

Specification and verification of the structural and behavioral properties of Publish/Subscribe architectures

Siwar Khelifi¹, Hatem Hadj kacem², Ahmed Hadj Kacem¹

University of Sfax

¹ ReDCAD Research Unit

FSEG, B.P. 1088, 3018 Sfax, Tunisia

siwar.khelifi@yahoo.fr, Ahmed@fsegs.rnu.tn

² MIRACL Laboratory

FSEG, B.P. 1088, 3018 Sfax, Tunisia

hatem.hadjkacem@fsegs.rnu.tn

Abstract

Distributed applications are dynamically built as federations of components that join and leave the cooperation. Publish/Subscribe paradigm is a promising infrastructure to support these applications. However this paradigm complicates the intuitive interpretation and subsequent validation of these systems. It is easy to understand what each component does, but it is hard to understand what the global federation achieves. In this paper, we describe an approach to support the modeling and validation of Publish/Subscribe architecture style. We integrate a functional and a structural approach based on automata with multiplicities. The aim is to express dynamism while offering a simple modeling which can be easy to understand. To ensure that the system is evolving correctly, we validate the whole system through model checking using SPIN which permits to specify some properties concerning the dynamic behavior of a system.

Keywords: Publish/Subscribe, architectural styles, structural and behavioral properties, SPIN, PROMELA, automata with multiplicities.

1. INTRODUCTION

Complex systems are usually presented as some systems of interacting entities which can be represented as a kind of networks [16]. From these interacting entities some emergent processes happen and will consist in constituting some kind of organizations. To ensure the reliability of such complex systems, computer assisted verification methodologies have become a necessary step in the design process. The Publish/Subscribe paradigm is considered as a complex system and proposed as a basis for middleware platforms that support architectures software. It is composed of highly evolvable and dynamic federations of components. Components can be objects, processes, servers, applications, tools or other kinds of system entities [10]. According to this paradigm, components do not interact directly, but their communications are mediated by the event-service. Components called producers declare the events which interest them. When a component publishes an event, the event-service broadcasts it to all corresponding components, called consumers. The different components must be subscribed before interacting in the system. Publish/Subscribe middleware decouples the communication among components. The producer doesn't know the consumer of his events, but it's the event-service which identifies their events dynamically. As a consequence, new component can dynamically join the system, become immediately active, and cooperate with the other components without any reconfiguration of the architecture. The system must preserve this consistency during reconfiguration. It must be left in a correct state after reconfiguration, by maintaining the conformity of its new architecture with respect to a set of structural and behavioral properties or architecture style. These constraints motivate our approach to model and validate Publish/Subscribe architectures.

Modeling and validation the Publish/Subscribe system is the most complex task, since we must consider how components communicate in a distributed environment. In one side, our approach consists in modeling the

Publish/Subscribe system by the supply of automata with multiplicities which are considered as a tool for the description of the dynamic and evolutionary systems. In the other side, to validate this type of system, we propose to use the model checker SPIN which makes it possible to analyze a program described by using PROMELA [13]. This permits to describe the behavior of each process of a system, and the interactions between them. To describe the behavioral and structural properties, we use LTL which allows specifying properties concerning the dynamic behavior of our system.

The structure of the paper is as follows. We first introduce (in Section 2) the main characteristics of a Publish/Subscribe architecture style and then we present the dynamic aspect of this system. After that (in Section 3), we present a survey which focus on research done in the area of software architecture design. And we explain our proposed approach to model Publish/Subscribe architectures. Then, (Section 4) provides a brief description of our verification process to validate Publish/Subscribe architecture. Finally, a conclusion section ends the paper.

2. A SURVEY OF PUBLISH/SUBSCRIBE ARCHITECTURE

The Publish/Subscribe paradigm is considered as a complex system proposed as a basis for middleware platforms that support architectures software. It is composed of highly evolvable and dynamic federations of components. According to this paradigm, components do not interact directly, but their communications are mediated by the event-service. We distinguish three events models. The first model is the peer-to-peer model in which the consuming entities subscribe in a specific name corresponding to a very definite event. Then, the producers deliver directly to the specific names that correspond to the entities subscribed. The second event model uses a mediator (event dispatcher) who plays a middleware role between the producers and the consumers. Indeed, the producers publish their events to a mediator. This later distributes these events to the subscribed entities interested with them. The third is the Cambridge event model that is based on an implicit event model. In our work, we are going to focus on the second model that is based on an event-service. We distinguish the architecture using centralized event-service and the architecture using distributed event-service. In the first type, one event-service plays the role of middleware between the producers and consumers. But in the second one, there is a network of event-service. In each type, the dispatcher has connection with producers and consumers and also with the other event-service. All dispatchers cooperate together in order to distribute the events produced by producers to the consumers.

It is necessary to respect the two following constraints: all dispatchers must be interconnected and the communication must be bi-directional. In the case of network dispatchers, the most used interconnection topologies are hierarchical, acyclic peer-to-peer and general peer-to-peer [3, 4, 20].

We must recall that the dispatcher, the producers, the consumers as well as the producers-consumers are software components part of the Publish/Subscribe system. The purpose of the architecture of this system is to describe how to deploy these components. Indeed, the topology of deployment of these components implies various variants of architectures.

According to the rules of interaction in a Publish/Subscribe style, the communication between the components of this architecture can be established either by the Push or the Pull mode.

- The Push mode is used to deposit the event if the entity is the initiator of communication. If not, the role of this mode is a notification through a message to inform the other partner that an event is deposited.
- The Pull mode is used to pull out the event if the entity is the initiator of communication. If not, the role of this mode is a notification through a message to inform the other partner that the event is withdrawn.

There are several alternatives of Push and Pull modes according to the partners. For the interaction between producer and dispatcher, two new alternatives of Push and Pull are at work. The first is PushP. It means to deposit the event if the producer is the initiator of communication. Otherwise, PushP plays the role of a notification. The second is PullD. It means to pull out the event if the dispatcher is the initiator of communication. If not, PullD plays the role of a notification. For the interaction between producer-consumer and dispatcher, we use the same alternatives but we must replace the PushP and Pull by PushPC and Pull if the producer-consumer is considered as a producer. For the interaction between dispatcher and consumer, there exist also two alternatives of Push and Pull mode. The first is PullC. It means to pull out an event if the consumer is the initiator. If not, PullC plays the role of a notification. The

second is PushD. It means to deposit the event to the consumers concerned by this occurrence if the dispatcher is the initiator of communication. Otherwise, PushD plays the role of a notification. Concerning the interaction between producer-consumer and dispatcher, if the producer-consumer is considered as a consumer, we use the same alternatives by replacing PullC and PushD by PullPC and Push. For the interaction between dispatcher and another one, in the case of dispatcher network, we use PushDD which means to deposit the event to a neighbouring dispatcher.

2.1. Dynamicity of the Publish/Subscribe System

The study of the dynamicity of software architecture is an active domain of research in the scientific community [18]. To allow the architectures to satisfy some properties regardless the changes of their environment, they must have the capacity to react to the event of the user or the environment and to execute architectural changes in an autonomous way. The reconfiguration of an architecture consists in proposing to the architectural level the following changes:

- The suppression or the addition of new components.
- The suppression or the addition of new link between components.

These mechanisms are the operations of the architecture reconfiguration. They require a control which aims to adapt the architecture to the requirements of the user or the environment. The process of adaptation to reconfigure an architecture must guarantee the preservation of its properties. This process must be controlled correctly to assure the modification and the maintenance of this architecture. For these reasons, we need:

- An evaluation of the change to determine which properties are affected and which inconsistencies can occur.
- An organization of the change to assure the system's consistence while new components and links are dynamically added or removed from the system.

3. MODELING OF PUBLISH/SUBSCRIBE ARCHITECTURE

Dynamic software architectures make it possible to consider the structural and behavioral changes of a system which will take place during the execution. It is important during design to analyze the possible changes of a system in order to avoid undesirable configurations. For that, it is necessary to have a design tool supporting the dynamic aspect of architecture. The architecture description languages (ADLs) is the frequently used tool to model the dynamic systems since they are capable to formalize dynamic architectures and/or to formalize architectural styles. Among these languages, we can mention ACME [9], ARMANI [24] or SPACE [19]. In spite of their capacities to represent the dynamics of the system and the architectural styles, they present some limits. On the one hand, for ACME and ARMANI for example, the definition of the architectural behaviors is not taken into account. Thus, it is impossible to specify the evolution of the architecture. On the other hand, they are limited to describe the styles of software architecture. They make it possible to reveal only static constraints, but not the behavior and dynamic constraints. Related works [21, 17] propose the use of graphs to represent the software architecture and the graph grammars to represent the architectural style. The graphs constitute the most intuitive tool to model complex situations. But, they also present some inconveniences. Indeed, graphs are used to represent the configuration of architecture while graph grammars showed their relevance to represent reconfiguration. To reconfigure architecture, we must guarantee the preservation of the properties and architecture style. However graph grammars don't permit to describe logical properties concerning for example the number of process components. they do not present an efficient tool to describe logical properties and to specify the post conditions of a reconfigured operation. Another tool that permits to ensure the consistency of the system is necessary when new components and links are dynamically added or removed from the system.

3.1. Our Approach

In this section, we will present our approach. Firstly, we describe architecture reconfiguration using an effective and simple tool automata with multiplicities. Next, we explain the verification process which we have elaborated to validate the Publish/Subscribe system with respect to its architectural style using a SPIN model checker.

3.1.1. Automata with multiplicities for modeling

To specify software architectures, we need a simple tool which allows us to obtain an easy description of the dynamicity of software architectures. The most intuitive tool is automata with multiplicities, since they describe

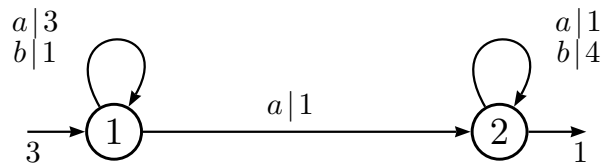


Figure 1: A \mathbb{N} -automaton

dynamicity of system using the concept of state and transition. The structure of automata is similar to the structure of a dynamic system. They are used, therefore, to describe the evolution of a system during the time. Automata with multiplicities (or weighted automata) are a versatile class of transition systems. They have been studied extensively for different various purposes such as image compression, speech recognition, formal linguistic (and automatic treatment of natural languages too) and probabilistic modeling. We show here that we can use this tool to model our Publish/Subscribe system and we try to keep the configuration of its architecture. To introduce the notion of automata with multiplicities, we must recall the notion of semiring.

A semiring $(k, \oplus, \otimes, 0_k, 1_k)$ is a set together with two laws and their neutrals. More precisely $(k, \oplus, 0_k)$ is a commutative monoid with 0_k as neutral and $(k, \otimes, 1_k)$ is a monoid with 1_k as neutral. The product is distributive with respect to the addition and zero is an annihilator ($0_k \otimes x = x \otimes 0_k = 0_k$) [12]. The set $(\mathbb{N}, +, \times, 0, 1)$ and $\mathbb{B} = (\{0, 1\}, \vee, \wedge, 0, 1)$ are well-known examples of semirings.

An automaton with multiplicities [6, 1] (**weighted automaton** or again **linear representation**) $\mathcal{A} = (\lambda, \mu, \gamma)$ with weights in a semiring k is given by:

- A : an alphabet,
- Q : a set of n states,
- $\mu : A \rightarrow k^{n \times n}$: the transition matrices,
- $\lambda \in k^{1 \times n}$: the initial vector,
- $\gamma \in k^{n \times 1}$: the final vector.

Such an automaton is usually drawn as a directed valued graph (see Figure 1). A transition $(i, a, j) \in \{1, \dots, n\} \times \Sigma \times \{1, \dots, n\}$ connects the state i with the state j . Its weight is $\mu(a)_{ij}$. The weight of the initial (respectively final) state i is λ_i (respectively γ_i).

The notion of automata with multiplicities is a generalization of boolean one. For the second type, the weight is one (if the transition exists) or zero (if not) but for the first type, the weight is taken from a set other than \mathbb{B} .

3.1.2. Automata with multiplicities for Publish/Subscribe architecture

Publish/Subscribe is now largely acknowledged as one of the most interesting paradigm for distributed interaction [8]. To model our system, we will use automata with multiplicities since a dynamical system can be viewed as a set of states (objects or components) interacting between them by means of events or actions. When a system is in state 1 and an event B occurs, then the system passes to state 2. Thus, to model our system, we consider the states of automaton with multiplicities as the software entities constituting architecture and the transitions can correspond to the communication link between these entities. The label of each transition is constituted of two values. The first value is from an entry alphabet whose elements corresponds to the Push or Pull mode. The second value is from a set of outputs whose elements correspond to the event deposited by the entity. At any time, the values of these variables describe the system's behavior. Over time, the dynamic system generates a series of values, controlling the behavior of the system.

Our model described in Figure 2 considers the system that contains the following entities: a producer, a consumer, a producer-consumer and a dispatcher (event service). There is the case of centralized dispatcher.

We must specify here that coefficients of transitions are taken from the Boolean semiring. This coefficient will represent the occurrence number of events sent between the components system. The alphabet of the

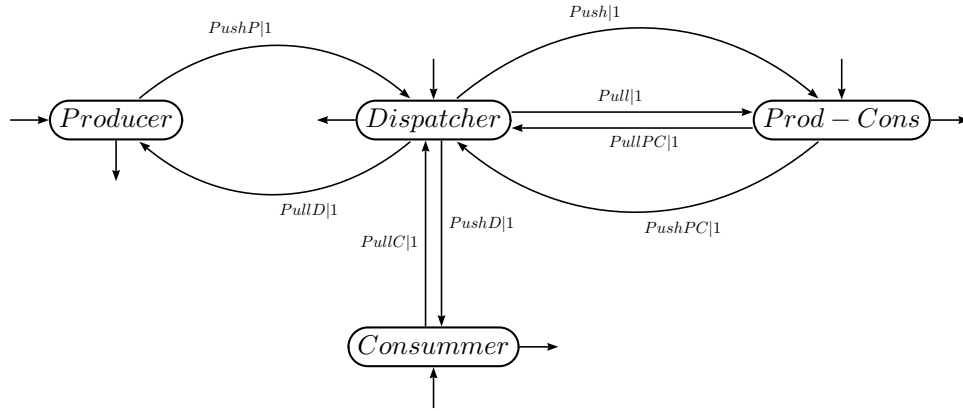


Figure 2: Modeling with event-service centralized

automaton is $A = \{PushP, PushD, PushPC, Push, PullD, PullC, PullPC, Pull\}$. The set of states is $Q = \{Producer, Dispatcher, Consumer, Prod - cons\}$,

$$\lambda = (1 \ 1 \ 1 \ 1), \quad \gamma = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \text{ and } \mu(Push) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

In figure 3, we have the automaton corresponding of the Publish/Subscribe modeling in the case of network dispatcher. The data of this automaton is the following:

$A = \{PushP, PushD, Pull, PushDD, PushPC, Push, PullD, PullC, PullPC\}$.

$Q = \{producer, Dispatcher, Dispatcher1, Consumer, Prod - cons\}$.

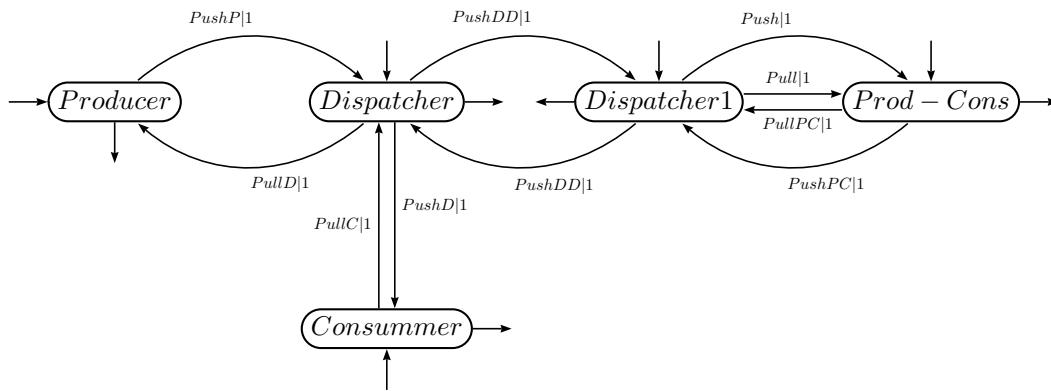


Figure 3: Case of dispatcher network

$$\lambda = (1 \ 1 \ 1 \ 1 \ 1), \quad \gamma = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \text{ and } \mu(PushDD) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

We described above the dynamicity of our system by means of an easy model to understand. To ensure that our model evolves correctly, it is necessary to study its principal properties in order to check the consistency of our system during its reconfiguration.

Nowadays, researches acquire a great deal of importance to check global properties of the Publish/Subscribe system. Indeed, the software architecture of this system is characterized by its style which defines a set of general rules that describes or forces the structure of this architectural behavior and the manner of the components interaction. It is therefore important to investigate the evolution of this architecture and to make sure that the structural and behavioral properties are respected during its reconfiguration.

Up to now, the related works [21] could check only the structural properties of the Publish/Subscribe system. Besides, the authors of [26] limits their work to verify only the properties of security and vivacity of this system. This paper attempts to explore and review these structural properties and examine those behavior. We are going to help the architect therefore to conceive a consistent system that must be corrected after its reconfiguration by maintaining the conformity of its style at the structural and behavioral level.

3.2. Validation

This activity aims to prove that the system developed maintains the conformity of its new architecture with respect to a set of structural and behavioral properties or architecture style. The system must preserve this consistency during reconfiguration. It must be left in a correct state after reconfiguration. Thus, we will try here to give a brief survey on the approaches proposed to validate Publish/Subscribe system after its reconfiguration. One of the most promising verification techniques, which is increasingly used, is Model Checking [22]. It is a successful verification technique for models expressed as finite state machines. The concept is general and applies to all kinds of logic and suitable structures. Model checking verifies a given set of state machines with respect to a set of temporal formula. A simple model-checking problem is to test whether a given formula in the propositional logic is satisfied with a given structure. There exist several tools for checking dedicated to the analysis and the checking of the complex system by the technique of model-checking. However FDR, SPIN and MURPHI, are representatives of explicit state model checkers. FDR (Failures Divergence Refinement) is a model checker for the specification and implementation CSP (Communicating Sequential Processes) [15]. It is used for proving safety, liveness and combination properties, and also for establishing refinement and equality relations between systems. In our work we cannot use FDR tool for many reasons. The most important reason can be presented as follows: the communication's mechanism between the processes is synchronous (based on the principle of appointment) and the communication's mechanism for the Publish/Subscribe system is asynchronous (if the recipient cannot immediately treat the reception of a message, this one will be stored in its thread). The tool adopted in our work is the model checker SPIN (Simple ProMela INterpreter) [14, 23]. This Model checker allows to present synchronous or asynchronous communication between processes [14, 11].

3.2.1. The Model Checker SPIN/PROMELA

The Model will be used in our work is SPIN. This Model checker allows to present synchronous or asynchronous communication between processes. It performs simulation and model checking on system verification models written in a language PROMELA. The models (programs) written in PROMELA are used as an input for SPIN software package because of their automated simulation and validation [2]. The choice of this tool is justified for three primary reasons. Firstly, Spin allows the checking of the properties expressed in LTL (logical temporal linear) [25]. This permits to specify properties concerning the dynamic behavior of our system. Next, according to [4], PROMELA can be considered as a language of specification to modelize finite state system. However our system is modeled by the automata with multiplicities which is a generalization of finite state automata. Finally, PROMELA allows modeling the communication between components by local channels (synchronous or asynchronous communication). However, communication between components for the Publish/Subscribe system is asynchronous.

PROMELA makes it possible to describe the behavior of each process of a system. It provides the sending and receiving primitives and the asynchronous composition of concurrent processes. Temporal logics are used in the description of information processing systems in order to specify properties concerning the dynamic behavior of the system. The addition of the temporal operators to the connectors and the operators of traditional logic, will make it possible to formally state properties on the sequence of states i.e. the executions of the studied system. Two ideas to check the validity of any property. First, we add assertions to a PROMELA specification and verify their validity by

running SPIN. Second, we can formulate LTL property and test their validity against the PROMELA specification, by running SPIN. The property is possibly enriched with specific labels identifying relevant points of process executions. Since PROMELA doesn't support the automata with multiplicities, we will convert this type of automata to Boolean one. In this case, the coefficients of the transitions will be taken from the Boolean semiring. Two types of properties we can be verified with the SPIN tool: the structural and the behavioral properties.

3.2.2. Structural properties

In order to check that our system evolve correctly, we must verify that some structural properties are satisfied by our system. There are many properties to verify but here we detail some them.

- Property S_1 : If the producer is the initiator, always if it sends PushP, then eventually, it will receive PullD. The LTL formula corresponding to this property is :

```
#define p1 sendPushP==true
#define p3 receivePullD==true
[] (p1 → ◇ p3)
```

- Property S_2 : If the consumer is the initiator, always if it sends PullC, then eventually, it will receive PushD. The formulation of this property with LTL is the following:

```
#define c1 sendPullC==true
#define c2 receivePuhD==true
[] (c2 → ◇ c1)
```

- Property S_3 : If the Producer-consumer is the initiator, always if it sends PushPC, eventually it will receive Pull.
- Property S_4 : Always if the dispatcher_1 receives one event, then possibly it will diffuse it to the dispatcher_2 or if the dispatcher_2 receives event, then eventually, it will diffuse it to the dispatcher_1. That is to say each event which comes to the one have to reach the other. This property concern the dispatcher network.

3.2.3. The behavioral properties

One of the prime characteristics of Publish/Subscribe is its federation of components which are highly evolutionary and dynamic. To validate the system against this high-level of survivability requirements, we must check the behavioral properties of Publish/Subscribe system, in order to make sure of its consistency. We must recall here that the subscriber must express these preferences concerning the events. Although there exist various choices of subscription: subject-based, content-based and type-based subscription [7, 5], we will be interested with the mechanism of subject-based subscription in order to study the behavioral properties of Publish/Subscribe system. This type of subscription model provides various properties relating to the dynamic behavior of our system. The principal properties which we checked are as follows:

- Property B_1 : If the producer is concerned with the subject_i, therefore always, it publishes events relating to the subject but not to the subject_j. That is to say if the producer is subscribed in Subject_i. He can not send an event for other subjected in which he is not subscribed. The formulation of this property with LTL is the following:

```
#define suje1 s1==true
#define suje2 s2==true
[] ( suje1 || ! suje2)
```

- Property B_2 : The consumer receives only the event which he's interests with. Always, if the producer publishes an event relating to the subject_i, eventually all the consumers entities subscribed on this subject must receive it. This property means that the consumer receives only the event concerning the subject in which it is subscribed. The LTL formula corresponding to this property is :

```
#define sujte1 retrysuj1==true
#define sujte2 retrysuj2==true
[] (sujte1 || ! sujte2)
```

Property	Result	Number of states	Memory usage (Mb)
S_1	True	228420	134.206
B_1	True	107993	134.047

Table 1: Model Checking results.

- Property B_3 : Always, The consumer must receives one and only one occurrence related to the same event from the dispatcher. This latter must send one occurrence related to the same event. The formulation of this property with LTL is the following:

```
#define consumeev1 con1eve1==true
#define consumev1 coneve1==true
[] (consumeev1 → ! consumev1)
```

- Property B_4 : The number of produced events is equal to the number of consumed events. The LTL formula corresponding to this property is :

```
#define Nbevs1 (ps1==ic)
#define Nbevs2 (ps2==ic1)
#define Nbevs3 (ps3==ic2)
[] ((<> Nbevs1) && (<> Nbevs2) && (<> Nbevs3))
```

- Property B_5 : If there are several producers who deposit the same event to the dispatcher, eventually one and only one occurrence of this event must be sent to all consumers concerned with it. With this property the consumer receives only once the same message arriving from several producers to the dispatcher.

```
#define productev1 eve1pr==true
#define product1ev1 eve1pr1==true
[] ( productev1 → ! product1ev1)
```

- Property B_6 : If there are three consecutive dispatchers D, D_2, D_1 . D and D_1 are the frontiers dispatchers and D_2 is an intermediate dispatcher connected with D and D_1 . Thus there are no other entities connected with D_2 except the dispatchers.

```
#define subscD2 sd4==true
#define retry reti==true
[] (retry → ! subscD2)
```

- Property B_7 : A consumer doesn't have the right to pull out an event only if it's published previously.

```
#define product produc==true
#define consume consom==true
[] (product → ◇ consome)
```

The model checking results are giving in Table 1. For the first structural property and the first behavioral one, we give the verification result, the number of states explored and the memory usage.

4. CONCLUSION

Publish/Subscribe is now largely acknowledged as one of the most interesting paradigm for distributed interaction. Since it is characterized by its dynamicity and evolution, this kind of systems is hard to model and to validate due to the fact that we cannot simply build a model with fixed topology. Then, modeling as well as checking of this system are the main aim of several researchers. We proposed in this paper, an approach to design and validate distributed architectures based on the Publish/Subscribe paradigm. Our approach allows to describe the dynamic of this software

architecture using automata with multiplicities of which its structure is similar to the structure of a dynamic system. To validate and verify the behavior and structure of the system model, we use SPIN model checker which is a particularly successful tool widely adopted to perform automatic verification of software specifications. Components and properties are translated into PROMELA and then passed to SPIN to make sure that the system respects its properties. With an aim of checking these properties, two classes of systems are especially highlighting our work. The first class is a centralized peer-to-peer applications, which are characterized by using a centralized event-service. In this case, the system is simply acting as one medium for the communication. The second one is a distributed peer-to-peer applications, which are characterized by using a network of event-service. Here, this application requires a good deployment of a dispatcher's network. This is often done to achieve some scalability, survivability, or control properties. Indeed, our primary aim is to evaluate the performance and validity of the properties in an event-service's network. Currently, we checked the structural and behavioral properties on Publish/Subscribe architecture using a centralized and a distributed event-service which topology is acyclic peer-to-peer with only one access point. In our future work, we will check all the properties already defined by adopting acyclic topology peer-to-peer with several access points.

Bibliography

- [1] Berstel J., Reutenauer C., "Rational Series and Their Languages". *EATCS, Monographs on Theoretical Computer Science*, Springer Verlag, Berlin (1988).
- [2] Bosnacki, D., "Extending Promela and Spin with Discrete Time". *In: Proc. of the VIII Conference on Logic and Computer Science*, (1997).
- [3] Carzaniga A., Rosenblum D., Wolf A. L., "Interfaces and Algorithms for a Wide-Area Notification Service". *Technical Report CU-CS-888-99*, University of Colorado, Berlin, (1999).
- [4] Carzaniga A., Rosenblum D., Wolf A. L., "Design and evaluation of a wide-area event notification service". *ACM Transactions on Computer Systems*, **19**(3):332-383, (2001).
- [5] Cugola G., Di Nitto E., Fuggetta A., "The JEDI event-based infrastructure and its application to the development of the OPSS WFMS". *IEEE Transactions on Software Engineering*, **27**(9):827-850, (2001).
- [6] Duchamp G., Hadj Kacem H., Laugerotte É., "Algebraic elimination of ε -transitions". *Discrete Mathematics and Theoretical Computer Science*, **7**(1):51-70, (2005).
- [7] Eugster P. T., Felber P. A., Guerraoui R., Kermarrec A. M., "The many faces of Publish/Subscribe". *ACM Comput. Surv.*, **35**(2):114-131, (2003).
- [8] Fiadeiro J.L., Lopes A., "A Categorical Semantics of Event-based Architectures". *Mathematical Structures in Computer Science*, **17**(5):1029-1073, (2007).
- [9] Garlan D., Monroe R., Wile D., "ACME: Architecture Description of Composed-Based Systems". *Foundations of Component-Based Systems*, Ed Kluwer S., (2000).
- [10] Garlan D., Kersonsky S., Kim J.S., "Model Checking Publish-Subscribe Systems". *Lecture Notes in Computer Science, Springer Berlin*, Vol. 2648, (2003).
- [11] Hadj Kacem A., Hadj Kacem N., "From Formal Specification to Model Checking of MAS Using CSP-Z and SPIN". *International Journal of Computing & Information Sciences*, **5**(1), (2007).
- [12] Hebisch U., Weinert H. J., "Semirings - Algebraic Theory and Applications in computer Science". *World Scientific Publishing*, Singapore, (1993).
- [13] Hozmann G. J., "Basic Spin Manual". *Technical report, AT& T Bell Laboratories*, Murray Hill, N.J., Mar., (1994).
- [14] Holzmann, G.J., "The model checker SPIN". *IEEE Transactions on Software Engineering*, **23**(5):279-295, (1997).
- [15] Hoare C.A., "Communicating Sequential Processes". *Prentice-Hall International*, Englewood Cliffs, New Jersey, (1985).
- [16] Kadri H., Ghnemat R., Hadj Kacem H., Bertelle C., Duchamp H. E., "Emerging Decision Support System for Geographical Information Systems". *International Conference on Economics, Law and Management ICELM2*, Tirgu-Mures, Romania, (2006).
- [17] Le Métayer D., "Describing software architecture styles using graph grammars". *IEEE Transactions On Software Engineering*, **24**(7):521-533, (1998).
- [18] Lévy N., Ramdane-Cherif A., "An approach for Dynamic Reconfigurable Software Architectures". *Sixth World Conference on Integrated Design and Process Technology (IDPT'02)*, Pasadena, California, USA, (2002).
- [19] Leymonorie F., Cimpan S., Oquendo F., "Extension d'un langage de description architecturale pour la prise en compte des styles architecturaux : Application à J2EE". *Proceeding of the 14th International Conference on Software and Software Engineering and their Applications (ICSSEA)*, Paris, (2001).

- [20] Liu Y., Plate B., "Survey of Publish/Subscribe Event Systems". *Indiana University, Department of Computer Science*, (2001).
- [21] Loulou I., Hadj Kacem A., Jmaiel M., Drira K., "Consistent reconfiguration for publish/subscribe architecture styles". In *The 1st International Workshop on Verification and Evaluation of Computer and Communication Systems VECoS'07, Algeria*. Electronic Workshops in Computing EWIC Series, The British Computer Society, (2007).
- [22] Masson A. P., "Vérification par model-checking modulaire de propriétés dynamiques PLTL exprimées dans le cadre de spécifications B événementielles", *Ph.D. Thesis, Université de Franche-Comté*, (2001).
- [23] Meenakshi. B, "A tutorial on SPIN", *Honeywell Technology Solutions Lab Bangalore 560076, India*, (2004).
- [24] Monroe R.T., "Capturing Software Architecture Design Expertise With Armani". *Technical Report CMU-CS-98-163, CMU School of Computer Science*, (2000).
- [25] Pnueli, A., "The temporal logic of programs". In: *FOCS*, 46-57, (1977).
- [26] Tanner A., Mühl G., "A formalisation of message-complete publish/subscribe systems". *Technical Report Rote Reihe 2004/11, Berlin University of Technology, Amsterdam, the Netherlands*, (2004).