Proceedings of the
1st Workshop on Artificial Intelligence and Formal Verification, Logics, Automata and Synthesis (OVERLAY),
Rende, Italy, November 19–20, 2019

Towards the Automated Verification of Publish/Subscribe Networks

Giorgio Delzanno[1]

[1]DIBRIS, University of Genova
[1]giorgio.delzanno@unige.it

Abstract

We present a formal model of publish/subscribe network architectures in which a central communication broker is in charge of distributing messages to clients subscribed to certain topics. We consider different semantics for the internal structured of the server and for the notification phase. We discuss applicability of an SMT-based infinite-state model checker to the proposed model and decidability results for abstractions obtained hiding the internal structure of a server.

1 Introduction

Protocols designed to operate in distributed systems are often defined for an arbitrary number of components and for asynchronous communication. Formal specification languages like Petri nets and automata are often used to model skeletons of this kind of systems. The coverability decision problem [1] is typically used to formulate reachability of bad configurations independently from the number of components of a system. To express safety properties of distributed systems we can lift the coverability decision problem, in which the initial configuration is fixed a priory, to a more general formulation existentially quantified over an infinite set of initial configurations [6, 7]. The existentially quantified coverability problem has been considered in [17, 18, 19, 5, 4] in order to reason on parameterized broadcast protocols. Falsification of this decision problem provides a characterization of initial configurations from which it is possible to reach a bad configuration, e.g., an anomaly in the protocol. Existentially quantified coverability problem is undecidable for systems with a static communication topology and atomic broadcast communication [1, 14, 17, 18, 6].

In this work we focus our attention on the application of the SMT-based Infinite-state Model Checker Cubicle [10] to formally specify and validate distributed protocols that combine asynchronous communication and synchronous operations on local data structures. Cubicle provides a specification language that combines both local and global update rules. The combination is particularly useful when dealing with specification of the internal behavior of protocols designed for client-server architectures. In the talk we will first present a formal model that can be applied to Pub/Sub protocols such as Redis, MQTT, etc. The model is based on a transition system in which clients are specified via labelled automata and the behaviour of the server is described via a global subscriber list and a finite set of worker threads. Spawning of worker threads accessing shared data is modelled using snapshot synchronisation [24] as in copy-on-write data structures [30]. This strategy consists in creating snapshots of shared data that are confined in the local data of worker threads. In our setting the shared data consists of the current subscribers list. Copy-on-write data structures [30] are a possible implementation of this synchronisation techniques that can be used to avoid race conditions on shared data structures. Copy-on-write data structures are particularly useful when book-keeping session data in a distributed application. Indeed they automatically produce a snapshot of the share data structures (lists, sets, etc) when combined with iterators (i.e. to scan a set/list) . The semantics of copy-on-write concurrent data structures typically provide a mechanism to generate snapshots on demand (e.g. when an iterator needs to scan the structure) producing an exception (that will be handled by the user) in case of simultaneous read and write access from multiple threads. In our model we will not model exception handling and assume that snapshots are generated and confined in a worker thread when necessary. Extending the semantics towards a more complex modeling of copy-on-write is an interesting direction for refining the model. The use of worker threads allows us to model the publishing of a message asynchronously w.r.t. to other operations executed on the server (e.g. registration of new clients, etc). Under the considered semantics for copy-on-write data structures, we will then show that computations in the resulting model can be transformed into round-based executions that are simpler to analyse (they reduce the number of possible interleavings) with respect to a more liberal interleaving model.

2 How to Formally Model Publish/Subscribe Networks

We use Id to denote a denumerable set of identifiers of client instances. Furthermore, we define Q as the finite set of client state labels, A as the finite set of action labels, and M as the finite set of message labels. For brevity, we assume that action labels have one of the following form: *local*, that denotes a local transition, *subscribe*, that denotes a subscription request, *unsubscribe*, that denotes an unsubscription request, $publish(m)$ with $m \in M$, that denotes a publish request for message $m \in M$.

The above listed type of actions are strictly related to the communication model typical of publish/subscribe architecture based on a client-server architecture in which every message is delivered to all subscribers via a central server (or, more in general, via a cluster/federation of servers). A client specification P is a tuple $\langle Q, q_0, R\rangle$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, and $R \subseteq Q \times A \times Q$ defines state transitions induced by action labels. In other words a client specification can be viewed as a finite state automata with labelled transitions that statically define its behaviour.

Client Configuration

A client configuration is a tuple $\langle i, s, b, f\rangle$, where $i \in Id$ is the client identifier generated after a connection request, $s \in Q$ is the current client state, $b \in 2^M$ is the set of messages received so far, and $f \in \{\top, \bot\}$ is a flag that defines the connection status of the client with respect to the global network, namely $\top$ corresponds to the normal operating status, whereas $\bot$ corresponds to a disconnection event. We assume that disconnected clients cannot roll back to a normal status, i.e., when they restart they will be assigned a new identifier, their internal state being completely reset. The client specification $\langle Q, q_0, R\rangle$ can naturally be extended with enabling conditions for transitions in R based on the presence of certain messages in the current message list (e.g. $\langle q_1, local, q_2\rangle$ only if message $m_1, \ldots, m_r$ have already been received). We will not discuss this extension in this paper.

Server Configuration

For a fixed $n \geq 1$, a server configuration is defined by a tuple $\langle L, W_1, \ldots, W_n\rangle$, where $L \in 2^{Id}$ is a finite set of identifers that represents the list of subscribed clients and, for each $i : 1, \ldots, n$ with $n > 1$, W_i represents the current state of the i-th worker thread used by the server to deliver messages. More specifically, W_i can be either $\bot$ or $\langle m_i, R_i\rangle$ with $m_i \in M$ and $R_i \subseteq 2^{Id}$. In the former case it denotes an idle thread, whereas in the latter it denotes initialized threads in which R_i is the current snapshot of the subscription list, and m_i is the message to be delivered. The use of a snapshot R of the current subscription list of the server is based on implementations of read/write operations on shared data structures based on snapshots/copy-on-write data structutes used in concurrent programming to avoid race conditions. The initialisation of a worker thread with the snapshot of the subscriber list allows the server to proceed asynchronously with the delivery of a message to all subscribers. In our model we consider a fairly realistic scenario in which the server employs a fixed number or worker threads for this kind of task. Furthermore, we model the semantics of a publish message asynchronously: the server first spawns (when available) a new task with a copy of the current subscriber list. The worker can then deliver the message to the active clients. The initial server configuration is the tuple $\langle \emptyset, I_1, \ldots, I_n \rangle$ where $I_i = \bot$ for $i : 1, \ldots, n$.

Global Configuration

A global configuration is defined by a tuple $\langle S, C \rangle$, where $S = \langle L, W_1, \ldots, W_n \rangle$ is a server configuration, and $C = \{c_1, \ldots, c_k\}$ is a finite set of client configurations such that $c_j = \langle i_j, s_j, b_j, f_j \rangle$ for $j : 1, \ldots, k$. We use N to denote the set of network configurations.

Pub/Sub Network

For fixed sets A, Q, M, and given a client specification $P = \langle Q, q_0, R \rangle$, and $n > 1$, a Pub/Sub Network PS is defined via a transitions system defined through a binary relation over network configurations. More precisely, the relation $\rightarrow \subseteq N \times N$ is defined as the least relation satisfying one of the conditions listed below. We show next some example of transitions such as *Publish* and *Notify*.

- Publish Transition:

$$Publish \quad \langle S, \{\langle i, s, b, f \rangle\} \cup C \rangle \rightarrow \langle S', \{\langle i, s', b, f \rangle\} \cup C \rangle$$

  under the assumptions: $f = \top$, $\langle s, publish(m), s' \rangle \in R$, $S = \langle L, W_1, \ldots, W_n \rangle$, $q \in \{1, \ldots, n\}$, $W_q = \bot$, $S' = \langle L, W_1, \ldots, W'_q, \ldots, W_n \rangle$, and $W'_q = \langle m, L \rangle$, With this rule a client instance sends a publish request to the server. The server acknowledges the request passing the message to an idle worker thread together with a snapshot of the current subscriber list L. The client updates its local state according to R.

- Notify Transition:

$$Notify \quad \langle S, C \rangle \rightarrow \langle S', C' \rangle$$

  under the following assumptions $C = \{c_1, \ldots, c_k\}$, $c_i = \langle id_i, s_i, b_i, f_i \rangle$ for all $i \in \{1, \ldots, k\}$, $S = \langle L_1, W_1, \ldots, W_n \rangle$, $W_q = \langle m, L_2 \rangle$ for some $q \in \{1, \ldots, n\}$, $S' = \langle L_1, W_1, \ldots, W'_q, \ldots, W_n \rangle$, $W'_q = \bot$, $C' = \{c'_1, \ldots, c'_k\}$ where for all $j : 1, \ldots, k$, $c_j = \langle id_j, s_j, b_j, f_j \rangle$, if $id_j \in L_2$ and $f_j = \top$, then $c'_j = \langle id_j, s_j, b'_j, f_j \rangle$ and $b'_j = b_j \cup \{m\}$, $c'_j = c_j$, otherwise. With this rule we model notification of message m via a global action performed by worker thread $W_q = \langle m, L_2 \rangle$ whose effect is to update the message list of each active client instance whose identifier is included in the list L_2. The remaining client instances (their identifier is not in the list or they are inactive) remain unchanged. After notification the worker thread resets its state to idle $\bot$ and returns available for distributing other published message.

The semantics of copy-on-write concurrent data structures in concurrent programming languages such as Java [30] is in general more complex than the model described above. Indeed, copy-on-write data structures typically provide a mechanism to generate snapshots on demand (e.g. when an iterator needs to scan the structure) producing an exception (that will be handled by the user) in case of simultaneous read and write access from multiple threads. In our semantics we do not model exception handling and focus instead on the confinement of snapshots of a shared data structure in worker threads.

3 Verification Problems and SMT Solvers

In our work we have applied the above described formal language to build a verifiable (parameterised) model that can be validated via the SMT-based Infinite-state Model Checker Cubicle. In this setting we use unbounded arrays to model arbitrary collections of publisher and subscriber processes as well as an unbounded shared data structure used as a communication media between processes. The evolution of the shared memory through the different rounds is modelled using a bi-dimensional unbounded arrays indexed both on round numbers and process identifiers. Each row of such a matrix can then be used to model the different phases of a protocol round, e.g., the formation of a subscriber group.

The resulting model can be validated through the SMT-based Infinite-state Model Checker Cubicle. Cubicle implements a symbolic backward reachability algorithm in which sets of configurations are represented via formulas in fragments of First Order Logic that combine Presburger Arithmetics and the Theory of Arrays [10, 29, 3, 23]. By construction, the Cubicle verification algorithm ensures that, upon termination, the resulting correctness proof is guaranteed to hold for any number of processes. In other words Cubicle can be applied as an automated engine for solving parameterised verification problems for the considered distributed protocol. In our experiments with the considered case-studies we successfully validated different versions of the Redis Pub/Sub protocol and identify corner cases for guards of transitions.

4 Decidability of Verification Problems

In [16] we have introduced a model of Publish Subscribe Networks inspired to Petri Nets in which the internal structure of a server is abstracted away and its behaviour is included in the transition system describing the interaction among clients. In this context we have considered different semantics for the notification phase in order to take into consideration exceptions due to node crashes. For the considered model, decidabilty of the coverability problem can be obtained via the application of Higman's and Dickson's lemmas and the theory of well-structured transition systems. The results are based on compositional properties of well-quasi orderings that can be applied in order to define verification algorithms based on symbolic backward reachability in which sets of Petri Net markings are finitely represented via constraint formulas. The extension of the above mentioned results to refined models of Pub/Sub Networks is left as future research.

References

[1] P. A. Abdulla and G. Delzanno. Parameterized verification. *STTT*, 18(5):469–473, 2016.

[2] P. A. Abdulla, G. Delzanno, N. Ben Henda, and A. Rezine. Monotonic abstraction: on efficient verification of parameterized systems. *Int. J. Found. Comput. Sci.*, 20(5):779–801, 2009.

[3] F. Alberti, S. Ghilardi, and N. Sharygina. A framework for the verification of parameterized infinite-state systems. *Fundam. Inform.*, 150(1):1–24, 2017.

[4] N. Bertrand, G. Delzanno, B. König, A. Sangnier, and J. Stückrath. On the decidability status of reachability and coverability in graph transformation systems. In *RTA'12*, volume 15 of *LIPIcs*, pages 101–116. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[5] N. Bertrand, P. Fournier, and A. Sangnier. Distributed local strategies in broadcast networks. In *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, pages 44–57, 2015.

[6] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. *Decidability of Parameterized Verification.* Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.

[7] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. Decidability in parameterized verification. *SIGACT News*, 47(2):53–64, 2016.

[8] B. Charron-Bost and A. Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.

[9] S. Conchon, G. Delzanno, and A. Ferrando. Parameterized verification of topology-sensitive distributed protocols goes declarative. In *Proceedings of NETYS 2018, to appear*, 2018.

[10] S. Conchon, A. Goel, S. Krstic, A. Mebsout, and F. Zaïdi. Cubicle: A parallel smt-based model checker for parameterized systems - tool paper. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, pages 718–724, 2012.

[11] S. Conchon, A. Goel, S. Krstic, A. Mebsout, and F. Zaïdi. Invariants for finite instances and beyond. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 61–68, 2013.

[12] H. Debrat and S. Merz. Verifying fault-tolerant distributed algorithms in the heard-of model. *Archive of Formal Proofs*, 2012.

[13] G. Delzanno. A logic-based approach to verify distributed protocols. In *Proceedings of the 31st Italian Conference on Computational Logic, Milano, Italy, June 20-22, 2016.*, pages 86–101, 2016.

[14] G. Delzanno. A unified view of parameterized verification of abstract models of broadcast communication. *STTT*, 18(5):475–493, 2016.

[15] G. Delzanno. Formal Verification of Internet of Things Protocols Invited talk at the 5th Workshop on Formal Reasoning in Distributed Algorithms (FRIDA), FLOC 2018 (draft available in the FLOC 2018 workshop web page).

[16] G. Delzanno. Parameterised Verification of Publish/Subscribe Networks with Exception Handling. RP 2019.

[17] G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, pages 313–327, 2010.

[18] G. Delzanno, A. Sangnier, and G. Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, pages 441–455, 2011.

[19] G. Delzanno, A. Sangnier, and G. Zavattaro. Verification of ad hoc networks with node and communication failures. In *FORTE/FMOODS'12*, volume 7273 of *LNCS*, pages 235–250. Springer, 2012.

[20] C. Dragoi, T. A. Henzinger, H. Veith, J. Widder, and D. Zufferey. A logic-based framework for verifying consensus algorithms. In *Verification, Model Checking, and Abstract Interpretation - 15th International Conference, VMCAI 2014, San Diego, CA, USA, January 19-21, 2014, Proceedings*, pages 161–181, 2014.

[21] C. Dragoi, T. A. Henzinger, and D. Zufferey. The need for language support for fault-tolerant distributed systems. In *1st Summit on Advances in Programming Languages, SNAPL 2015, May 3-6, 2015, Asilomar, California, USA*, pages 90–102, 2015.

[22] C. Dragoi, T. A. Henzinger, and D. Zufferey. Psync: a partially synchronous language for fault-tolerant distributed algorithms. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 400–415, 2016.

[23] S. Ghilardi and S. Ranise. Backward reachability of array-based systems by SMT solving: Termination and invariant synthesis. *Logical Methods in Computer Science*, 6(4), 2010.

[24] M. Herlihy, N. Shavit. The art of multiprocessor programming. Morgan Kaufmann 2008.

[25] A. Mebsout. *Inférence d'invariants pour le model checking de systèmes paramétrés. (Invariants inference for model checking of parameterized systems)*. PhD thesis, University of Paris-Sud, Orsay, France, 2014.

[26] T. Tsuchiya and A. Schiper. Verification of consensus algorithms using satisfiability solving. *Distributed Computing*, 23(5-6):341–358, 2011.

[27] `http://alt-ergo.lri.fr`.

[28] `http://functory.lri.fr/`.

[29] `http://users.mat.unimi.it/users/ghilardi/mcmt/`.

[30] `https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CopyOnWriteArraySet.html` .