

Abstracting Socket-based, Distributed, Communicating Processes

Applying CFMs and MSCs to TCP/IP based Applications

Alexander Heußner

LaBRI, Université Bordeaux 1
351 cours de la Libération
33405 Talence cedex, France
heussner@labri.fr

SUPERVISORS: Anca Muscholl, Jérôme Leroux, Grégoire Sutre (LaBRI)
KEYWORDS: communicating FIFO machines, model checking, Berkeley sockets, message sequence charts, network protocols

Abstract. The following project description presents our ideas of applying Communicating FIFO Machines (CFM) and Message Sequence Charts (MSC) based symbolic model checking to the verification of distributed software that communicates with the help of messages over a TCP/IP based network and that is based on a well-defined API like Berkeley sockets. After a short introduction, we will present our motivation and a brief comparison to existing approaches as well as give a brief agenda of our future research.

This project is still in its initial phase; hence, the following introduction will present a survey of the underlying research questions and an outlook on the planned proceeding. Results in form of rigorous proofs, a prototypical implementation, or a practical application will be part of future work. Nevertheless, the following paragraphs will serve as a base for discussing our ideas in the context of MOVEP 2008.

1 Introduction

One class of important applications that depends essentially on correctness and reliability are the nowadays ubiquitous distributed systems that exchange messages and data with the help of a protocol over an underlying TCP/IP-based network. These systems can be seen as a set of processes that communicate over a network by some kind of protocol on top of TCP or UDP. Hence, a correct implementation of TCP offers message passing over FIFO channels to the application layer above straight off. Concerning the implementation of this protocol, the underlying network is abstracted with the help of an API. Further, these processes also include sophisticated ways of using concurrency to allow their deployment in environments that depend on a high throughput and the

management of a large number of simultaneous connections to other peers (i.e., processes that normally run on other machines).

In the following, the focus will reside on the correctness of the implementation regarding an a priori given specification as well as the interplay between the global and local concurrency (and therewith the “classical” set of problems like deadlock, livelock, etc.). In a first approximation, the underlying network is assumed to work correctly and that its behaviour can be described relative to the API calls of the participating processes. This contrasts, for example, with the large research area on the verification of security protocols, viz. [Bel07].

From the wide range of possible formal models, Communicating FIFO Machines (CFM, based on Communicating Finite State Machines [BZ83]) and – to a limited degree – Message Sequence Charts (MSC) [GMP03] are chosen to abstract and represent the behaviour of these distributed systems. From an implementational point of view, CFM seem a promising and simple model that captures these processes and their exchange of messages in a quite natural way.

MSC allow an even more abstract view onto the sequence of exchanged messages but they depend on a deeper knowledge of the ideas “behind” the protocol, which are easily available at the stage of protocol-/software-engineering (because of MSC’s proximity to basic modelling representation languages like UML sequence diagrams), but which are difficult to derive from a legacy code base.

Nevertheless, both formalisms allow to use symbolic model checking in order to avoid the direct computation of the reachable state space or even the simulation of the real implementation in a test environment (e.g., as done for the TCP protocol in [ME04]).

1.1 Related Work

Regarding the application area of distributed systems, the closest approach is that of [CMGMS05] which uses “classical” model checking based on a state-space exploration for the verification of client-server programs on top of the Berkeley socket API. The state space exploration – and the model checking of properties expressed in LTL – is passed on to SPIN whereas the C code is translated to a PROMELA model (viz. [Hol01] for the underlying ideas) enriched with the semantic model of the API’s system calls. This general procedure can be applied to other arbitrary well-defined API-based models [CMGM07] and enriched with simple means of abstraction [CMGM06], but does not allow the sophisticated methods possible with the framework of CFM and MSC.

There is a large variety of different approaches that try to abstract CFM-based models in order to avoid the inherent undecidability results of this formalism [BZ83] and to allow to answer simple reachability questions symbolically (under certain boundary conditions); the most famous approaches are: SRE [AABJ04], QDD [BGWW97], CQDD [BH99], and SLRE [FIS03]. These four ways to use (extensions of) regular languages as abstractions include a simple semi-algorithm (which allows for a simple loop acceleration scheme) to answer reachability questions. Recent advances like [LGJ07] enhance CFM with the capability to exchange

messages over a lattice and introduce ideas from Abstract Interpretation [CC77] to allow a simple automata-theoretic way to derive a symbolic approximation.

TREX, the Tool for the Reachability Analysis of CompleX Systems [ABS01], allows to check a wide range of properties for timed automata (with parameters and counter variables, that communicate by shared variables and *lossy* FIFO channels). Nevertheless, its input requires to be SDL, LOTOS, or any other language compatible with the IF toolbox; regarding our goal to verify legacy code, a preprocessor that abstracts this code into a model in these languages is inevitable. Nevertheless, TREX applies a symbolic approach and will serve as a friendly competitor that allows to compare some of our results. Albeit focused on another modus operandi and other goals, CADP [GMLS07] would propose another standard of comparison.

Message Sequence Charts (MSC) is a popular specification formalism used for telecommunication protocols, that corresponds to the ITU norm Z.120. MSC is a high-level, graphical notation and a lot of recent research was devoted to model-checking with respect to MSC properties and synthesis of CFM from MSC specifications [GKM07].

In contrast to the approaches that rely on symbolic representations of channels, the MSC-based technique refers to (sets of) execution traces and exploits the inherent concurrency of the network of communicating processes. Applying MSC led to identifying a realistic restriction on CFM behaviors, which allows to do exact model-checking. This restriction consists in assuming that message events can be scheduled in such a way that channels of a given size suffice for capturing *all* behaviors of the CFM.

2 Directions of the work

The core of the project is to transfer classical methods from software model checking to the framework of CFM (and MSC). A first step is the implementation of a CEGAR based approach for CFM model checking and its implementation in a simple software tool [LGHS08]. Counter-Example Guided Abstraction Refinement (CEGAR, [CGJ⁺00]) is the cornerstone of most current real world implementations, and, in our case, would act as the first touchstone for applying classical model checking methods to the CFM case in comparison to the established techniques.

In order to apply symbolic methods to existing implementations of communicating processes in the wild, one has to extract a (symbolic) model that represents the behaviour of the system. Further, this step is also inevitable to derive a sound basis for specifying properties that are to be verified against the “standard” behaviour of the system. Unfortunately, most TCP/IP-based application protocols in use are defined by natural language RFCs, accompanied by reference implementations with test cases, and *not* with the help of a formal underlying model. Therefore, the formal model of the “correct” protocol has to be derived by testing the existing implementations first.

The nature of these socket-based, distributed processes demands a new view onto CFM and their possible enhancements, because a formal model needs to be able to cope with the *dynamic* generation of connections, the interrelation of different independent machines that support a notion of *local* concurrency as well as the dynamic generation of communication channels and of subthreads (as common in the implementation of servers via forking, threading, and thread-pools [Ste04]) as well as distinct ways to manage message queues (e.g., the famous TCP half-close [Ste95, sect.18.5] is also part of the socket API and needs a counterpart in CFM modelling).

The inclusion of MSC is a double-edged sword: first, this introduces a totally different point of view regarding the underlying semantics (cf. partial-order semantics vs. operational semantics) and the applied methodologies; due to their purely representational nature, MSC are a high-level representation, far away from the concrete source-code level; however, they allow a more nuanced modelling of certain special cases (e.g., regarding the reordering of messages in the queue) and are close to the way a designer would specify a protocol. Contrasting an MSC and a CFM approach would allow to establish best practices to include different model checking strategies into industry tools that support the protocol engineer.

To conclude, this field combines a large variety of different approaches and their interplay: on the theoretical level, the focus will reside on CFM, MSC, and new models (based on communicating pushdown automata [BMOT05] or (flat) counter-machines [LS06], etc.) as well as applying formal methods (e.g. Abstract Interpretation [CC77], CEGAR [CGJ+00], and Predicate Abstraction [Das03]) on top of these; utilizing these ideas in the concrete verification of distributed, socket-based, communicating processes (either given as legacy code or as formal (diagrammatic) representation) will round off the practical side of our project.

Final Remarks

The preceding presentation puts our project up to discussion; consequently, we are open to any feedback. Please feel free to contact the author(s).

References

- [AABJ04] Parosh Aziz Abdulla, Aurore Annichini, Ahmed Bouajjani, and Bengt Jonsson. Using Forward Reachability Analysis for Verification of Lossy Channel Systems. *The Journal of Formal Methods in System Design*, 24:25–43, 2004.
- [ABS01] Aurore Annichini, Ahmed Bouajjani, and Mihaela Sighireanu. TRex: A Tool for Reachability Analysis of Complex Systems. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 368–372. Springer, 2001.
- [Bel07] Giampaolo Bella. *Formal Correctness of Security Protocols*. Information Security and Cryptography – Texts and Monographs. Springer, 2007.

- [BGWW97] Bernard Boigelot, Patrice Godefroid, Bernard Willems, and Pierre Wolper. The Power of QDDs. In Pascal Van Hentenryck, editor, *SAS*, volume 1302 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 1997.
- [BH99] Ahmed Bouajjani and Peter Habermehl. Symbolic Reachability Analysis of FIFO-Channel Systems with Nonregular Sets of Configurations. *Theor. Comput. Sci.*, 221(1-2):211–250, 1999.
- [BMOT05] Ahmed Bouajjani, Markus Müller-Olm, and Tayssir Touili. Regular Symbolic Analysis of Dynamic Networks of Pushdown Systems. In *CONCUR*, pages 473–487, 2005.
- [BZ83] Daniel Brand and Pitro Zafropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.
- [CC77] Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL – Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, 1977.
- [CGJ⁺00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-Guided Abstraction Refinement. In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.
- [Das03] Satyaki Das. *Predicate Abstraction*. PhD thesis, Department of Electrical Engineering, Stanford University, 2003.
- [CMGM06] Pedro de la Cámara, María del Mar Gallardo, and Pedro Merino. Abstract Matching for Software Model Checking. In Antti Valmari, editor, *SPIN*, volume 3925 of *Lecture Notes in Computer Science*, pages 182–200. Springer, 2006.
- [CMGM07] Pedro de la Cámara, María del Mar Gallardo, and Pedro Merino. Model Extraction for ARINC 653 Based Avionics Software. In Dragan Bosnacki and Stefan Edelkamp, editors, *SPIN*, volume 4595 of *Lecture Notes in Computer Science*, pages 243–262. Springer, 2007.
- [CMGMS05] Pedro de la Cámara, María del Mar Gallardo, Pedro Merino, and David Sanan. Model Checking Software with Well-Defined APIs: the Socket Case. In *FMICS '05: Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, pages 17–26, New York, NY, USA, 2005. ACM.
- [FIS03] A. Finkel, S. Purushothaman Iyer, and G. Sutre. Well-Abstracted Transition Systems: Application to FIFO Automata. *Information and Computation*, 181(1):1–31, 2003.
- [GKM07] Blaise Genest, Dietrich Kuske, and Anca Muscholl. On communicating automata with bounded channels. *Fundamenta Informaticae*, XX:1–21, 2007.
- [GMLS07] Hubert Garavel, Radu Mateescu, Frédéric Lang, and Wendelin Serwe. CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes. In Werner Damm and Holger Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 158–163. Springer, 2007.
- [GMP03] Blaise Genest, Anca Muscholl, and Doron Peled. Message Sequence Charts. In *Lectures on Concurrency and Petri Nets: Advances in Petri nets (Tutorial volume from the 4th Advanced Course on Petri Nets, ACPN 2003)*, number 3098 in LNCS, pages 537–558. Springer, 2003.
- [Hol01] Gerard J. Holzmann. From Code to Models. In *ACSD*, pages 3–10. IEEE Computer Society, 2001.
- [LGHS08] Tristan Le Gall, Alexander Heußner, and Grégoire Sutre. CEGAR-based Model-Checking for Communicating FIFO Machines. (in preparation), 2008.

- [LGJ07] Tristan Le Gall and Bertrand Jeannot. Analysis of Communicating Infinite State Machines using Lattice Automata. Technical Report 1839, IRISA, March 2007.
- [LS06] Jérôme Leroux and Grégoire Sutre. Flat Counter Automata almost everywhere! In Parosh Aziz Abdulla, Ahmed Bouajjani, and Markus Müller-Olm, editors, *Software Verification: Infinite-State Model Checking and Static Program Analysis*, volume 06081 of *LNCS*, pages 489–503. Springer, 2006.
- [ME04] Madanlal Musuvathi and Dawson R. Engler. Model Checking Large Network Protocol Implementations. In *Proceedings of the First Symposium on Networked System Design and Implementation*, pages 155–168. USENIX Assoc., 2004.
- [Ste95] W. Richard Stevens. *TCP/IP Illustrated, Volume 1; The Protocols*. Addison Wesley, Reading, 1995.
- [Ste04] W. Richard Stevens. *UNIX Network Programming Vol 1: Networking APIs – The Sockets Networking API*, volume 1 of 3. Prentice Hall, third edition, 2004.