A formal framework to model and validate event-based software architecture

Yahya Asadollahi*, Vahid Rafieb, Samaneh Asadollahic, Somayeh Assadollahia

*Islamic Azad University-Malayer Branch  
 bComputer Engineering Department-Arak University, Arak-Iran  
 cComputer Engineering Department-Islamic Azad University-Arak Branch

Abstract

Today, distributed systems with loose connections are increasingly developed using event-based architectures. In these architectures, systems are composed of separate components which cooperate dynamically. The communication and cooperation of components in these architectures are done through event passing mechanisms. The main challenge in these architectures is to define a framework to model and validate systems under development.

In this paper, we propose a framework using ASMETA (ASM mETA language) for modelling systems through event-based architectures. Validation of the whole system is done through model checking using Bogor. We describe the process of automatically mapping ASM models to BIR –the input language of Bogor-. In the end, we present our experimental results using the proposed approach with different case studies.

Keywords: Event-Based systems; Modeling; ASMETA; Model checking; Bogor;

1. Introduction

During the last decade the event-based architectural style has received many attentions in designing and developing of a wide range of applications. Now a day, this architecture is being noted as a key technology and promising infrastructure that is used to support the distributed applications. It is capable of combining the various set of loosely coupled components that do not directly communicate with each other but interact using generating and receiving event notifications. Each component can be noted as a producer or a consumer. Producers publish the event notifications and each consumer states its interest to the events notified by the producers through subscriptions. One of the main properties of the model is that the message receiver is unknown to the sender but the dispatcher dynamically recognizes them through subscriptions. In this architecture, it is easy to determine what each component separately performs but difficult to understand how the set of the components cooperate. Although it is possible each component act correctly when noted separately but the set of the components may provide non correct service in the cooperative jobs [1,2]. Hence, modelling and validation of the event-based architecture based systems is complex, since it should be determined how the relation between components in a distributed environment is.

To solve these problems, in this paper, we propose a formal framework in terms of an architectural style which supports modelling and validation of the event-based architecture based software systems. In the modelling part of the framework, we use ASMETA, a set of tools for Abstract State Machines (ASMs). As a matter of fact, the main property of this toolset is providing a metamodelf based language for ASM's named AsmetaL (Asmeta language that

E-mail address: asadollahi@iau-malayer.ac.ir.
we used it to write ASM models in a text format. Then the written model in AsmetaL will be compiled using the AsmetaLc tool. In the end of this part we used AsmetaS (Asmeta simulator) for performing and simulating the mentioned model. In the next step, the produced ASM models are translated to BIR-Bogor input language-to validate the proposed framework. Then we validated the whole system through applying model checking technique by using Bogor model checker.

The paper is organized as follows. Section 2 presents related work. In section 3 and 4, we present our approach to model and validate event-based architectures. Finally, section 5 concludes the paper.

2. Related Work

Nowadays the model checking of the software systems is noted as an active research part. In practice this method has proved its advantages in some area of the computer science and has been also accepted in the industry. There are many research efforts which have generally applied the model checking to validate the event based systems and especially Publish/Subscribe (P/S) systems.

Baresi et. al. [1,2] suggest an approach for modeling and validation of the distributed systems in the P/S architecture format which is based on the model checking. This approach considers two concepts in P/S systems, dispatcher and other components. Components are defined as UML state chart diagram while the dispatcher is applied as a pre-defined parametric component. In the validation phase, the properties required by the components are defined by LSC’s and converted to Automata, then the properties are translated to Promella -SPIN input language- and are delivered to SPIN for the validation of the whole system.

Garlan et al. [3] and the researchers involved in the Cadena project [4] exploit model checking techniques to verify distributed publish/subscribe architectures. These approaches do not use UML diagrams, but specify the behavior of components using their specific language [3] or using an IDL-like specification language [4]. Garlan et al. present different middleware specifications that can be integrated into the validation tool. The properties are described in CTL, which is a formalism provided by the SMV [5] model checker.

Kaveh and Emmerich [6] apply model checking methods to validate distributed systems which use remote method invocation. Components are specified using statechart diagrams. When a transition fires, some remote methods are invoked. Only potential deadlocks are explored by this tool.

The main difference between our approach and other presented approaches is the formality of the proposed framework for the modeling and validation of the software systems that are based on the event-based architecture which taking the formality into account has also capability verification through model checking.

3. Our approach

In this paper, the modeling part of the presented approach is based on ASMETA tool set, which is a set of tools for the ASMs developed by exploiting the Model-driven development (MDD) approach [7].

In our approach by AsmetaL syntax we describe each component in the format of enum domains regarding the state it can have in the system and we modele the event happening in the system through transition rules. To describe the proposed framework, we use the presented example in [1] as a case study. The mentioned example describes a simple service that controls an e-house. The service allows users to take bathroom. Whenever the user needs a bath it will publish an event meaning that it intends to take bathroom. When the bathroom receives the event, it will turn the heating on and increases the temperature and starts filling the bath. When the bath becomes full of water and the temperature reaches to hot, the bathroom will publish an event notifying user that the bathroom is ready and user can take bathroom. Power manager manages the electricity preparation. If there is any electricity disconnection, this component notifies the failure and switches from primary power resource to secondary one. We use the figure 1 and figure 2 from [1] to better understand the current event in the system.
This system includes five components that cooperate with each other to render services: user, bath, bathroom, heating and power manager. In this system each component comprises a set of states that change their states at the result of an event happening. For example, the bathroom component includes IDLE, READY and WAITING states. To describe a component and the set of its states we have used enum domain syntax. Figure 1 shows enum domains related to the components mentioned in our example.

```
enum domain Bathstate = {START | FULL}
enum domain Bathroomstate = {IDLE | WAITING | READY}
enum domain Heatingstate = {COLD | WARMING | HOT | FREEZING | OFF}
enum domain Powerstate = {OK | ALARM}
dynamic controlled bathstatus: Boolean
dynamic controlled temperaturestatus: Boolean
dynamic controlled bath: Bathstate
dynamic controlled bathroom: Bathroomstate
dynamic controlled heating: Heatingstate
dynamic controlled nextstate: Heatingstate
dynamic monitored needforbath: Boolean
dynamic monitored power: Powerstate
```

Fig. 3. Enum domains equivalent to components

The components involved in the event-based architecture do not directly interact with each other. Instead, they communicate through the following events: Consume(), publish(), subscribe(), unsubscribe(). Events include a name and a set of parameters. For example the event of publish (bathroom, ready) means that the component notifies that the bathroom is ready.

As mentioned before, we mapped the events existing in the architecture to the transition rules as we modeled publish and subscribe events through the update rules and consume rule through the conditional rule. For example in the figure 1 when user notifies that needs bath, the bathroom will change its state and notifies that the temperature should be increased and bath should start being filled. At that time, the component presents its subscriptions by adding some subscription about temperature, heating and bath. Figure 2 shows the rules related to these events that should be performed in the main rule part using block rule in parallel.
If bath is full of water and the temperature is hot, an event will be published meaning that the bathroom is ready. The event will be modeled as shown in figure 3.

macro rule r_cold2warming =
        if heating = COLD and nextstate = WARMING then
            par
                heating := WARMING
                nextstate := HOT
                bath := FULL
            endpar
        endif

macro rule r_waiting =
        if bathroom = IDLE and needforbath = true then
            bathroom := WAITING
        endif

macro rule r_ready =
        if bathroom = WAITING and bathstatus = true and temperaturestatus = true then
            bathroom := READY
        endif

Fig. 4. Transition rules equivalent to events

4. Verification and Validation

Up to now, we presented how to model the proposed architecture. In this section, we analyze our proposed method. Since the ASMETA toolset’ability for analysis is limited to simulation, hence we have decided to used the Bogor model checker to verify the presented approach. Bogor input language is a BIR text file. The language provides the essential constructs that are generally available in the modeling language of the verification tools including various kinds of primitive data like int, boolean and enum and non-primitive like record and array.

To use Bogor in this part, we should map the model written in Asmetal to the BIR language equivalent with it. We mapped each rule existing in Asmetal into a location (loc) in the BIR language. For example, figure 6 shows the location equivalent with r_ready rule of the figure 5.

loc loc0:
    when bathroom == WAITING && bathstatus == true && temperaturestatus == true
    do( Bathroom:=READY)
    goto loc1;

Fig. 5. Transition rule equivalent to an event

Fig. 6. Transition rule as a location in BIR
If there are parallel rules, a thread will be also created for each rule. Figure 7 shows the threads equivalent with two parallel rules r_waiting and r_cold2warming.

![Fig. 7. Transition rule as a thread in BIR](image)

5. Conclusion

In this paper we presented a formal framework to model and validate distributed systems based on the event-based architecture. We used a set of tools around ASMs, called ASMETA toolset, to model software systems. Components are modeled as enum domains and events are modeled as transition rules. We translated the ASM models into BIR - the input language of Bogor - to validate the proposed framework and validated the whole system through Bogor model checker. For the future work, we have a plan to implement an integrated environment to support the proposed approach for developing distributed systems. Testing other approaches instead of using Bogor, is our another future work to reach a reasonable solution for verification.

References