Towards Safe and Scalable Compositional Analysis and Programming

Azer Bestavros

http://www.cs.bu.edu/~best

Computer Science Department
Boston University

Specifying, designing, and developing correct, efficient, and resilient software systems is a notoriously hard problem, particularly when placing these systems in open contexts in which they will interact with dynamic and unpredictable environments, peers, and adversaries. By *"correct"* we mean that we know with certainty some desirable invariants of a system. Many techniques are already available to describe, discuss, and deduce the invariants of a single software component: type systems, model checking, mathematical analyses and countless derivative tools allow us to speak confidently about many local invariants (*e.g.,* well-formed output, minimum throughput, maximum response time, *etc*.)

While there are many interesting, useful, and plausibly verifiable properties of single software agents for which plausible verification systems do not yet exist, we do possess a fairly good handle on what invariant properties for single software components look like and how to go about expressing and testing them. What we do not yet have is a solid grasp upon how to describe, discuss, and deduce the *global invariants* of open, extensible software systems, or how to (hopefully efficiently) bridge the gap between local and global invariants, where global invariants describe the acceptable range of behaviors and emergent properties when the components or agents making up a system interact (*e.g.,* deadlock-free, stable, fair, *etc*.)

The "science" of service composition has clearly emerged as one of the grand themes driving many of our research questions in networking and distributed systems. This stems from the rise of sophisticated applications and new networking paradigms. By *service composition* we mean that the performance and correctness properties local to the various constituent components of a service can be readily composed into global (end-to-end) properties without re-analyzing any of the constituent components in isolation, or as part of the whole composite service. The set of laws that would govern such composition is what will constitute that new "science" of service composition. The combined heterogeneity and dynamic open nature of network systems makes composition quite challenging, and thus programming network services has been largely inaccessible to the average user.

We believe that this new science of composition must be built upon several theories (*e.g.,* control theory, game theory, network calculus, percolation theory, economics, queuing theory), each of which could be viewed as the most appropriate "language" via which certain properties of system components can be expressed and composed into larger systems. We then seek to lift these lower-level specifications to a higher level by abstracting away details that are irrelevant for safe composition at the higher level, thus making theories scalable and useful to the average user and more importantly for analysis of larger-scale systems.

Formal systems (like control or scheduling theory) allow us to abstract away some properties of a system to make it possible to reason about the system at a level which is impractical while retaining full detail. Sometimes, however, even these abstract representations afford us more detail than we actually require; in such cases, it may be advantageous to utilize "loose" descriptions of systems and components where those less precise descriptions are sufficient to demonstrate some desirable invariants. For example, it may suffice simply to know whether or not a controller is over-damped, whether the aggregate signaling path delay exceeds some threshold, whether the total steady-state error decreases exponentially, polynomially, or simply monotonically in time, or some combination of such properties. These simpler abstractions, in turn, may be more suitable for export into other domains which are interested in reasoning about high-level qualitative properties of the components and their interactions without getting bogged down in the minutiae of the particular analysis techniques used to derive those underlying (more precise) invariants and conclusions. Such an approach reflects the very essence of a type in the programming languages and formal models sense: an abstract description of some aspect of a system which captures interesting invariants (convergence, TCP-friendliness, rate and time bounds) while discarding detail

which may hamper analysis (*i.e.,* which is either unnecessary for proving desirable invariants at higher layers of abstraction, or which may make a decision algorithm intractable or even non-computable).

Simply put, we believe that a more scalable and accessible approach to the checking of systems for correctness is to move away from *internal models*, in which the inner workings of a component are reduced to a technique-specific abstract caricature, toward *external models*, in which the workings of components are caricatured qualitatively and quantitatively at their interfaces. This allows us to separate representation from computation, *i.e.,* to reason about and form conclusions without regard for their supporting analysis. It is with these observations in mind that we have proposed the TRAFFIC framework (Typed Representation and Analysis of Flows For Interoperability Checks), in which each of the components of a composite system is represented simply as a black box, with all correctness constraints presented at its points of composition (edges) in the form of types. Our TRAFFIC framework is an ingredient of the larger iBench (The Internet Programming WorkBench) initiative at Boston University, whose aim is the development of a rigorous discipline for the specification, verification, programming, and maintenance of distributed applications and services over the Internet.

The TRAFFIC framework provides a specification language that is expressive enough to describe different components of a network service, and one which supports type hierarchies inspired by type systems in general programming languages that enable the safe composition of software components.[1] One of the salient properties of the TRAFFIC framework is that it affords its users (software designers and programmers alike) a seamless tradeoff between model complexity (how much of a black-box's internal details are exposed at the boundaries) and expressive power (what safety and security properties we are able to ascertain) on the one hand, and scalability (how large a system are we able to verify) on the other hand. This tradeoff is crucial and is one of the major benefits from adopting a type-theoretic approach to tackling this problem.

Defining the proper (formal) interface between the various system components that are to be composed together – including those underlying the hosting infrastructure – is a balancing act between efficiency and expressive power. We argue that the difficulty we face today in verifying and certifying (let alone understanding) the behavior of large, distributed software artifacts is the result of our inability to find the right tradeoff between efficiency and expressive power.

Here, history may well be our guide. Today, and to a large extent, our ability to build large trustworthy software systems suffers from the same lack of organizing principles as did programming of stand-alone computers some thirty years ago. Primeval programming languages were expressive but unwieldy; software engineering technology improved not only through better understanding of useful abstractions, but also by automating the process of verification of safety properties both at compile time (*e.g.,* type checking) and run times (*e.g.,* memory bound checks). What programming languages have done to software engineering is to force programmers to adopt a disciplined approach to programming that reduces the possibility of "bugs" and by making it possible to mechanically check (whether at compile time or run-time) for unacceptable specifications/behaviors. In many ways, this was done at the expense of reducing the expressive power given to programmers. High-level programming languages do not afford to programmers the same expressive power that assembly language does (*i.e.,* there are programs that one can write in assembly language that one cannot write in Java).

High-level abstractions that restrict expressiveness are not unique to programming languages, they are certainly the norm in Operating Systems, whereby programmers are allowed to interact with (say) system code and resources in prescribed (less expressive) ways. This loss of expressive power is precisely what has enabled us to deal with issues of scale of software artifacts and systems.

Along these lines, and to summarize, the same kinds of benefits in dealing with issues of scale and complexity could find their way into the verification and certification processes of large software systems used in support of our critical infrastructure, if we adopt a more disciplined approach – an approach that: (1) seamlessly enables a tradeoff between expressive power and scalability of verification, and (2) allows for compositional analysis that leverages multiple underlying theories. These are precisely the principles guiding our design of the TRAFFIC framework, which is part of our larger iBench initiative at Boston University.

[1] Interested readers are invited to try out the web interface accessible from http://www.cs.bu.edu/groups/ibench for examples of how TRAFFIC is used to specify/verify compositions of network services, using results from encapsulated theories (e.g., network calculus).

References [2]

- The iBench Initiative at Boston University: http://www.cs.bu.edu/groups/ibench. The iBench initiative is supported partially by a National Science Foundation ITR grant entitled *Internet Flows as First-Class Values: Support for Dynamic, Flexible Internet Services.*
- Azer Bestavros, Adam Bradley, Assaf Kfoury, and Ibrahim Matta. Safe Compositional Specification of Networking Systems. *ACM SIGCOMM Computer Communication Review (CCR)*, 34(3), July 2004.
- Azer Bestavros, Adam Bradley, Assaf Kfoury, and Ibrahim Matta. Typed Abstraction of Complex Network Compositions. In *Proceedings of ICNP'05: The 13th IEEE International Conference on Network Protocols*, Boston, MA, November 2005.
- Azer Bestavros, Adam Bradley, Assaf Kfoury, and Michael Ocean. SNBENCH: A Development and Run-Time Platform for Rapid Deployment of Sensor Network Applications. In *Proceedings of the IEEE International Workshop on Broadband Advanced Sensor Networks (Basenets 2005)*, Boston, MA, October 2005.
- Adam Bradley, Azer Bestavros, and Assaf Kfoury. A Typed Model for Encoding-Based Protocol Interoperability. In *Proceedings of ICNP'04: The 12th IEEE International Conference on Network Protocols*, Berlin, Germany, October 2004.
- Adam Bradley, Azer Bestavros, and Assaf Kfoury. Systematic Verification of Safety Properties of Arbitrary Network Protocol Compositions Using CHAIN. In *Proceedings of ICNP'03: The 11th IEEE International Conference on Network Protocols*, Atlanta, GA, November 2003.
- Adam Bradley, Azer Bestavros, and Assaf Kfoury. Safe Composition of Web Communication Protocols for Extensible Edge Services. In *Proceedings of the 7th International Web Caching and Content Delivery Workshop*, Boulder, CO, August 2002.

[2] Please contact the author (best@cs.bu.edu) for other publications, presentations, and for unpublished results and on-going work.