous model of the application domain.
- C. **Abstract.** Rather than describing a specific implementation, Perry and Wolf [1] have established that architectural specifications set forth abstract constraints that need be met, thereby specifying a *set* of programs indirectly by means of desired properties (also *intentional specifications*.) The appropriate level of abstraction may vary, but specifications must not include details that unnecessarily constrain the design and implementation processes that follow.
- D. **Concise.** Various function and object calculi and extensions thereof, such as λ -calculus, \mathbf{F}_{\langle} , and \mathbf{Ob}_{\langle} [5], provide a rigorous system for assigning a well-defined interpretation to predefined subsets of certain programming languages. Despite their contribution, these formalisms suffer from one glaring flaw: Their verbosity. Often, a few lines of source code translate into three pages of dense mathematical symbols, thereby defying the very attempt at achieving insights into the program. To preserve clarity and manageability, we conclude that statements (in the specification language) that set forth “simple” constraints should not exceed “reasonable” length. A more rigorous *concision* criterion is given in [4].
- E. **Compact.** “Small” languages are easier to acquire and use (as we learn from the case of PL/1). The compactness of a language promotes its clarity, facilitates proofs on its properties, and simplifies the reasoning with the predefined primitives.
- F. **Reasoning.** Formal reasoning frameworks, such as *predicate calculus* or *process algebras*, allow the users to derive properties from specifications. For example, a first order proof theory guarantees the decidability of any statement made therein. Thus, a well-defined notation that is shown to map into a well-understood mathematical framework should enhance our ability to reason with the specifications.
- G. **Tool support.** Finally, we separately require the specification language to facilitate tool support in the construction of and reasoning with architectural definitions. A tool, for instance, can automate the proof that a concrete program indeed conforms to a specification (e.g., a *reference architecture* or *architectural style*). In the case of large systems, such capacity is indispensable.

De Facto

Presently, the search for architectural specification languages is largely in its cradle. Noticeable directions include *Architecture Description Languages* (ADL), such as *Rapide* [6] and *Wright* [7], which combine an

¹ <http://www.cs.concordia.ca/~faculty/eden/>

underlying formalism for architectural specifications with tools for reasoning on specifications. ADLs often also allow step-wise refinement from specifications towards a working program. Unfortunately, so far, each ADL has proved to be restricted to a specific application domain, that is, generally being weak with respect to requirement 1 (*expressive*).

Focused on object-oriented design motifs, LePUS [8, 9, 10, 11] is a formal specification language with a visual equivalent that is defined by higher order monadic logic. LePUS is a very small (E) specification language. Also, it allows for highly succinct specifications of relatively complex patterns [12]. LePUS diagrams can be used not only for the specification of reference architectures, but they also can document complex programs, effectively capturing and conveying a clear picture of the complex relations. Unfortunately, it is yet to prove that tool support (G) in LePUS specifications is possible not only in principle.

Sizeable research effort is aimed at using the Unified Modeling Language [13] for architectural specifications, although it is a notation originally defined to support the design of object-oriented software design and development. UML is most certainly expressive (A), consisting of a hodgepodge of largely unrelated modeling notations concerning an assortment of domains, such as static and dynamic object modeling, procedural modeling, administrative concerns, managerial issues, and so forth, allowing for symbols-laden, verbose, and obfuscated specifications. As it is informally defined (B), it offers no advantage in terms of reasoning power (F). More importantly, while UML allows one to address miniscule implementation details, it does allow for generic specifications (excluding variables of any type), thereby fitting into a lower level of abstraction than needed (C). Despite the plenitude of constructs defined, it does not adequately incorporate the appropriate elements listed above (A). Finally, the large number of design concerns that were attempted by its designers, UML scores the lowest on *compactness* (E). Nonetheless, despite these obvious flaws, a curiously large number of publications offer to extend UML *even further* in the attempt to resolve some of them (e.g., the pUML group [14], and [15]).

Despite its wide popularity, this author considers UML too bloated and ill-defined to allow a well-defined extension therefrom which also will satisfy requirements (A) through (G).

References

- [1] D. E. Perry, A. L. Wolf (1992). "Foundation for the Study of Software Architecture". ACM SIGSOFT *Software Engineering Notes*, Vol. 17, No. 4, pp. 40-52.
- [2] G. Odenthal, K. Quibeldey-Cirkel (1997). "Using Patterns for Design and Documentation." *Proceedings of the European Conference of Object Oriented Programming 1997*. Lecture Notes in Computer Science. Berlin: Springer.
- [3] D. Garlan, M. Shaw (1993). "An Introduction to Software Architecture." *Advances in Software Engineering and Knowledge Engineering*, Vol. 2, pp. 1-39.
- [4] A. H. Eden, Y. Hirshfeld. "Principles in Formal Specification of Object Oriented Design." *CASCON 2001*, November 5-8, 2001, Toronto, Canada.
- [5] M. Abadi, L. Cardelli (1996). *A Theory of Objects*. Berlin: Springer-Verlag.
- [6] D. C. Luckham, J. Vera (1995). "An Event-Based Architecture Definition Language." *IEEE Transactions on Software Engineering*, Vol. 21, No. 9, pp. 717-734.
- [7] R. Allen, D. Garlan (1997). "A Formal Basis For Architectural Connection." *ACM Transactions on Software Engineering and Methodology*, Vol. 6, No. 3, pp. 213-249.
- [8] A. H. Eden (2000). "Precise Specification of Design Patterns and Tool Support in Their Application," PhD diss., Department of Computer Science, Tel Aviv University.
- [9] A. H. Eden. "Formal Specification of Object-Oriented Design." *International Conference on Multidisciplinary Design in Engineering CSME-MDE 2001*, November 21-22, 2001, Montreal, Canada.
- [10] A. H. Eden (2001). "A Theory of Object Oriented Software Architecture." Submitted: *Journal of Systems and Software*.
- [11] <http://www.cs.concordia.ca/~faculty/eden/lepup>
- [12] A. H. Eden. "Visualization of Object Oriented Architectures." *Workshop on Software Visualization, International Conference on Software Engineering*, May 13-14, 2001, Toronto, Canada.
- [13] G. Booch, I. Jacobson, J. Rumbaugh (1999). *The Unified Modeling Language Reference Manual*. Reading, MA: Addison-Wesley.
- [14] <http://www.cs.york.ac.uk/puml/>
- [15] D. Garlan, A. Kompanek. "Reconciling the Needs of Architectural Description with Object-Modeling Notations." *Proceedings of the Third International Conference on the Unified Modeling Language - << UML >> 2000*, October 2000, York, UK.