Parameterised Verification of Publish/Subscribe Networks with Exception Handling

Giorgio Delzanno(✉)

DIBRIS, University of Genova, Genoa, Italy
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 notification phase in order to take into consideration exceptions due to node crashes. For the considered model, we study decidability of verification problems formulated in terms of coverability, a non trivial class of reachability problems well-suited to validate properties of parameterised systems.

1 Introduction

Publish/subscribe protocols such as MQTT are widely used for interconnecting heterogeneous collections of network services and devices, e.g., in Internet of Things applications. In this paper we present a new formal model of publish/-subscribe protocols such as MQTT in which a broker is in charge of distributing messages to clients subscribed to certain topics. In this setting we use a transition system parametric on the specification of individual nodes to provide an operational semantics to basic operations such as (un)subscription and push notifications. Node crashes and connection failures are modelled via state information included in the representation of individual nodes. We then provide a formal specification of different implementations of the broker internal structure. The semantics is inspired to a working prototype of pub/sub broker that we implemented in Java using RMI (Remote Method Invocation) communication. More in detail, we consider two scenarios in which to model the delivery of a published message m to subscribers of a given topic t.

In the first scenario the broker acknowledges the request and, inside a synchronisation block, forwards the message the other clients subscribed to topic t. Communication failures are captured locally via a try-catch statement as in Fig. 1. In the considered example the broker stores client communication data in a shared Map protected by a synchronization region (the Map *topicRelation*). In RMI the communication data are encapsulated in stub objects that act as interfaces when invoking remote methods on each client (in our example the method

E. Filiot et al. (Eds.): RP 2019, LNCS 11674, pp. 107–120, 2019.
https://doi.org/10.1007/978-3-030-30806-3_9

```
boolean publish(String topic, String news, String sender)
throws Exception {
 ClientInterface client;
  synchronized(topicRelation) {
    Map<Integer, ClientInterface>
        subscriberList = topicRelation.get(topic);
    synchronized(subscriberList) {
     Iterator<Map.Entry<Integer, ClientInterface>> entries =
            subscriberList.entrySet().iterator();
      while (entries.hasNext()) {
        Map.Entry<Integer, ClientInterface>
            entry = entries.next();
        client = entry.getValue();
        try {   stub.send(topic,sender,news); }
        catch (RemoteException e) {
                System.out.println("Notification error);
        }
      }
    }
 }
 return true;
}
```

Fig. 1. Broker in Java: first scenario.

send). Remote methods throw RemoteExceptions. In this example, since exceptions are handled locally, notification always reach all connected nodes leaving the state of all other nodes unchanged.

In the second scenario we consider an implementation in which an exception generated during a push notification sent to a certain client is propagated to the caller of the *publish* method. Going back to our Java example, this scenario corresponds to the implementation of the *publish* method with synchronized regions, an iterator over a Map, and a remote callback on a client stub (method *send*) in Fig. 2. In this scenario we assume that every invocation to the *publish* method is embedded into a try-catch statement, e.g., to propagate error notifications to the server or to modify the current list of active clients.

In the paper we give a formal account of the above mentioned scenarios by introducing a transition system modelling configurations with an arbitrary number of publishers and subscribers and a single broker. The behaviour of the broker is hard-wired in the semantics of the transition system. For the considered model, we focus our attention on decidability properties of verification problems formulated in terms of coverability, a non trivial class of reachability problems well-suited to validate properties of parameterised systems.

The reason why we consider parameterised formulations of verification problems is strictly related to the nature of distributed algorithms and protocols. Indeed, protocols designed to operate in distributed systems are often defined

```
boolean publish(String topic, String news, String sender)
    throws Exception {
 synchronized(topicRelation) {
     Map<Integer, InfoSub> subscriberList =
         topicRelation.get(topic);
     synchronized(subscriberList) {
         Iterator<Map.Entry<Integer, InfoSub>>
             entries = subscriberList.entrySet().iterator();
         while (entries.hasNext()) {
             Map.Entry<Integer, InfoSub>
                 entry = entries.next();
             entry.getValue().send(topic,sender,news);
         }
     }
 }
 return true;
}
```

Fig. 2. Broker in Java: second scenario.

for an arbitrary number of components. Formal specification languages like Petri nets and automata are often used to model skeletons of this kind of systems. In this setting the coverability decision problem [1] is typically used to formulate reachability of bad configurations independently from the number of components of a system. Furthermore, 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 formulation in which the initial configuration is picked up from an infinite set of initial configurations [3,4]. This formulation of the coverability problem has been considered in [5,7–12] in order to reason on Broadcast Protocols. Falsification of this decision problem provides a characterisation of initial configurations from which it is possible to reach a bad configuration.

Plan of the Paper. In Sect. 2 we introduce our formal model of Pub/Sub Networks, inspired to extensions of Petri nets with data with a first formulation of the notification phase. In Sect. 4 we study decidability properties for coverability in parameterised formulations of Pub/Sub Networks. In Sect. 5 we consider an extensions with retained messages inspired to the MQTT protocol. In Sect. 6 we introduce a variant of the notification phase in which we model exception handling using a global conditions on the operating status of client nodes and reconsider decidability properties for coverability in parameterised formulations of the proposed variant of Pub/Sub Networks. In Sect. 7 we address some conclusions, consider other extensions and proposed some open problems.

2 Formal Model of Pub/Sub Architectures

In this section we introduce a formal model that will help us to analyse the interactions between publishers and subscribers via a server with synchronized operations on internal data structures. In the rest of the paper we will use the following terminology. Multisets over elements $e_1, e_2, \ldots$ in a support set D will be indicated as $\{\!\{e_1, e_2, \ldots\}\!\}$, multiset union as $\oplus$, multiset difference as $\ominus$, $\sqsubseteq$, $\sqsubset$ as multiset inclusion, and $\in$ as membership. Furthermore, we will use the standard notation such as $\cup, \cap, \setminus, \subseteq, \subset$ and $\in$ for operations over sets. We will use 2^A to indicate the powerset of A. Finally, we will use $\langle e_1, \ldots, e_n \rangle$ to denote tuples of elements in D.

Topics, States, Messages and Actions

We define T to be a finite set of, fixed a priori, labels representing topics names. We define Q to be a finite set of labels of client states. Furthermore, we define M to be a finite set of message labels. Finally, we consider a finite set of action labels having the following form:

- *local*, that denotes a local transition,
- *subscribe*(s) for $s \subseteq T$ that denotes subscription to a subset of topics,
- *unsubscribe*(s) for $s \subseteq T$ that denotes unsubscription from a subset of topics,
- *publish*(m, t) with $m \in M$ and $t \in T$, that denotes publishing of message m on topic t.

The above listed type of actions are strictly related to the communication model typical of publish/subscribe architecture in which every message is delivered to all subscribers via a shared broker.

Client Specification

In the rest of the section we will first introduce the static specification of individual clients. The dynamic semantics of a client will be described only after having introduced the notion network configuration. In this setting we will consider systems composed by a single server and an arbitrary number of client instances, each one defined by the same client specification. A client configuration c is a tuple $\langle q, s, b, f \rangle$, where $q \in Q$ is the current client state, $s \in 2^T$ is the set of topics for which the client is a subscriber, $b \in 2^M$ 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.

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 $R \subseteq Q \times A \times 2^M \times Q$ defines state transitions induced by action labels. In other words a client specification can be viewed as a finite state automata with labelled transitions which statically defines its behavior. The tuple $\langle q, a, s, q' \rangle$ denotes a transition from q to q' associated to action a whose firing requires the presence of at least the set of messages s in the local message list. For instance, $\langle q_1, local, \{m, n\}, q_2 \rangle$ can be fired in $\langle q_1, s, b, f \rangle$ only if $\{m, n\} \subseteq b$.

We assume here that disconnected clients cannot roll back to a normal status, i.e., when they restart they will be assigned a new identity, their internal state being completely reset. In other words we will simulate restart by using client creation. The model can naturally be extended in order to consider a richer set of operations for the manipulation of local message sets (e.g. remove messages, reset buffers, etc.). For the sake of brevity, we will keep the model simple and discuss how these extension affect the decidability of coverability when necessary.

2.1 Pub/Sub Networks

We are ready now to define a model for Pub/Sub Networks. In this paper we will consider a single Pub/Sub broker. The client-server architecture of such a server will be implicitly defined via the semantics of *publish* operations.

A Pub/Sub Network S consists of fixed sets A, Q, M, and a client specification $P = \langle Q, q_0, R\rangle$. In this setting a network configuration is defined as a multiset $\gamma = \{\!\{c_1, \ldots, c_k\}\!\}$ of client configurations, i.e., $c_i = \langle q_i, s_i, b_i, f_i\rangle$ with $q_i \in Q$, $s_i \in 2^T$, $b \in 2^M$ and $f_i \in \{\bot, \top\}$ for $i : 1, \ldots, k$. We use N to denote the set of Network Configurations of finite but arbitrary size. The set N_0 of Initial Network Configurations is the subset of N is which client configurations are restricted to those with form $c_0 = \langle q_0, \emptyset, \emptyset, \top\rangle$.

Operational Semantics

The operational semantics of a Pub/Sub Network η is defined via a transitions system defined through a binary relation $\rightarrow$ over Network Configurations. To specify the semantics of this operation we first introduce some auxiliary definitions. Let $t \in T$ and γ be a configuration, $E_f(t, \gamma)$ is the multiset containing all client configurations of the form $\langle q, s, b, f\rangle$ occurring in γ such that $t \in s$. Furthermore, we use $Add_m(t, \gamma)$ to denote the multiset obtained from γ by adding message m in all local message sets of configurations of subscribers of topic t with flag f. More precisely, $Add_m(t, \{\!\{\}\!\}) = \{\!\{\}\!\}$, $Add_m(t, \{\!\{\langle q, s, b, \top\rangle\}\!\} \oplus \gamma) = \{\!\{\langle q, s, \{m\} \cup b, \top\rangle\}\!\} \oplus Add_m(t, \gamma)$, if $t \in s$; $Add_m(t, \{\!\{c\}\!\} \oplus \gamma) = \{\!\{c\}\!\} \oplus Add_m(t, \gamma)$ otherwise. We also use $Add_m(t, \gamma)$ to denote the multiset obtained from γ by adding message m in all local message sets of configurations of subscribers of topic t with flag f. The relation $\rightarrow \subseteq N \times N$ is the least relation satisfying one of the conditions listed below.

Local Operations. We first consider actions with local effect.

$$Local \qquad \{\!\{\langle q, s, b, \top\rangle\}\!\} \oplus C \;\rightarrow\; \{\!\{\langle q', s, b, \top\rangle\}\!\} \oplus C$$

under the assumption $\langle q, local, q'\rangle \in R$. With this rule a client instance updates its local state after firing a local action.

$$Subscription \qquad \{\!\{\langle q, s, b, \top\rangle\}\!\} \oplus C \;\rightarrow\; \{\!\{\langle q', s \cup s_1, b, \top\rangle\}\!\} \oplus C$$

under the assumption $\langle q, subscribe(s_1), q'\rangle \in R$. With this rule a client instance subscribes to the set of topics s_1.

$$Unsubscription \qquad \{\!\{\langle q, s, b, \top\rangle\}\!\} \oplus C \;\rightarrow\; \{\!\{\langle q', s \setminus s_1, b, \top\rangle\}\!\} \oplus C$$

under the assumption $\langle q, unsubscribe(s_1), q' \rangle \in R$. With this rule a client unsubscribes from the set of topics s_1.

$$Disconnection \quad \{\!\{\langle q, s, b, \top \rangle\}\!\} \oplus C \;\rightarrow\; \{\!\{\langle q, s, b, \bot \rangle\}\!\} \oplus C$$

With this rule we can non-deterministically turn an active client instance into a disconnected instance. This rule models a fairly realistic scenario in which the broker is equipped with a background service for sending heartbeat messages to each client in order to check their connection status.

Global Operations. We now turn our attention to global operations that model the publish action. As discussed in the introduction, we first consider a semantics based on a broadcast message embedded into synchronisation blocks in which possible errors during individual notifications are handled locally, e.g., using try-catch statements. The semantics of *publish* is defined as follows.

$$Publish \quad \{\!\{\langle q, s, m, \top \rangle\}\!\} \oplus \gamma \rightarrow \{\!\{\langle q', s, m, \top \rangle\}\!\} \oplus \gamma'$$

under the following assumptions:

- $\langle q, publish(m, t), q' \rangle \in R$,
- $\xi = E_\top(t, \gamma)$,
- $\mu = \gamma \ominus \xi$,
- $\gamma' = Add_m(t, \xi) \oplus \mu$.

With this rule a client sends a publish request for message m on topic t to the server. The server acknowledges the request and, inside a synchronisation block, forwards the message m to all other active clients subscribed (at least) to topic t. We assume that m is not sent to the sender client. Since disconnected clients are not selected in $E_\top(t, \gamma)$, the server forwards the message only to active clients. The state of all other clients (disconnected ones and clients that are not subscribed to topic t) remain unchanged. Since our semantics does not keep track of exact time information on node failures and message receptions, the message m is not added to disconnected nodes in order to avoid confusion when inspecting a final configuration.

Computations. A computation σ is a (possibly infinite) sequence of network configurations $\sigma = \gamma_0 \dots \gamma_i \dots$ such that $\gamma_i \rightarrow \gamma_{i+1}$ for $i \geq 0$. Using a standard notation, we will use $\rightarrow^*$ to denote the transitive closure of $\rightarrow$.

3 Example: Specification of an IoT System

Let us consider an example inspired to the standard workflow of an IoT application based on MQTT. MQTT is often used for both device discovery and data acquisition. In the discovery phase a subscriber registers to a topic exposing a list of available sensors. After receiving access details for a specific sensor, a subscriber can start listening to data coming from the sensor.

$$
\begin{aligned}
r_1 &= \langle init, subscribe(sensors), \emptyset, listen\rangle \\
r_2 &= \langle listen, local, \{s_i\}, aux_i\rangle, \\
r_3 &= \langle aux_i, unsubscribe(sensors), \emptyset, sub_i\rangle \\
r_4 &= \langle sub_i, subscribe(s_i), \emptyset, acquire_i\rangle \\
r_5 &= \langle acquire_i, \{d_{i1}, \ldots, d_{ik}\}, \emptyset, ok_i\rangle \\
r_6 &= \langle ok_i, unsubscribe(s_i), \emptyset, init\rangle \\
r_7 &= \langle ok_i, unsubscribe(s_i), \emptyset, end_i\rangle
\end{aligned}
$$

Fig. 3. IoT subscriber

Subscriber. The workflow of a subscriber can be described as follows. Consider the set of topics $T = \{sensors, s_1, \ldots, s_n\}$ and messages $M = \{s_1, \ldots, s_n\} \cup \bigcup_{i=1}^{n}\{d_{i1}, \ldots, d_{ik}\}$ in which $d_{i,j}$ represents a piece of data coming from sensor s_i. A subscriber can then be described by the specification in Fig. 3. In this model the subscriber first register on topic *sensors*. Then he waits for message s_i for some i and use the message label to register to topic s_i. After registration the subscriber waits for messages $d_{i1}, \ldots, d_{ik}$. Rule r_6 specifies the completion the reception phase, i.e., it simulates the reception of all data sent by sensor s_i. Notice that communication is asynchronous, i.e., the subscriber accumulates individual messages in its message set and then moves to state ok_i only when all messages have been received. The subscriber then unsubscribes from s_i and moves either to state *init* or to the halting state end_i.

Discovery Service. A discovery service is in charge of generating from time to time the list of available sensors on the *sensor* topic. This service can be described via the following specification.

$$\langle sinit, publish(s_i, sensors), send_i\rangle \ for\ i : 1, \ldots, n$$

Sensors. Each sensor s_i acts as a publisher that is in charge of sending data along the corresponding topic s_i. The publisher associate to sensor s_i can be described via the following model.

$$\langle init_i, publish(d_{ij}, s_i), init_i\rangle \ for\ i : 1, \ldots, n,\ j : 1, \ldots, k$$

In our model the broker is implicitly defined via the semantics of the *subscribe*, *unsubscribe*, and *publish* operations. In Fig. 4 we present an example of computation in the above defined Pub/Sub Network in which for clarity states are labeled with process indexes. Notice that messages associated along with a certain topic are delivered only to the current set of subscribed clients. For instance, in our example the client in state $init_2$ never receives message s_1 since it is not subscribed to topic *sensors* when the publisher sends the message. In other words clients do not read messages from a shared global memory. Subscriber groups are formed dynamically and messages are delivered to the current set of subscribers. In our example when sensor s_i is added to the public registry via

$$\langle\langle sinit, \emptyset, \emptyset, \top\rangle, \langle init_1, \emptyset, \emptyset, \top\rangle, \langle init_2, \emptyset, \emptyset, \top\rangle, \langle init_3, \emptyset, \emptyset, \top\rangle,$$
$$\langle init_4, \emptyset, \emptyset, \top\rangle\rangle \rightarrow$$
$$\langle\langle sinit, \emptyset, \emptyset, \top\rangle, \langle init_1, \emptyset, \emptyset, \top\rangle, \langle init_2, \emptyset, \emptyset, \top\rangle, \langle listen, \{sensors\}, \emptyset, \top\rangle,$$
$$\langle init_4, \emptyset, \emptyset, \top\rangle\rangle \rightarrow$$
$$\langle\langle sinit, \emptyset, \emptyset, \top\rangle, \langle init_1, \emptyset, \emptyset, \top\rangle, \langle init_2, \emptyset, \emptyset, \top\rangle, \langle listen, \{sensors\}, \{s_1\}, \top\rangle,$$
$$\langle init_4, \emptyset, \emptyset, \top\rangle\rangle \rightarrow$$
$$\langle\langle sinit, \emptyset, \emptyset, \top\rangle, \langle init_1, \emptyset, \emptyset, \top\rangle, \langle init_2, \emptyset, \emptyset, \top\rangle, \langle aux_1, \emptyset, \{s_1\}, \top\rangle,$$
$$\langle init_4, \emptyset, \emptyset, \top\rangle\rangle \rightarrow$$
$$\langle\langle sinit, \emptyset, \emptyset, \top\rangle, \langle init_1, \emptyset, \emptyset, \top\rangle, \langle init_2, \emptyset, \emptyset, \top\rangle, \langle sub_1, \{s_1\}, \{s_1\}, \top\rangle,$$
$$\langle init_4, \emptyset, \emptyset, \top\rangle\rangle \rightarrow$$
$$\langle\langle sinit, \emptyset, \emptyset, \top\rangle, \langle init_1, \emptyset, \emptyset, \top\rangle, \langle init_2, \emptyset, \emptyset, \top\rangle, \langle acquire_1, \{s_1\}, \{s_1\}, \top\rangle,$$
$$\langle init_4, \emptyset, \emptyset, \top\rangle\rangle \rightarrow$$
$$\langle\langle send_1, \emptyset, \emptyset, \top\rangle, \langle init_1, \emptyset, \emptyset, \top\rangle, \langle init_2, \emptyset, \emptyset, \top\rangle, \langle acquire_1, \{s_1\}, \{s_1, d_{1,3}\}, \top\rangle,$$
$$\langle listen, \{sensors\}, \{s_1\}, \top\rangle\rangle \rightarrow \ldots \rightarrow$$
$$\langle\langle send_1, \emptyset, \emptyset, \top\rangle, \langle init_1, \emptyset, \emptyset, \top\rangle, \langle init_2, \emptyset, \emptyset, \top\rangle, \langle ok_1, \{s_1\}, \{s_1, d_{1,1}, \ldots, d_{1,k}\}, \top\rangle,$$
$$\langle listen, \{sensors\}, \{s_1\}, \bot\rangle\rangle \rightarrow$$
$$\langle\langle send_1, \emptyset, \emptyset, \top\rangle, \langle init_1, \emptyset, \emptyset, \top\rangle, \langle init_2, \emptyset, \emptyset, \top\rangle, \langle end_1, \emptyset, \{s_1, d_{1,1}, \ldots, d_{1,k}\}, \top\rangle,$$
$$\langle listen, \{sensors\}, \{s_1\}, \bot\rangle\rangle$$

Fig. 4. Computations

the discovery service, each subscriber can receive its data and possibly reach its halting state. For instance, it should not be possible to reach a configuration in which a subscriber is in state end_i from initial configurations in which there are no discovery service nodes in state $init_i$. This kind of property should hold for any number of nodes. In the next section we will discuss how this kind of properties can be stated formally on parameterised families of transition systems of Pub/Sub Networks.

4 The Coverability Decision Problem

In this paper we will focus our attention on safety properties for Pub/Sub Networks with a finite but arbitrary number of clients (parameterised system) described via a decision problem called Coverability considered in other formal models of concurrent and distributed systems such as Petri nets, Broadcast Protocols, Lossy FIFO systems, and Ad Hoc Networks (see e.g. [2,14]).

Reachability Sets. In order to formally define the problem we introduce some auxiliary definitions. Given a set of configurations $C \subseteq N$, the Pre and $Post$ operators are defined as follows:

$$Post(C) = \{\gamma' | \exists \gamma \in C \ s.t. \ \gamma \rightarrow \gamma'\}$$
$$Pre(C) = \{\gamma | \exists \gamma' \in C \ s.t. \ \gamma \rightarrow \gamma'\}$$

$Post^*(C)$ [resp. $Pre^*(C)$] is defined as $\bigcup_{i=0}^{\infty} Post^i(C)$ [resp. $\bigcup_{i=0}^{\infty} Pre^i(C)$]. The set of configurations reachable from $C \subseteq N$ is defined as $Post^*(C)$. For instance,

$Post^*(N_0)$ is the set of configurations reachable from initial configurations of arbitrary size. Similarly, given a set of target configurations T, $Pre^*(T)$ is the set of predecessor configurations that can reach configurations in T after finitely many steps.

Coverability. The Coverability Decision problem is strictly related to the above mentioned correctness criterion. Let $\langle N, \leq \rangle$ be a total ordering on Network configurations. Furthermore, for a set of configurations S, let $uc_{\leq}(S) = \{\gamma' | \gamma \leq \gamma',\ \gamma \in S\}$.

Definition 1. *Let $\langle \eta, \rightarrow \rangle$ be a Pub/Sub Network defined over the sets A, M, Q and the client specification P, with an associated predecessor operator Pre, with an ordering $\leq$ on Network Configurations, and with a set N_0 of Initial Network Configuration. Given a finite set of configuration $F \subseteq N$, the Coverability Decision Problem consists in checking whether $N_0 \cap Pre^*(uc_{\leq}(F)) = \emptyset$ or, alternatively, $Post^*(N_0) \cap uc_{\leq}(F) = \emptyset$.*

The rationale behind this definition is as follows. Assume that $T = uc_{\leq}(F)$ represents a set of bad configurations of arbitrary size (e.g. violations of a given safety property) that can be finitely generated via the upward closure of F (if γ represents a violations, then any γ' larger than γ represents a violation). The condition $N_0 \cap Pre^*(T) = \emptyset$ [resp. $Post^*(N_0) \cap T = \emptyset$] holds if and only if there exist no finite computations that starting from some initial configuration (of any size) can reach a bad configuration in T.

4.1 Decision Procedure for Coverability in Pub/Sub Networks

In this section we will study instances of the coverability problem that can be applied to verify properties by considering both local states and received messages.

Definition 2. *Given two client configurations c_1, c_2, the ordering $\leq_c$ is defined as follows: $c_1 = \langle q_1, s_1, b_1, f_1 \rangle \leq_c c_2 = \langle q_2, s_2, b_2, f_2 \rangle$ if and only if $q_1 = q_2$, $s_1 = s_2$, $b_1 \subseteq b_2$, and $f_1 = f_2$.*

The ordering on configurations can be lifted to Network configurations as follows.

Definition 3. *Given two Network Configurations γ_1, γ_2, the ordering $\leq_c$ is defined as follows: $\gamma_1 \leq_n \gamma_2$ if and only if there exists an injective map h from the configurations in $\gamma_1 = \{c_1, \ldots, c_k\}$ to configurations in $\gamma_2 = \{d_1, \ldots, d_n\}$ such that $c_i \leq_c h(c_i)$ for $i : 1, \ldots, k$.*

Theorem 1. *The Coverability Decision Problem is decidable for Pub/Sub Networks.*

Proof. We apply the methodology introduced in [2,14] to prove that $\langle \rightarrow, \leq_n \rangle$ is a well-structured transition systems.

We first observe that the ordering $\leq_n$ is obtained embedding equality over finite sets and finite set inclusion into multiset inclusion. By Higman Lemma's [16], the resulting ordering is a well-quasi-ordering, i.e., for any sequence $\gamma_1\gamma_2\ldots$ there exist indexes i, j s.t. $\gamma_i \leq_c \gamma_j$.

The transition relation $\rightarrow$ induced by a client specification P is monotone w.r.t. $\leq_n$, i.e., if $\gamma_1 \leq_n \gamma_2$ and $\gamma_1 \rightarrow \gamma_3$, then there exists γ_4 s.t. $\gamma_2 \rightarrow \gamma_4$ and $\gamma_3 \leq_n \gamma_4$. The proof is based on the observation that enabling conditions for a transition rely only on the occurrence of a certain control state and on the presence of at least a certain sets of messages in the local message list. Thus, augmenting the number of client configurations or the size of local message lists cannot prevent the firing of a rule. In particular, this property holds for the *Publish* rule of the operational semantics.

Given a finite set of configuration C it is possible to compute a finite representation of $Pre(uc_{\leq_n}(C))$. An indirect proof can be given via the observation that the semantics of *Publish* can be encoded using a transfer arc operation on Petri Nets. The encoding is based on a preliminary flattening step in which topics set, message lists and connection flag are hardwired into the control state of individual components. In other words we can generate a flatten specification in which control states have the form $\langle q, s, b, f \rangle$ and in which $\leq_n$ is multiset inclusion (over finitely many labels).

The combination of all above properties proves that Pub/Sub Networks are a well-structured transition systems w.r.t. $\leq_n$. Decidability of coverability follows then from the general results in [2,14]. □

5 Notification with Retained Messages

In Pub/Sub protocols such as MQTT the broker can be instructed in order to retained the last published message for every topic. Retained messages are then distributed to new subscribers right after their first connection. The semantics of this kind of operations requires the introduction of a global state to bookkeep published messages. For brevity, we assume here that all messages (a finite set) are maintained in the broker. More specifically, a network configuration with retained messages is defined as a multiset $\gamma = \langle g, \{\!\{c_1, \ldots, c_k\}\!\} \rangle$ where $g : T \rightarrow 2^M$ is a mapping from topics to published messages, and c_i is a client configuration. We use $g(s)$ to denote $\bigcup_{t \in s}\{g(t)\}$.

The semantics of *publish* is redefined in order to update the global configuration.

$$Publish \qquad \langle g, \{\!\{\langle q, s, m, \top\rangle\}\!\} \oplus \gamma\rangle \rightarrow \langle g', \{\!\{\langle q', s, m, \top\rangle\}\!\} \oplus \gamma'\rangle$$

under the following assumptions:

- $\langle q, publish(m, t), q' \rangle \in R$,
- $\xi = E_{\top}(t, \gamma)$,

- $\mu = \gamma \ominus \xi$,
- $\gamma' = Add_m(t, \xi) \oplus \mu$,
- $g'(t) = g(t) \cup \{m\}$, $g'(r) = g(r)$ for $r \neq t$.

The semantics of *subscribe* is redefined in order to update the message set of a node upon subscription to a given topic.

$$Subscription \qquad \langle g, \{\!\{\langle q, s, b, \top\rangle\}\!\} \oplus \gamma \rightarrow \{\!\{\langle q', s \cup s_1, b \cup g(s_1), \top\rangle\}\!\} \oplus \gamma\rangle$$

under the assumption $\langle q, subscribe(s_1), q'\rangle \in R$. With this rule a client instance subscribes to the set of topics s_1.

For the extended model, the following property then holds.

Theorem 2. *The Coverability Decision Problem is decidable for Pub/Sub Networks with retained messages.*

Proof. The proof of Theorem 1 can be extended in order to deal with the semantics with retained messages. Indeed, we observe that (1) the extended transition system is still monotone and that (2) it is still possible to compute a finite representation of predecessor states passing through a flattening of the transition system that reduces configurations to multisets of control states. The resulting system can then be viewed as a Petri net with transfer arc and a control unit (the global state) for which coverability is known to be decidable [2,14]. □

We notice that synchronisation steps with control unit can also be encoded via simpler models such a Process Rewrite Systems with weak unit or finite state constraints [17–19].

6 Handling Exceptions During Notifications

In this section we consider a semantics of the *publish* operation in which the broker does not handle node failures locally to individual notification messages. In this scenario the failure during a notification for a specific client, e.g. the client is disconnected and the notification generates and exception, can lead to a failure of the entire notification phase. As a consequence, the message might be delivered to a strict subset of the active destination nodes. We assume that the sender proceed with its execution without forcing the broker to roll-back to a previous state. Let $publish_e$ denote the operation with the above described implementation of the publish operation.

Operational Semantics for $publish_e$. Let us first define $Up_\perp(\gamma)$ as the multiset obtained by setting all connection flags occurring in γ to $\perp$, namely $Up_\perp(\{\!\{\}\!\}) = \{\!\{\}\!\}$, $Up_\perp(\{\!\{\langle q, s, b, \top\rangle\}\!\} \oplus \gamma) = \{\!\{\langle q, s, b, \perp\rangle\}\!\} \oplus Up_\perp(\gamma)$, $Up_\perp(t, \{\!\{c\}\!\} \oplus \gamma) = \{\!\{c\}\!\} \oplus Up_\perp(\gamma)$ otherwise. The operational semantics of $publish_e$ is defined as follows.

$$Publish_e \qquad \{\!\{\langle q, s, m, \top\rangle\}\!\} \oplus \gamma \rightarrow \{\!\{\langle q', s, m, \top\rangle\}\!\} \oplus \gamma'$$

with the following assumptions:

– $\langle q, publish_e(m,t), q' \rangle \in R$,
  - $E_\top(t,\gamma) = \xi \oplus \eta$,
  - $\mu = \gamma \ominus (\xi \oplus \eta)$,
  - $\gamma' = Add_m(\xi) \oplus Up_\perp(\eta) \oplus \mu$.

With this rule a client sends a publish request for message m on topic t to the broker. The server acknowledges the request and, inside a synchronisation block, forwards the message m to all other active clients subscribed (at least) to topic t. If during the notification phase (typically a scanning of an internal data structure) a disconnected client is detected (i.e. the corresponding notification operation fails) the procedure exits. This effect is modelled using a non-deterministically chosen subset of active destination nodes ξ with $\xi \sqsubseteq E_\top(t,\gamma)$ that represents subscribers ready to receive message m before failure detection. The remaining potential receivers η are marked as disconnected. In this semantics, among all possible executions, we consider the case in which, during the notification phase, no disconnected clients are detected as well the case in which none or a strict subset of clients receive the notification.

The semantics of the new operation is slightly different from the typical broadcast operations adopted in Petri Nets that we took as target operation to prove decidability of coverability in the first part of the paper. Indeed this operation applies a non-deterministic split during the transfer phase in which instances are transferred from one state to another. The non-deterministic split redistributes all instances in a given state to a finite set of different states without cancellations or duplications. Despite of the use of a non-standard transfer operation, coverability is still decidable as proved in the following theorem.

Theorem 3. *Coverability is decidable for Pub/Sub Networks with the* $publish_e$ *operation.*

Proof. The proof is based on a reduction of the considered decision problem to coverability for parameterised systems composed by many finite-state components with a single monitor in which each component reacts in a non deterministic way to broadcast messages sent by the monitor. This kind of systems has been introduced in [6] to model the behaviour of synchronous systems. The decision procedure is based on a symbolic reachability algorithm based on a constraint solver for linear integer (in)equalities. The reduction requires the following steps. For the *publish* operation the encoding requires a preliminary flattening step in which topics set, message lists and connection flag are hardwired into the control state of individual clients. The flattening can then be used to associate finitely many counters (to keep track of occurrences of states in network configurations) to each control state in accord with the counting abstraction used e.g. to model Petri nets as vector addition systems. The flattening and the counter representation of control states provides a way to represent transition rule using linear integer inequalities over variables ranging over natural numbers. For instance, enabling conditions of the $publish_e$ operation can be expressed via lower bounds constraints of the form $X \leq 1$ for the counter X that denotes a given control state, e.g., a publisher state. The effect of a transition can be expressed as an

affine transformation, i.e., a linear combination defined on the current number of client instances in different counters associated to control states. Differently from other models such as transfer nets and affine well-structured Nets [15] the semantics of $publish_e$ requires inequalities in special form in which the left hand side may consists of an expression $X'_1 + \ldots + X'_n = X_1 + \ldots + X_n + Exp(Y_1, \ldots, Y_m)$ for variables X'_i denoting the value of the counter in the next state and variables X_i and Y_j denoting the current values of the counters for $i : 1, \ldots, n$ and $j : 1, \ldots, m$. In addition we need to insert the side conditions $X'_1 \geq X_i$ to ensure that variables occurring in $X'_1 + \ldots + X'_n$ will be incremented. As an example, the transitions on counters $X' + Y' = X + Y + Z, X' \geq X, Y' \geq Y, Z' = 0$ can be used to model the transfer of all instances in Z in X and Y. The effect of the transfer is to distribute the instances in Z non-deterministically between X and Y. Decidability of coverability in the resulting counter representation of the flattened transition system follows by observing that the problem can be solved by applying the symbolic backward reachability algorithm based on constraint solvers for inequalities over natural numbers proposed in [6]. The algorithm maintains constraint-based representations of infinite set of configurations via unions of constraints of the form $X_1 \geq c_1, X_n \geq c_n$. To apply termination results based on Dickson's lemma [13] on the $\leq_v$ ordering to the resulting procedure, transitions of the form $X' + Y' = X + Y + Z, X' \geq X, Y' \geq Y, \ldots$ require a further normalization steps in order to eliminate constraints of the form $X' \geq X$. The idea here is to associate an auxiliary variable $AuxZ$ to each variable Z whose value must be split between several variables. Before firing the transfer action, the transition system enters a special state in which instances in Z are moved to $AuxZ$. This phase is non-deterministically terminated in order to start the transfer arc from Z and $AuxZ$ to variables X and Y respectively. □

7 Conclusions

We have studied coverability problems for a formal model of Publish/Subscribe Networks inspired to extensions of Petri nets with broadcast and transfer arcs. Our model combines asynchronous communication with global operations and non-deterministic actions to model the effect of exceptions generated during communication between broker and individual clients. The proposed model, extensions and variants seem to be different from other infinite-state models proposed in the literature, see e.g., [1,3] for a survey on extensions of Petri nets used to model distributed systems.

For the considered model, we prove preliminary results for the coverability decision problem. The model discussed in this paper can be extended in different directions. One possible extension consists of a new operation $publish_r(m, t, q)$ in which m is a message, t is a topic name, and $q \in Q$ denotes the state in which the sender is redirected when an exception is generated during the notification step. The operational semantics requires to detect absence/presence of crashed nodes, i.e., global, universally quantified, condition for firing a transition. Finding suitable semantics for this operation for which coverability remains decidable is still an open problem.

References

1. Abdulla, P.A., Delzanno, G.: Parameterized verification. STTT **18**(5), 469–473 (2016)
2. Abdulla, P.A., Jonsson, B.: Ensuring completeness of symbolic verification methods for infinite-state systems. Theor. Comput. Sci. **256**(1–2), 145–167 (2001)
3. Bloem, R., et al.: Decidability of Parameterized Verification. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, San Rafael (2015)
4. Bloem, R., et al.: Decidability in parameterized verification. SIGACT News **47**(2), 53–64 (2016)
5. Conchon, S., Delzanno, G., Ferrando, A.: Parameterized verification of topology-sensitive distributed protocols goes declarative. In: NETYS 2018, pp. 209–224 (2018)
6. Delzanno, G.: Constraint-based model checking for parameterized synchronous systems. In: Armando, A. (ed.) FroCoS 2002. LNCS (LNAI), vol. 2309, pp. 72–86. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45988-X_7
7. Delzanno, G.: A logic-based approach to verify distributed protocols. In: CILC 2016, pp. 86–101 (2016)
8. Delzanno, G.: A unified view of parameterized verification of abstract models of broadcast communication. STTT **18**(5), 475–493 (2016)
9. Delzanno, G.: Formal verification of internet of things protocols. In: FRIDA@FLOC (2018)
10. Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized verification of ad hoc networks. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 313–327. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15375-4_22
11. Delzanno, G., Sangnier, A., Zavattaro, G.: On the power of cliques in the parameterized verification of ad hoc networks. In: Hofmann, M. (ed.) FoSSaCS 2011. LNCS, vol. 6604, pp. 441–455. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19805-2_30
12. Delzanno, G., Sangnier, A., Zavattaro, G.: Verification of ad hoc networks with node and communication failures. In: Giese, H., Rosu, G. (eds.) FMOODS/FORTE -2012. LNCS, vol. 7273, pp. 235–250. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30793-5_15
13. Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. Am. J. Math. **35**(4), 413–422 (1913)
14. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theor. Comput. Sci. **256**(1–2), 63–92 (2001)
15. Finkel, A., McKenzie, P., Picaronny, C.: A well-structured framework for analysing petri net extensions. Inf. Comput. **195**(1–2), 1–29 (2004)
16. Higman, G.: Ordering by divisibility in abstract algebras. Proc Lond. Math. Soc. **2**(7), 326–336 (1952)
17. Kretínský, M., Rehák, V., Strejcek, J.: On extensions of process rewrite systems: rewrite systems with weak finite-state unit. Electr. Notes Theor. Comput. Sci. **98**, 75–88 (2004)
18. Kretínský, M., Rehák, V., Strejcek, J.: Extended process rewrite systems: expressiveness and reachability. In: CONCUR 2004, pp. 355–370 (2004)
19. Mayr, R.: Process rewrite systems. Inf. Comput. **156**(1–2), 264–286 (2000)