Specification, analysis and verification of distributed systems

At an increasing rate, humanity is creating distributed systems through hardware and software—systems consisting of multiple components that interact with each other through message passing or other synchronisation mechanisms. Examples are distributed databases, communication networks, operating systems, industrial control systems, etc. Many of these systems are hard to understand, yet vitally important. Therefore, significant effort needs to be made to ensure their correct working.

Formal methods are an indispensable tool towards that end. They consist of specification formalisms to unambiguously capture the intended requirements and behaviour of a system under consideration, tools and analysis methods to study and reason about vital properties of the system, and mathematically rigorous methods to verify that (a) a system specification ensures the required properties, and (b) an implementation meets the specification.

The standard alternative to formal specification formalisms are descriptions in English, or other natural languages, that try to specify the requirements and intended workings of a system. History has shown, almost without exception, that such descriptions are riddled with ambiguities, contradictions and under-specification. Formalisation of such a description—regardless in which formalism—is the key to elimination of these holes.

A formal specification of a distributed system typically comes in (at least) two parts.

One part formulates the *requirements* imposed on the system as a list of properties the system should have. Amongst the formalisms to specify such requirements are temporal logics like Linear-time Temporal Logic (LTL) [Pnu77] or Computation Tree Logic (CTL) [EC82]. Amongst others, they can specify *safety properties*, saying that something bad will never happen, and *liveness properties*, saying that something good will happen eventually [Lam77].

The other part is a formal description of how the system ought to work on an *operational* (= step by step) basis, but abstracting from implementation details. For distributed systems such accounts typically consist of descriptions of each of the parallel components, as well as of the communication interfaces that specify how different components interact with each other. Languages for giving such formal descriptions are *system description languages*. When a system description language features constants to specify elementary system activities, and operators (like parallel or sequential composition) to create more complex systems out of simpler ones, it is sometimes called a *process algebra*. Alternatively, operational system descriptions can be rendered in a model of concurrency, such as Petri nets or labelled transition systems. Such models are also used to describe the meaning of system description languages.

Once such a two-tiered formalisation of a system has been provided, there are two obvious tasks to ensure the correct working of implementations: (a) guaranteeing that the operational system description meets the requirements imposed on the system, and (b) ensuring that an implementation satisfies the specification. The latter task additionally requires a definition of what it means for an implementation to satisfy a specification, and this definition should ensure that any relevant correctness properties that are shown to hold for the specification also hold for the implementation.

A third type of task is the study of other properties of the implementation, not implied by the specification. Examples are measuring its execution times, when these are not part of the specification, or its success rate, for operations for which success cannot be guaranteed and only a best effort is made. Potentially, these tasks call for applications of probability theory.

Traditional approaches to ensure the correct working of distributed systems are simulation and testbed experiments. While these are important and valid methods for system evaluation, in particular for quantitative performance evaluation, they have limitations in regards to the evaluation of basic correctness properties. Experimental evaluation is resource-intensive and time-consuming, and, even after a

very long time of evaluation, only a finite set of operational scenarios can be considered—no general guarantee can be given about correct system behaviour for a wide range of unpredictable deployment scenarios. I believe that formal methods help in this regard; they complement simulation and test-bed experiments as methods for system evaluation and verification, and provide stronger and more general assurances about system properties and behaviour.

Achievements of process algebra and related formalisms

Process algebra is a family of approaches to the specification, analysis and verification of distributed systems. Its tools encompass algebraic languages for the specification of processes (mentioned above), algebraic laws to reason about processes, and induction principles to derive behaviours of infinite systems from those of their finite approximations.

Many industrial size distributed systems have been successfully specified, analysed and verified in frameworks based on process algebra. Examples can be found though the following links. Major toolsets primarily based on process algebra include FDR [GABR14], CADP [GLMS11], mCRL2 [GM14] and the Psi-Calculi Workbench [BGRV15, BJPV11]. Most of these toolsets also use model checking or other mathematical techniques that explore the state spaces of distributed systems. Similar toolsets primarily based on the latter techniques include SPIN [Hol04], UPPAAL [BDL04], PRISM [KNP10] and TLA [Lam02].

References

[BDL04] G. Behrmann, A. David & K.G. Larsen (2004): *A Tutorial on Uppaal*. In M. Bernardo & F. Corradini, editors: Revised Lectures on *Formal Methods for the Design of Real-Time Systems*, LNCS 3185, Springer, pp. 200–236, doi:10.1007/978-3-540-30080-9_7.

[BGRV15] J. Borgström, R. Gutkovas, I. Rodhe & B. Victor (2015): *The Psi-Calculi Workbench: A Generic Tool for Applied Process Calculi*. *ACM Trans. Embedded Comput. Syst.* 14(1), pp. 9:1–9:25, doi:10.1145/2682570.

[BJPV11] J. Bengtson, M. Johansson, J. Parrow & B. Victor (2011): *Psi-calculi: a framework for mobile processes with nominal data and logic*. *Logical Methods in Computer Science* 7(1), doi:10.2168/LMCS-7(1:11)2011.

[EC82] E.A. Emerson & E.M. Clarke (1982): *Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons*. *Science of Computer Programming* 2(3), pp. 241–266, doi:10.1016/0167-6423(83)90017-5.

[GABR14] T. Gibson-Robinson, P.J. Armstrong, A. Boulgakov & A.W. Roscoe (2014): *FDR3 - A Modern Refinement Checker for CSP*. In E. Ábrahám & K. Havelund, editors: Proceedings *Tools and Algorithms for the Construction and Analysis of Systems* - 20th International Conference (TACAS'14), Grenoble, France, April 2014, LNCS 8413, Springer, pp. 187–201, doi:10.1007/978-3-642-54862-8_13.

[GLMS11] H. Garavel, F. Lang, R. Mateescu & W. Serwe (2011): *CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes*. In P.A. Abdulla & K.R.M. Leino, editors: Proceedings *Tools and Algorithms for the Construction and Analysis of Systems* - 17th International Conference (TACAS 2011), Saarbrücken, Germany, March 26-April 3, 2011, *Lecture Notes in Computer Science* 6605, Springer, pp. 372–387, doi:10.1007/978-3-642-19835-9_33. Available at http://dx.doi.org/10.1007/978-3-642-19835-9_33.

[GM14] J.F. Groote & M.R. Mousavi (2014): *Modeling and Analysis of Communicating Systems*. MIT Press.

[Hol04] G.J. Holzmann (2004): *The SPIN Model Checker - primer and reference manual*. Addison-Wesley.

[KNP10] M. Kwiatkowska, G. Norman & D. Parker (2010): *Advances and Challenges of Probabilistic Model Checking*. In: *Proc. 48th Annual Allerton Conference on Communication, Control and Computing*, IEEE Press, pp. 1691–1698.

[Lam77] L. Lamport (1977): *Proving the correctness of multiprocess programs*. *IEEE Transactions on Software Engineering* 3(2), pp. 125–143, doi:10.1109/TSE.1977.229904.

[Lam02] L. Lamport (2002): *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley.

[Pnu77] A. Pnueli (1977): *The Temporal Logic of Programs*. In: *Foundations of Computer Science (FOCS '77)*, IEEE, pp. 46–57, doi:10.1109/SFCS.1977.32.