Software Design From a Verification Perspective

Tevfik Bultan
Department of Computer Science, University of California
Santa Barbara, CA 93106, USA
bultan@cs.ucsb.edu

In a typical software life-cycle, the implementation phase is preceded by the design phase. It is clear that the output of the implementation phase is a program in a high level programming language that can be compiled to executable code. However, it is not always clear what the output of the design phase is. It can be a textual description of the design, a set of hand-drawn diagrams, or a set of UML diagrams, all with varying degrees of ambiguity. I believe that there is an urgent need for better languages for developing and documenting software designs.

Programming languages have a significant influence on software design. I believe that, without the compiler technology, this influence would not have been as significant. Although object orientation is a very useful concept, had there not been compilers to automatically translate object oriented programs to machine code, object oriented design would not have been as popular. I believe that software design languages can be as influential in software development as programming languages if they are supported by automated verification tools. Forming a relationship between verifiers and design languages, similar to the relationship between compilers and programming languages, would change the cost/benefit ratio of using a design language: 1) Automated verification of software designs will reduce the future errors in the implementation phase. 2) Automated verification (and falsification) of software designs will lead to better understanding of the design by the software developers. 3) A design language that is tightly-coupled with a verification tool forces the language developers to formalize the semantics. This will prevent ambiguous interpretations that can occur in languages such as UML. 4) Existence of an automated verification tool will steer the software developers towards developing verifiable designs.

There has been significant progress in automated verification techniques in the last two decades based on model checking and automated theorem proving. I think these developments can be leveraged to software design by developing design languages which have verifiable features. There are already efforts in this direction such as Reactive Modules (by Alur and Henzinger) for software-hardware codesign, Alloy (by Jackson) for object oriented design, and Action Language (by my group) for concurrent system design. Below, I will discuss several concepts that can lead to verifiable design languages and talk about the interplay between verification technology and design language constructs.

Finite State Models: Model checking research has been especially successful in automated verification of finite state machines. Techniques based on symbolic representations (such as BDDs used by the SMV model checker) or efficient state space search (such as partial order reductions used by the SPIN model checker) have been successful in scaling to large finite state models. We need to leverage this technology in software design. There are numerous opportunities for using finite state machines in software design, such as, using finite state machines to model the states of an object as in UML state machines. Language constructs which promote/support finite state models should be part of software design languages.

Interfaces: State of the art programming languages do not provide adequate mechanisms for representing module interfaces. Interface of a module should provide the necessary information about how to interact with that module without giving all the details of its internal structure. Think of an object class in an object oriented language. The

interface of an object class consists of names and types of its fields, and names, return and argument types of its methods. This is a weak interface specification and does not contain sufficient information in most cases. For example, it contains no information about the order the methods of the class should be called. In recent years, model checking techniques have been successfully used for verification of module interfaces which are specified as finite state machines (SLAM project by Ball and Rajamani, interface automata by de Alfaro and Henzinger, and recent work on concurrency controller interfaces by my group). I believe that interface specification and verification should be an essential part of software design languages.

Behavior: Given multiple modules, it should be possible to check the correctness of their interaction by analyzing their interfaces. However, if one wants to check the correctness of an individual module, what should be the appropriate abstraction? Behavior specification of a module, which complements the interface specification, should provide this abstraction. Automated theorem proving tools which are specialized for software verification demonstrate that one can verify powerful properties about module behaviors. For example, Extended Static Checking techniques based on Simplify theorem prover developed at Compaq Systems Research Center demonstrate that non-trivial behavior specifications can be checked statically before generating executable code. These techniques can be used in a design language for behavior verification.

Modularity and Hierarchy: Modularity is an essential software design principle. Modularity is also essential for scalability of automated verification. A design language should provide language constructs for modular and hierarchical designs and these constructs should be tied to modular and hierarchical verification strategies. For example, Reactive Modules language and its verification tool Mocha use this approach.

Refinement and Abstraction: Forming a refinement relationship between different layers of a design, from more abstract to more concrete, is an essential tool in software design. There has been significant advances in automated abstraction of software in recent years, based on techniques such as predicate abstraction. Recent advances in decision procedures that combine multiple theories (such as, CVC and SVC by Dill's research group) are likely to improve automated abstraction techniques further. These tools and techniques can be used for automatically generating abstractions of software designs and for automatic refinement checking.

Infinite State Verification: Symbolic model checking technology has been extended to verification of infinite state systems with unbounded integers, queues, etc. Research by my group and others significantly improved the efficiency of verifying systems that can be specified using linear arithmetic constraints. Since it is rare to have nonlinear constraints in software designs, this technology can be effectively integrated to verification tools for software design languages.

Bounded Search: The advances in the efficiency of boolean satisfiability (SAT) solvers makes non-trivial bounded verification feasible. Using a bound on the design model or the verification search depth, SAT solver technology can be used to automatically verify software designs. For example, Alloy analyzer successfully leverages this technology by verifying bounded models of software designs, whereas in bounded model checking SAT solvers are used by bounding the search depth.

To conclude, I believe that software designs should be written in a design language as implementations are written in a programming language. A design language should be tightly-coupled with an automated verifier. Results from verification research should be leveraged in developing a design language that is amenable to automated verification. Software developers who use such a language will be motivated to produce verifiable designs.