

Towards Fine-grained Automated Verification of Publish-Subscribe Architectures

Luciano Baresi, Carlo Ghezzi, and Luca Mottola

Dipartimento di Elettronica ed Informazione—Politecnico di Milano
{baresi, ghezzi, mottola}@elet.polimi.it

Abstract. The design and validation of distributed applications built on top of Publish-Subscribe infrastructures remain an open problem. Previous efforts adopted finite automata to specify the components' behavior, and model checking to verify global properties. However, existing proposals are far from being applicable in real contexts, as strong simplifications are needed on the underlying Publish-Subscribe infrastructure to make automatic verification feasible.

To face this challenge, we propose a novel approach that embeds the asynchronous communication mechanisms of Publish-Subscribe infrastructures *within* the model checker. This way, Publish-Subscribe primitives become available to the specification of application components as additional, domain-specific, constructs of the modeling language. With this approach, one can develop a fine-grained model of the Publish-Subscribe infrastructure without incurring in state space explosion problems, thus enabling the automated verification of application components on top of realistic communication infrastructures.

1 Introduction

The Publish-Subscribe interaction paradigm is rapidly emerging as an appealing solution to the needs of applications designed for highly-dynamic environments. Using this paradigm, application components *subscribe* to event patterns and get *notified* when other components *publish* events matching their subscriptions. Its asynchronous, implicit and multi-point communication style is particularly amenable to those scenarios where application components can be added or removed unpredictably, and the communication must be decoupled both in time and in space [1]. Because of this flexibility, Publish-Subscribe infrastructures have been developed for a wide range of application scenarios, from wide-area notification services to wireless sensor networks.

However, the high degree of decoupling brings also several drawbacks. In particular, verifying how a federation of independently-written software components interconnected in such a loosely-coupled manner is often difficult because of the absence of a precise specification of the behavior of the communication infrastructure. Model checking has been proposed as a possible solution, but existing works do not propose a precise characterisation of the different guarantees the underlying Publish-Subscribe infrastructure can provide. For instance, different message delivery policies, reliability guarantees or concurrency models can easily change the final outcome of the verification effort.

The wide spectrum of deployment scenarios, and the consequent vast range of available systems, makes the aforementioned characterization non-trivial. In addition, modeling these features using the primitives of existing model checkers inevitably results in state space explosion problems, thus ultimately hindering the verification effort.

In this paper we argue that a fine-grained Publish-Subscribe model requires a different approach to the problem. We propose to augment an existing model-checker with domain-specific constructs able to expose Publish-Subscribe primitives as constructs of the modeling language. This way, the mechanisms needed to implement the Publish-Subscribe interaction style can be embedded *within* the model-checker, and one can achieve a fine-grained characterization of the different guarantees of the Publish-Subscribe system without incurring in state space explosion problems.

The rest of the paper is structured as follows. Section 2 briefly surveys the existing approaches, and highlights how they miss important characteristics of existing Publish-Subscribe infrastructures. Section 3 proposes our solution and reports on some initial results demonstrating how our approach better scales to complex systems than existing ones. Finally, Section 4 concludes the paper with an outlook on our current and future research plans.

2 Modeling Publish-Subscribe Systems

The identification of the different guarantees provided by Publish-Subscribe infrastructures is a challenging task. This becomes even harder when we consider those characteristics that would impact the verification of applications running on top. To this end, Table 1 summarizes a set of QoS dimensions provided by existing Publish-Subscribe infrastructures that could affect the outcome of the verification process. We claim that the majority of available systems can be precisely classified along these different dimensions. However, existing proposals for automated verification of Publish-Subscribe infrastructures fail to capture many of these different characteristics.

The work in [2, 3] is limited to the CORBA Communication Model (CCM) as middleware. Similarly, [4] concentrates on the addition of a Publish-Subscribe notification service to an existing groupware protocol. On the other hand, the work in [5] develops a compositional reasoning based on an assume-guarantee methodology. The technique is shown to be applicable to particular instances of Publish-Subscribe middleware. In all these cases, the proposed solution addresses just a very narrow scenario, i.e., a particular instance of Publish-Subscribe system.

Garlan et al. [6] provide a set of pluggable modules that allow a modeler to choose one possible instance out of a set of possible choices. However, the available models are far from fully capturing the different characteristics of existing Publish-Subscribe systems. For instance, application components cannot change their interests (i.e., subscriptions) at run-time, and the event dispatching policy is only characterized in terms of delivery policy (*asynchronous*, *synchronous*, *immediate* or *delayed*). The approach is improved in [7] by adding more expressive events, dynamic delivery policies and dynamic event-method bindings. However, this proposal only deals with the specification of different delivery policies depending on the state of the overall model, and still does not capture finer-grained guarantees such as real-time constraints.

QoS Class	Possible Choices	Description
<i>Message Reliability</i>	Absent	Notifications can be lost.
	Present	Notifications are guaranteed to eventually arrive at the interested subscribers.
<i>Message Ordering</i>	None	Notifications are delivered in random order.
	Pair-wise FIFO	Notifications are delivered to a given subscriber in FIFO order with respect to publish operations from the same publisher.
	System-wide FIFO	Notifications are delivered to subscribers in the same order as publish operations, regardless of the component that published the message.
	Causal Order	Causally related notifications are delivered according to the causality chain among them.
	Total Order	All components subscribed to the same events are notified in the same order (which is not necessarily the order in which these events are published).
<i>Filtering</i>	Precise	Notifications are only delivered for subscribed events.
	Approximate	Components can be notified on events for which they are not subscribed (false positives), or miss events in which they are interested. (Notice how approximate filtering is deterministic, while reliability problems are in general unpredictable.)
<i>Real-Time Guarantees</i>	None	Notifications are delivered without time guarantees.
	Soft RT	On the average, notification are delivered in T time units after the publish operation.
	Hard RT	Notifications are guaranteed to be delivered within T time units after the publish operation.
<i>Subscription Propagation Delay</i>	Absent	Subscriptions are immediately active and deliver event notifications.
	Present	Subscriptions start to deliver event notifications after a random time.
<i>Repliable Messages</i>	Absent	Subscriptions set up to convey replies travel independently of the original notification. If subscriptions are delayed, the deliver of the reply is not guaranteed.
	Present	Subscriptions used to convey replies travel with the originating message. Therefore, they are guaranteed to be active at the time the reply is published.
<i>Message Priorities</i>	Absent	All notifications are treated in the same way.
	Present	Notifications are delivered according to specific priorities, no mechanism is used to prevent starvation of messages.
	Present w/ Queue Scrunching	Notifications are delivered according to specific priorities, queue scrunching dynamically raises the priority of messages that waited too long in their original priority queue. This way, each message is eventually delivered.
<i>Queue Drop Policy</i>	None	Queues are of infinite length.
	Tail Drop	Given queues of length L , messages exceeding this threshold are dropped upon arrival.
	Priority Drop	Given queues of length L , messages are dropped starting from lower priority messages up to higher priority messages.

Table 1. Publish-Subscribe QoS dimensions.

Finally, the work in [8] characterizes the Publish-Subscribe infrastructure in terms of reliability, event delivery order (the same as publications, the same as publications but only referring to the same publisher, or none), and subscription propagation delay. Still, it does not consider several of the dimensions listed in Table 1.

3 Our Proposal

The definition of all the mechanisms described in Table 1 is clearly unfeasible if one keeps the traditional approach of expressing both the application components and the communication infrastructure in terms of the primitives of the model checker. Based on this, we propose a novel approach to augment an existing model checker with Publish-Subscribe-style constructs. This way, we build the communication infrastructure *inside* the model checker, thus avoiding the aforementioned performance problems.

Our approach leverages off the simplicity of the Publish-Subscribe APIs (composed of the basic operations `publish(Event)` and `subscribe(EventPattern)`),

Scenario	Bogor with embedded Publish-Subscribe		SPIN - Promela	
	Memory	Time	Memory	Time
Causal2Publish	32.8 Mb	103 sec	298.3 Mb	>15 min
Causal5Publish	45.6 Mb	132 sec	589.6 Mb	>1 hour
Causal7Publish	52.3 Mb	158 sec	OutOfMemory	NotConcluded
Causal10Publish	61.1 Mb	189 sec	OutOfMemory	NotConcluded
Priorities2Publish	18.3 Mb	47 sec	192 Mb	>10 min
Priorities5Publish	26.9 Mb	103 sec	471.2 Mb	>30 min
Priorities7Publish	37.9 Mb	134 sec	OutOfMemory	NotConcluded
Priorities10Publish	49.1 Mb	163 sec	OutOfMemory	NotConcluded

Table 2. Comparing our approach with [8].

and makes them available as additional constructs of the input language of the model checker.¹ Before performing the actual verification, the user binds this general Publish-Subscribe API to a particular combination of the different guarantees highlighted in Table 1, thus “instantiating” a particular communication infrastructure on top of which the application model is run.

To implement our approach, we are currently embedding a Publish-Subscribe communication mechanisms —with the various guarantees highlighted in Table 1— within the model checker Bogor [9]. Its open architecture makes it easy to add domain-specific mechanisms to the model checker. To check our additions, we devised a wide range of test cases. Every test is represented by a set of Bogor processes expressed in BIR (Bogor Intermediate Language), which make use of the aforementioned Publish-Subscribe API as any other BIR construct.

The solution we propose impacts on two orthogonal aspects. Firstly, it enables automated verification of application components on top of realistic communication infrastructures. This way, the gap between the system model in the early design phases and the actual implementation can be narrowed down, and potential problems caught in advance. Secondly, it eases the translation of the behavior of application components —usually expressed in a given specification formalism— into the input language of the model checker, since the application components and the model checker rely on the same communication primitives.

3.1 Early Results

To substantiate our claims, we report some initial results we gathered by comparing our approach with the solution in [8], which uses SPIN and Promela. We designed a set of possible interactions among five different processes with the only goal of making processes coordinate by exchanging messages. We characterized the different scenarios by means of the number of publish operations to be performed —on a per-process basis— during each run of the test application. On the average, half of the processes are subscribed to published events and receive the corresponding notifications. The properties we are interested in are simple assertions, whose only goal is to make sure that messages are delivered according to the chosen policy (i.e., in causal order or according to the respective priorities). In a sense, these assertions verify that the implemented mechanisms are semantically correct.

¹ Notice how the Publish-Subscribe APIs we consider explicitly deal with subscriptions to general patterns of events, therefore overcoming the limitations of existing proposals (e.g., [6]).

Table 2 illustrates the performance of the two approaches, both in terms of memory consumption and time needed to complete the verification process. The experiments were executed on a Pentium 4 with 1 Gb RAM. Our approach outperforms the one based on SPIN in all cases. When the number of publish operations increases, our solution allows the verification effort to terminate where SPIN would run out of memory. This clearly highlights how the requirement of realistically modeling the communication infrastructure cannot be addressed by using only the primitives provided by conventional model checkers.

4 Conclusions and Future Work

This paper presents a novel approach to the automated verification of applications built on top of fine-grained and realistic models of Publish-Subscribe architectures. We argue that such a level of detail cannot be achieved by means of conventional model checkers. Our proposal flips the problem and augments the input language of an existing model checker with the primitives of Publish-Subscribe communication infrastructures. The first results summarized in this paper are encouraging and motivate further work.

We plan to conclude the implementation in Bogor of the different guarantees illustrated in Table 1, and further evaluate the effectiveness of our approach with meaningful test cases. However, our ultimate objective is the development of a tool to enable automated verification of applications built on top of Publish-Subscribe systems.

References

1. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. *ACM Comput. Surv.* **35**(2) (2003)
2. Deng, X., Dwyer, M.B., Hatcliff, J., Jung, G.: Model-checking middleware-based event-driven real-time embedded software. In: *Proc. of the 1st Int. Symposium on Formal Methods for Components and Objects.* (2002)
3. Hatcliff, J., Deng, X., Dwyer, M.B., Jung, G., Ranganath, V.: Cadena: an integrated development, analysis, and verification environment for component-based systems. In: *Proc. of the 25th Int. Conf. on Software Engineering (ICSE03).* (2003)
4. Beek, M.H., Massink, M., Latella, D., Gnesi, S., Forghieri, A., Sebastianis, M.: A case study on the automated verification of groupware protocols. In: *Proc. of the 27th Int. Conf. on Software engineering (ICSE05).* (2005)
5. Caporuscio, M., Inverardi, P., Pelliccione, P.: Compositional verification of middleware-based software architecture descriptions. In: *Proc. of the 19th Int. Conf. on Software engineering (ICSE04).* (2004)
6. Garlan, D., Khersonsky, S.: Model checking implicit-invocation systems. In: *Proc. of the 10th Int. Workshop on Software Specification and Design.* (2000)
7. Bradbury, J.S., Dingel, J.: Evaluating and improving the automatic analysis of implicit invocation systems. In: *Proc. of the 9th European software engineering Conf.* (2003)
8. Zanolin, L., Ghezzi, C., Baresi, L.: An approach to model and validate publish/subscribe architectures. In: *Proc. of the SAVCBS'03 Workshop.* (2003)
9. Robby, Dwyer, M.B., Hatcliff, J.: Bogor: an extensible and highly-modular software model checking framework. In: *Proc. of the 9th European software engineering Conf.* (2003)