Research Statement

Ramesh Bharadwaj

December 21, 2000

System and Software Engineering

The proliferation of the Internet and the World Wide Web has provided an increasingly ubiquitous and massively parallel *distributed computing infrastructure*, giving us the ability to develop a new generation of sensor-rich, massively parallel, distributed, and autonomous applications which have the potential for effecting profound social, environmental, logistic, and economic changes. We may take it as a given that the problems of infrastructure creation, network protocol design, quality of service, mobility, etc., have already been "solved" (however awkward or unsatisfactory the current solutions may be). The challenges of application development, however, still remain: How are these applications going to be built? How do we ensure that they behave in a reliable and predictable manner? How can we guarantee their performance? How can we make sure that they are self-stabilizing and fault-tolerant? How will they fully utilize this amazing infrastructure? I am convinced that answers to these questions lie in the application of sound engineering methods and tools – which are well founded in mathematics and logic – to the system development problem.

My research focuses on methods and tools for the specification, analysis, testing, and synthesis of application *frameworks*. By a framework I mean a set of components (software or hardware) that make up a reusable design for a specific application class. My current emphasis is on frameworks for embedded applications – such as control systems and signal processing systems – for the avionics, automotive, military, and aerospace industries. Typically, such systems are *safety critical*, i.e., a failure or malfunction during their operation can cost human lives.

In my most recent work, I extended the scope and applicability of the Software Cost Reduction (SCR) method, which was developed at the Naval Research Laboratories by Prof. David Parnas and others in the early eighties, to deal with issues of system engineering and hardware-software codesign [BH99]. The proposed method includes guidelines for developing and structuring the behavioral specifications of systems, sub-systems, or components, at a *very high level of abstraction*. Our initial application of this method has been to requirements elicitation, specification, and analysis which achieves multiple objectives: (a) The specifications provide a common vocabulary for users, domain experts, and software developers to work together, without being bogged down by low-level details such as code structure or user interface design issues. (b) The high level of abstraction makes the specifications amenable to automated static analysis such as checking for self-consistency and verification of required properties such as safety and security properties. (c) Being executable, the specifications may be animated, giving users and domain experts the ability to simulate specific scenarios. (d) Automatic behavioral synthesis frees developers from writing behavior specific "function driver" code, and lets them concentrate on more human-centric and generic modules such as code for user interfaces and device drivers.

Formal Verification

Model checking has emerged as an effective technique for the automated analysis of hardware and protocol descriptions. However, it is widely acknowledged that without a prior reduction to the size of the state space of a problem by the application of abstraction – a process that is manual, tedious, and often error-prone – model checking does not scale to handle realistic descriptions of software (due state explosion), since the state spaces of such descriptions are typically infinite or arbitrarily large. For such system descriptions, theorem proving has been the only viable alternative. The widespread use of model checking, however, is not because of its *verification* capability, but rather because of its ability for *refutation* – something current day theorem provers cannot do very well. Current generation theorem provers therefore have limited utility in practice. Also, since they require manual effort and mathematical sophistication to use, current day theorem proving technology is not cost effective and cannot be used directly by domain experts.

Recently, however, there has been the emergence of a "third way" [BS00]. In essence, this technology is founded on the inductive assertion approach pioneered by Floyd and Hoare, and employs decision procedures, i.e., fully automated techniques for proving theorems in *decidable* subtheories without regard to the size of the state space, which could be infinite or arbitrarily large, as embodied in early tools such as Shostak's Stanford Pascal Verifier. What is new is the emergence of a new generation of tools, such as my tool Salsa, which mimic the user interfaces of model checkers, i.e., which prove properties of state machine descriptions automatically and efficiently, and also provide counterexamples (potential candidates for refutation of stated properties) in case of failure. The key thing to recognize is that this technology allows us to carry out verification *more automatically* than model checkers (since there is no need for manual abstraction) *without* being hampered by the state explosion problem.

The tool Salsa has been successfully applied to a number of industrial strength case studies, which have repeatedly demonstrated the usefulness and applicability of this approach for the verification of system specifications of embedded and reactive systems, including software for *real-time* systems. More recently, Salsa was successfully applied by Prof. Rance Cleaveland and his colleagues to analyze behavioral models of power train controllers developed by engineers at the Ford Motor Company – models on which model checkers and process algebra tools failed to work. Salsa found a number of anomalies and bugs in these descriptions, leaving the Ford Engineers visibly impressed [Rance Cleaveland, personal communication]. This is what Prof. Cleaveland had to say about the Ford case study:

- Salsa "worked" on *real* models authored by Ford Engineers
- The hybrid system component did not rear its ugly head and
- The future of software verification is in invariant checking of infinite state systems [i.e., Salsa].

References

[BH99] R. Bharadwaj and C. Heitmeyer. Hardware/Software Co-Design and Co-Validation using the SCR Method. In Proc. *IEEE Int'l High Level Design Validation and Test Workshop (HLDVT'99)*, San Diego, CA, November 1999.

[BS00] R. Bharadwaj and S. Sims. Salsa: Combining constraint solvers with BDDs for automated invariant checking. In Proc. *Tools and Algorithms for the Construction and Analysis of Systems (TACAS '2000)*, Berlin, March 2000.