Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2012-027 August 27, 2012

# Using Timed Input/Output Automata for Implementing Distributed Systems

Peter M. Musial

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

# From Formal Methods to Executable Code

Peter M. Musial
CSAIL, MIT, MA, USA
pmmusial@csail.mit.edu

## Abstract

In this paper we will discuss one approach to achieving software reliability. In particular, where software systems are modeled using a formal mathematical framework that is used to verify their behaviors. Once verified these are translated to executable code. Formal system specifications and their behavior analysis are valuable tools that should be at the disposal of the software developers, especially when dealing with systems exhibiting high levels of concurrency. However, theoretically sound specifications have a limited impact, unless tools exist that automatically transform these specifications from high level representation to executable code. One challenge that arises with this approach is to provide a comprehensive and usable set of abstractions (such as files, network protocols, console, etc.) that will serve as building blocks of the abstract software models. Another difficulty is to ensure performance of the generated code. Finally, the translation process has to be formally verified to result in executable code that can be deemed as reliable and correct by its construction.

## Introduction

In designing distributed systems, the best practice today involves a patchwork of specifications, including graphical object modeling tools, manual documentation of component interaction, formal specification of interfaces, descriptions of algorithms and protocols with varying degrees of formalism, and specifications of distributed system configuration and deployment. Even when services and algorithms are specified formally, rigorous reasoning about the specifications is often left out of the development process. Without a comprehensive design framework, it is very difficult to ensure that all necessary types of specifications are produced within the design effort. It is usually the case that when a distributed system is first deployed, numerous forgotten or underspecified aspects of the system begin to surface. Granted that one is able to amass all necessary specifications, it is extremely difficult to deal with the dissimilar kinds of specifications and specification formats and media. During the development of a system, it becomes difficult to maintain traceability between the specifications and the resulting implementation. This problem is further aggravated when an existing distributed system needs to be refined, optimized, extended or redeployed. Many of these problems remain outside of the realm of academic research.

## Background

Several formal frameworks exist for modeling and reasoning about complex systems [12, 26, 6, 5, 16] (to name a few) and for which software support was developed [10, 25, 13, 7, 18, 28]. These frameworks provide a high level notation that can be used to express concurrent systems (resp. algorithms) at various levels of abstraction, and the mathematical support to reason about their properties. However, for the aforementioned and other frameworks we found a very limited or nonexistent support for automated software development.

During implementation, when high level abstractions are left up to human interpretation then this opens a possibility of undesirable behaviors being introduced into the final code thereby nullifying all formal efforts. There is a documented success for automated code generation for embedded systems [14]. A natural question to ask if the same can be repeated for concurrent systems in general or under what constraints.

Our research is based on the Timed Input/Output Automata [15] (TIOA) formal framework that allows modeling timed & untimed systems and reasoning about their behaviors. The TIOA framework supports a rich set of proof techniques. Invariant assertion techniques are used to prove that properties of automata are true in all reachable states [29]. Compositional reasoning where properties of a system are evaluated by reasoning about each of its components individually [22]. Another important proof strategy is hierarchical proofs [21] where comparable automata are tested whether one implements the other. TIOA framework (and its predecessor Input/Output Automata [24]) have been used to model and verify a wide variety of distributed systems and algorithms [22, 23, 31, 9] and to express and prove several impossibility results. The TEMPO [20] toolkit developed by VeroModo, Inc. [33] contains tools to support analysis of systems such as, a *checker* that checks syntax and performs static semantic analysis; a *simulator* to produce and explore execution traces for an automaton; a *translation module* to the UPPAAL model-checker [19]; and a *translation module* to the PVS interactive theorem prover [30]. Other Extensions are possible in form of additional plugins.

We find that the Nuprl [1] proof development system and its toolkit [28] present the most relevant body of work. In [3, 11] a framework consisting of Nuprl, Input/Output Automata [22] framework, and the O'Caml programing model have been used to verify certain properties the Ensemble [4], a group communication system and its implementation. Also the Nuprl toolkit [28] supports translation modules to both Java and O'Caml. A contribution of [3, 11] was its formal approach to inheritance and its semantics.

In contrast, we place TIOA at the center and use its notation to model systems and its mathematical support for verification. The TEMPO toolkit implements a software framework around TIOA with native analysis and translation tools and integration of external verification environments.

## TEMPO as a Design Tool

Flexibility and power of the TIOA framework is contributed to (i) a designer can utilize nondeterminism to allow multiple correct system specifications, hence relaxing assumptions on behaviors of the environment in which it operates (at least at certain parts of the execution); (ii) complex systems can be decomposed into subsystems, where composition of these subsystems yields the unified system abstraction. Such structured design enables one to view specifications at multiple levels of abstraction. Since TEMPO is derived from TIOA, it inherits all of its properties.

The two key problems mentioned in the introduction are the absence of formalism during the design phase and the difficulty with traceability between design and implementation phases. Both characteristics (i) and (ii) can be used to remedy these. The TIOA framework allows designers to use behavior specifications which are more abstract and more natural in expressing system requirements. Once a system is modeled in the TIOA notation, its behavioral specification can be verified using native mathematical support (or external tools). Finally, by use of layered abstractions a designer can successively refine complex specification with additional level of detail until the model is suitable for the final transformation step to executable code. This process allows traceability between specifications and the resulting implementations.

The TEMPO framework extends the TIOA notation and provides a high-level programming environment. The addition of the translation module to Java (discussed next) extends its utility and provides a link from high-level formalism to low-level implementation.

## Translation to Java

Employing properties of the TIOA framework and the ease of new functionality integration to the TEMPO toolkit we designed and implemented a TEMPO translation module to executable Java programs [27]. This tool allows us to derive Java codes that are correct by construction, where the translation rules have been formally verified to preserve properties of the source model; i.e., the generated Java code can be viewed as a specialization of the source model, and that the set of its allowed behaviors solves the same problem as its source.

The TEMPO notation allows modeling of almost any type of concurrent system, where not all can be expressed as executable programs (the TIOA is much more powerful with few theoretical limitations in contrast to programming languages constrained by properties of targeted physical platforms). For this reason some restrictions are imposed on the input models. Systems admissible for compilation should be in the *node-channel* form that reflects the message-passing architecture of networked processing nodes and properties of the communication medium. It is the responsibility of the system designer to decide on the distribution of computation for the intended deployment setting. Given the generality of the TEMPO framework, each networked component can run a different node algorithm. In that case compilation proceeds on the node-by-node basis. Alternatively, a node algorithm could be general enough to allow it to be executed on any number of nodes. Java as the target programming language ensures portability of the generated programs (through specialized Java Virtual Machines) and hides architectural specifics. In order to further promote portability we have chosen MPI implementation presented in [2] and also support communication channels specialized to TCP sockets.

Translation from TEMPO notation to Java is in a way a natural process with some careful steps taken along the way. The key challenge is to provide a library of usable abstractions, such as the ones to the MPI and TCP channels. Additionally our aim is to make the generated code as efficient as possible and at the same time human readable. This approach allows us to deliver an initial code that encourages correctness and that can be further optimized by a programmer who does not need to understand the intricacies of the TEMPO model. Of course any manual augmentation can introduce unwanted behaviors and software bugs, so all correctness guarantees are being voided.

Thus far this new tool was used to generate executable code for Paxos [32, 17, 27] and a few other complex algorithms. Robustness of these translations is currently being tested on a variety of deployment platforms. We are continuing our research to understand limitations of this tool and its applicability to the development of large scale systems.

## Limitations and Future Work

The development of TEMPO plugins is continuing. In particular the simulator and the translator both require more work in order to be considered as practical development tools. The toolkit would further benefit from creation of additional plugins providing link to the new proof management systems such as Coq [8]. We are also planning on supplementing the set of plugins with additional code translation modules, for example to the C/C++ programming language.

## Conclusions

The nature of software, hardware, and requirements engineering demands unique solutions in order to ensure that software systems perform as expected. Work presented in this paper presents a small step towards closing the existing gap between formal model abstractions and their software implementation. Given the ability of innovation in the theory and practice of computer science, one expects this gap to be only temporary.

# References

[1] S. Allen, M. Bickford, R. Constable, R.L. andEaton, C. Kreitz, L. Lorigo, and E. Moran. Innovations in computational type theory using nuprl. *Journal of Applied Logic*, 4(4):428–469, 2006.

[2] M. Baker, B. Carpenter, and A. Shafi. MPJ express: Towards thread safe java hpc. *In the IEEE International Conference on Cluster Computing*, pages 25–28, September 2006.

[3] M. Bickford and J. Hickey. An object-oriented approach to verifying group communication systems, 1998.

[4] K. Birman, B. Constable, M. Hayden, J. Hickey, C. Kreitz, R. Van Renesse, O. Rodeh, and W. Vogels. The horus and ensemble projects: accomplishments and limitations. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings*, volume 1, pages 149 –161 vol.1, 2000.

[5] S. Budkowski. Estelle development toolset (edt). *Computer Networks and ISDN Systems*, 25(1):63–82, 1992.

[6] S. Budkowski and P. Dembinski. An introduction to estelle: A specification language for distributed systems. *Computer Networks and ISDN Systems*, 14(1):3–23, 1987.

[7] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15, 1994.

[8] Coq home page. `http://coq.inria.fr`.

[9] R. Fan, R. Droms, N. Griffeth, and N. Lynch. The dhcp failover protocol: A formal perspective. In J. Derrick and J. Vain, editors, *Formal Techniques for Networked and Distributed Systems FORTE 2007*, volume 4574 of *Lecture Notes in Computer Science*, pages 211–226. Springer Berlin / Heidelberg, 2007.

[10] D. Harel. Statecharts: A visual formalism for complex systems, 1987.

[11] J. Hickey, N. Lynch, and R. van Renesse. Specifications and proofs for ensemble layers. In *TACAS '99*, pages 119–133. Springer-Verlag, 1999.

[12] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21:666–677, August 1978.

[13] iNMOS Ltd. *Occam 2 Refrence Manual*. Prentice-Hail, 1987.

[14] N. Izerrouken, O. S. Y. Kai, M. Pantel, and X. Thirioux. Use of formal methods for building qualified code generator for safer automotive systems. In *Proceedings of the 1st Workshop on Critical Automotive applications: Robustness &#38; Safety*, CARS '10, pages 53–56, 2010.

[15] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. *The Theory of Timed I/O Automata*. Synthesis Lectures on Computer Science. Morgan Claypool Publishers, 2006.

[16] L. Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, 1994.

[17] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[18] L. Lamport, D. Ricketts, S. Zambrovski, and Others. TLA+2. `http://research.microsoft.com/en-us/um/people/lamport/tla/tla2.html`.

[19] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1:134–152, 1997.

[20] N. Lynch, L. Michel, and A. A. Shvartsman. Tempo: A toolkit for the timed input/output automata formalism. In *In Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks, and Systems – Industrial Track: Simulation Works*, 2008.

[21] N. Lynch and F. Vaandrager. Forward and Backward Simulations - Part I: Untimed Systems. *Information and Computation*, 121:214–233, 1995.

[22] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[23] N. A. Lynch and M. Merritt. *Atomic Transactions: In Concurrent and Distributed Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[24] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2:219–246, 1989.

[25] N. Marti-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework, 1996.

[26] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[27] P. Musial. Using Timed Input/Output Automata for Implementing Distributed Systems. Technical report, Cambridge, MA, USA, 2011.

[28] Nuprl home page. `http://www.nuprl.org`.

[29] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs I. *Acta Informatica*, 6:319–340, 1976. 10.1007/BF00268134.

[30] S. Owre, S. Rajan, J. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T. A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[31] A. Pogosyants, R. Segala, and N. Lynch. Verification of the randomized consensus algorithm of aspnes and herlihy: a case study. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1997.

[32] R. D. Prisco. *Revisiting the Paxos Algorithm*. PhD thesis, Massachusetts Institute of Technology, 1991.

[33] Tempo People. VeroModo, Inc. `http://www.veromodo.com`.