A Petri-Net model for the Publish-Subscribe paradigm and its application for the verification of the BonjourGrid middleware

Leila Abidi*, Christophe Cérin† and Sami Evangelista†

*Université de Tunis, Unité de recherche en technologie de l’information et de la communication (UTIC), École Supérieure des Sciences et Techniques de Tunis, 5 Avenue Taha Hussein, BP 56, Bâb Manara, Tunis, Tunisie. Email: leila.abidi@lipn.univ-paris13.fr
†Université de Paris 13, LIPN UMR CNRS 7030, 99, avenue Jean-Baptiste Clément, 93430 Villetaneuse, France Email: christophe.cerin@lipn.univ-paris13.fr, sami.evangelista@lipn.univ-paris13.fr

Abstract—In this article we focus on the modelization of the BonjourGrid protocol which is based on the Publish-Subscribe (Pub-Sub) paradigm, a paradigm for asynchronous communication that is useful for implementing some approaches in distributed programming. The aim of this paper is to isolate the generic mechanisms of construction for the publish-subscribe approach then to model and verify, based on those mechanisms, the BonjourGrid protocol that allows the coordination of multiple instances of desktop grid middleware. We produce models using colored Petri nets in order to describe a specific modeling approach for the Pub-Sub paradigm. Such models are important, first, to formally verify the adequacy of BonjourGrid in the coordination of resources in desktop grids - for example by proving the absence of a deadlock in the BonjourGrid protocol, and second, to offer a ‘composition’ mechanism for integrating any protocol based on the Pub-Sub paradigm. These ideas are illustrated along the BonjourGrid case study and they constitute a methodology of building Pub-Sub systems.

Keywords-Principles of services, Service models, Grid Computing Middleware, Publish-Subscribe paradigm, BonjourGrid, Formal Models, Colored Petri Nets.

I. INTRODUCTION

The development of distributed computing and the presence of many resources on the Internet largely underutilized contributed to the emergence of mechanisms for resources sharing. This concept was embodied in the Grid Computing systems. It promotes resource sharing. Once resources can be represented as standard-based interfaces such as Web services, Grid Computing needs to address its service-oriented behaviors. In parallel with advances on middleware and tools, Grid platforms for researchers has been deployed such as TeraGrid [24], EuroGrid [1] and Grid5000 [25]. We have built in a previous coding work a resource sharing Grid Middleware named BonjourGrid [7]–[9] which is based on Service-Oriented Architecture in order to hide the complexity of discovering and coordinating resources. Our Resources Grid Middleware aims at offering a distributed infrastructure and the orchestration of middleware to support resources discovery and coordination in a Desktop Grid Computing environment. From a modeling perspective the resource sharing we introduce in this paper is related to Service models that is to say to principle of services (M.1 category as listed in [2]).

In the area of Service Computing, Services Systems are mainly implementing by IT-based middleware (Bonjour from Apple in our context) that realize a certain business service (the sharing of volunteers or PC resources across different Desktop Grid middleware). In order to make a system dynamic, we may authorize that multiple instances of Desktop Grid middleware coexist inside the Grid and may also disappear when some jobs are accomplished.

They are now numerous Desktop Grid middleware, among them BOINC [26], Condor [10] and XtremWeb [27]. Desktop Grids (DGs) are made with PCs and Internet as the communication layer. DGs aim at exploiting the resources of idle machines over Internet. The BonjourGrid middleware has appeared in this context. The basic idea is to exploit, dynamically, different instances of DG middleware. The coordination is fully distributed through Publish-Subscribe (Pub-Sub) mechanisms. BonjourGrid is the first middleware of this kind, to our knowledge, and it provides a view of the architecture and of the execution of applications running inside the middleware that match the ideas of decentralization.

Given the nature of an Service-Oriented Architecture, we need to prove that the protocol -on which is based our Grid Middleware- is correct (regarding some desired properties) and to demonstrate that the protocol scales well since we introduce in the model as much participants as we want… in fact as much participants as the verification tool can manage. We also need formal verification in order to validate the BonjourGrid protocol and then to ensure the proper functioning of the BonjourGrid middleware over largely distributed resources. Since BonjourGrid is established on Pub-Sub paradigm, we propose in this paper to start by providing a formal model to verify the Pub-Sub system. Based on it, we synthesize an improved formal model to validate
BonjourGrid. The contributions of this paper consist on providing a formal verification for the BonjourGrid protocol by modeling it according to the colored Petri nets, and performing simulations on the obtained net to gain more confidence in our model and current implementation.

We do not use Web Services, Web service Publishing or Web Services Discovery standards as the realization technologies because BonjourGrid was initially designed to run on top of clusters or clusters of clusters running in the same LAN and not over Internet. Web Services are not employed in clusters of clusters. Second, we are looking for High Performance computing systems and Web Services are usually synchronous systems that may not scale. We do prefer to use an asynchronous paradigm (Pub-Sub) as the realization technology. Our work is however rooted to Service Discovery and Publication processes and to a formal methodology to help construct a robust distributed system. We will explain later on what are the application issues of our work.

This paper is organized as follows. After an introduction of the context of this work, we introduce in section 2 the principle of resource coordination (as implemented in BonjourGrid) and the benefit in using Pub-Sub systems. We also introduce our motivations for the design of BonjourGrid and our motivations about its formal verification. Section 3 is related to our contributions and it describes the different steps of our work in order to provide a formal specification of the Pub-Sub paradigm. In section 4, we present our views to mix our Pub-Sub substrate with the core part of the BonjourGrid protocol. Section 5 is devoted to related works. Section 6 is about application issues and section 7 concludes the paper.

II. CONTEXT AND MOTIVATIONS

A. Resources coordination

Desktop Grids (DGs) are characterized by a dynamic environment due to the heterogeneity and volatility of resources, in our case PCs at home. User’s machines can join or leave the grid at any time, without any constraint. Each machine has its own properties such that its memory size, bandwidth, CPU/core number... which makes the work of the scheduler a difficult task.

Consequently, the power of DGs that resides in the participation of volunteers, constitutes also a weakness in terms of resources orchestration when a job is submitted. Thus, the main problem with DGs is coordination, in particular when we have to execute communicating applications i.e. applications that are modeled by a task graph with precedence.

To bypass these problems, BonjourGrid counts on a distributed vision for the coordination and the execution of applications based on existing DGs middleware. Moreover, the coordination mechanism is based on the Pub-Sub paradigm.

B. The Pub-Sub paradigm

The Pub-sub paradigm is an asynchronous mode for communicating between entities. Some users, namely the subscribers or clients or consumers, express and record their interests under the form or subscriptions, and are notified later by another event produced by other users, namely the producers [11].

As stated on Figure 1, subscribers record their interest by a call to the subscribe() operation inside the event service management system, without knowing the source of events. The unsubscribe() operation allows to stop a subscription. The notify() or publish() operation is called by publishers in order to generate events that will be propagated to subscribers and such event are managed by the event service management system too. Each subscriber will receive a notification for every event that is conform to its interest. This communication mode is thus multipoint, anonymous and implicit. It is a multipoint mode (one-to-many or many-to-many) because events are sent to the set of clients that have declared an interest into the topic. It is an anonymous mode because the provider does not know the identity of clients. It is an implicit mode because the clients are determined by the subscriptions and not explicitly by the providers.

It is also known that this asynchronous communicating mode allows spatial decoupling (the interacting entities do not know each other), and time decoupling (the interacting entities do not need to participate at the same time). This total decoupling between the production and the consumption of services increases the scalability by eliminating many sorts of explicit dependencies between participating entities. Eliminating dependencies reduces the coordination needs and consequently the synchronizations between entities. These advantages make the communicating infrastructure well suited to the management of distributed systems and simplify the development of a middleware for the coordination of DGs.

C. BonjourGrid

BonjourGrid is an approach for the decentralization and the self organization of resources in DG systems [7]–[9].
The key idea is to exploit existing DG middleware (Boinc, Condor, XtremWeb) and concurrently to manage multiple instances of DG middlewares. The notion of meta desktop grid middleware has been introduced with BonjourGrid and the Pub-Sub paradigm is used intensively for the coordination of the different DG middlewares.

Each user, behind a desktop machine in his office, can submit an application. BonjourGrid deploys a master (coordinator), locally on the user machine, and requests for participants (workers). Negotiations to select them should now take place. Using a publish/subscribe infrastructure, each machine publishes its state (idle, worker or master) when changes occur as well as information about its local load or its use cost, in order to provide useful metrics for the selection of participants. Those information about the machine are interpreted as a resource which is published as a Web service. Under these assumptions, the master node can select a subset of worker nodes according to selection criteria. That aspect is managed as a Web service discovery. The master and the set of selected workers build the Computing Element (CE) which will execute and manage the user application. That aspect is managed as a Web services composition. When the execution of an application of a CE terminates, its master becomes free, returns to the idle state and it releases all workers before returning to the idle state too. Then, the nodes can participate to others projects.

To implement this approach, BonjourGrid has been decomposed in three fundamental parts: a) A fully decentralized resources discovery layer, based on Bonjour protocol [22]; b) A CE, using a Desktop Grid middleware such as XtremWeb, Condor or Boinc, which executes and manages the various tasks of applications; c) A fully decentralized protocol of coordination between a) and b) to manage and control all resources, services and CEs.

D. Needs for formal verification

Papers about Pub-Sub systems [12]–[14] are invitations to investigate more deeply the BonjourGrid protocol, in particular under the perspective of the verification of a distributed system. In the remainder of the section, we explain how we interpret the nature of the Pub-Sub paradigm and then we propose a specific formal model for the BonjourGrid protocol. This result tends to show that the protocol is correct (regarding some desired properties) but also to demonstrate that the protocol scales well since we can introduce in the model as much participants as we want... in fact as much participants as the verification tool can manage in its service-oriented architecture.

III. CONTRIBUTIONS

A. Analysis and criticisms

The starting point of this work is to provide a formal verification for the BonjourGrid protocol. Such verification is for proving that the protocol is correct, then that any analyzed configuration will produce the ‘good’ answer. This goal requires the formal verification of properties that we expect for the protocol: safety (nothing "bad happens") for which the absence of deadlock is an example and liveness ("something good eventually happens").

We have modeled the BonjourGrid protocol according to the colored Petri Net [3] formalism and we have used the CPN Tools [28]. CPN Tools is a fast and efficient simulator that handles both untimed and timed nets. Full and partial state spaces can be generated and analyzed, and a standard state space report contains information such as boundedness properties and liveness properties. By means of a simple query language it is possible to specify and to check system-specific properties.

A dedicated tool named ASK-CTL [5] allows to evaluate and to validate expressions written in Standard ML and representing the properties of accessibility, liveness for instance. One of main motivations that led us for the modeling approach by CPN was its capacity to abstract formal verification from mathematical computing and especially the possibility of coding easily properties using Standard ML.

Most of the evaluations have been conducted with the formal library ASK-CTL [5]. A preliminary modelization led to difficulties in the validation of some properties when using the CPN Tools. Some evaluation results are not expected. In the BonjourGrid system, if we have a worker then we have also a coordinator that is attached to this worker. In other words, if there exists a worker then there exists also at least one coordinator. This property has been coded in Standard ML and we expect that it will evaluate to true. It has been evaluated to false.

The non-validation of some properties leads to the following conclusions:

- either we have made a bad modelization of the protocol;
- or we have made a good modelization but the protocol is not correct;
- or our model is not faithful enough and overly abstracts the BonjourGrid protocol which is more complex.

The last explanation is the correct one. In our modelization we have made choices that do not match the implementation choices. This is partly due to the fact that we made a verification after the implementation and not during the initial stage of the specification. We could investigate again the colored Petri Net to smooth out problems but we have taken another decision because the current Petri Net model does not capture or isolate in a clear manner what is relevant to BonjourGrid and what is relevant to any Pub-Sub system.

We shall say again that our quest is to provide with a black box that modelizes the Pub-Sub paradigm and that is able to be composed easily with any protocol that expect the paradigm. In this sense, BonjourGrid is just a case study to analyze the difficulties in building such system.

We need more abstraction and we decided to continue in the following directions:
• modelization of the Pub-Sub paradigm by using colored Petri Nets;
• verification based on the previous tools (CPN tools and ASK-CTL library).

B. A colored Petri Net model for the Pub-Sub paradigm

The colored Petri net model in Figure 2 modelizes the Pub-sub paradigm. It introduces an initial state and 10 components with 10 events. Each component can be a publisher (represented by EP for EventPublished on the figure), or a subscriber (represented by ES for EventSubscribed on the figure) or both.

This model is compliant with the publish-subscribe protocol, since a component can publish an event as many times as it wants, it can also subscribe to an event as many times as it wants. An event can be issued by one or more components and of course one or more components can subscribe to this event.

Published events are saved in a directory that is modeled here by the state Registry. When a component subscribes to an event E it goes to state subscriber waiting (WaitingSubscriber). Once the event is published, the transition Notify can be fired. A condition should be checked when we fire the transition Notify: "a component cannot subscribe to the event it published", which was modeled using the guard \[S <> P\].

CPN Tools generates a state space report which is a textual file containing answers to a set of standard behavioral properties such as information about the boundedness, home, liveness, and fairness properties. Practical use has shown that the properties of a system which are investigated first are very often contained in this set of system independent properties. The state space report can be produced totally automatically, and by studying it the user gets a rough idea, as to whether the CPN model works as expected or the modelized system contains errors [4].
part of the BonjourGrid protocol. We do not cover all the BonjourGrid protocol for the sake of simplicity. BonjourGrid main events are "Idle worker", "RequestForParticipation", and "FreeCoordinator".

The Petri Net for the Pub-Sub substrate defines 3 connectors, namely "Publish", "Subscribe" and "Notify" transitions. The first connector stands for publishing an event $e_p$ from a machine $c$, the second one is for subscribing an event $e_s$ coming from a machine $c$, and the third one is for notifying events to interested machines. Theses 3 connectors, parameterized by a pair (event,machine), are represented namely by places "Component" and "Registry" on Figure 5.

We propose now to show how the BonjourGrid protocol is interfaced according to these 3 connectors. The challenge is to consider a 'black box' (the Petri net for the Pub-Sub model which is represented in blue color on Figure 5) that cannot be modified and to try to specify the behavior of BonjourGrid as external events of the black box. By doing this we exhibit a general methodology.

So, the main idea is to plug the BonjourGrid protocol on top of the Pub-Sub protocol. The later is mainly presented by the cycle: "publication", "subscription" and "notification". BonjourGrid elements are plugged on inputs and outputs of this cycle. Firstly, a component may submit an application $a$ and publish a "RequestForParticipation" event associated to that application. In parallel, other components may subscribe to that "RequestForParticipation" event. A coordination starts when subscribed machines are notified by their corresponding events. In this step, idle machines become Workers.

Workers may subscribe into the "FreeCoordinator" event that would allow to release them. When coordination is finished, the coordinator component publishes the "FreeCoordinator" event then subscribed workers are notified. They are released and return to the Idle state.

So, to resume, in the beginning an "idle service" is published to specify that a resource is in the Idle state, the Web service specifies also the characteristics of the resource. Then, when an application would be submitted, a "coordinator service" is published to specify that a resource has become coordinator for a given application. A request for participation is submitted in order to discover eventual participants (among idle resources). Resources are discovered and invoked based on their characteristics. Selected resources that confirm their participation, publish a "worker service". When coordination is finished, workers are released and they publish again an "idle service".

Here, we have successfully achieved our second aim by formally modeling the BonjourGrid protocol built on top of that Pub-Sub Petri Net. Using CPN Tools we then performed manual simulations of the net to gain more confidence in our model. During this first step no straightforward problems could be discovered. The next step was naturally to perform an exhaustive simulation exploring all the possible states of the system, i.e., its state space. Since our model is scalable with respect to the number of participating network nodes, we analyzed several configurations with a number of participants ranging from 2 to 7.

Table I provides a sample of the state space report provided by CPN Tools: the number of reachable States in the state space, the number of Arcs in the state space, and the Time taken by CPN Tools to build the state space. The search time was limited to 5 hours, i.e., 18,000 seconds. Hence, the data reported for 7 participants are incomplete and represent only a lower approximation.

Our model is naturally subject to the well known state explosion problem in that the number of reachable states grows exponentially with respect to the number of participants. However, the analysis of these small configurations gave us more confidence in the BonjourGrid system especially considering the following facts:

- We have not found any deadlock states (i.e., states that do not admit executable transitions). The absence of such a state is obviously required in our context.
- All possible transitions are executable. Hence, our specification seems to be correct from the perspective of event triggering: all possible events can eventually happen.
- All state spaces built are composed of a single strongly connected component. It seems therefore impossible to be trapped in a specific or undesired system configuration. This liveness property is indeed a crucial prerequisite for our system.

Although this preliminary analysis yields satisfactory results it is a step towards the formal verification of our protocol. The next step is now to analyze larger configurations which means the ability to circumvent the state explosion problem. Two perspectives are thus privileged. First, we intend to switch to a dedicated verification platform. CPN Tools is indeed very convenient from a modeling and simulation
Table I
STATE SPACE REPORT FOR SEVERAL CONFIGURATIONS

<table>
<thead>
<tr>
<th>Participating nodes</th>
<th>States</th>
<th>Arcs</th>
<th>Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>35</td>
<td>108</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>215</td>
<td>1,005</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1,295</td>
<td>8,204</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>7,775</td>
<td>62,635</td>
<td>45</td>
</tr>
<tr>
<td>6</td>
<td>46,655</td>
<td>458,778</td>
<td>5,956</td>
</tr>
<tr>
<td>7</td>
<td>&gt; 300,000</td>
<td>&gt; 3,000,000</td>
<td>&gt; 18,000</td>
</tr>
</tbody>
</table>

These works concern the problem of constructing a system as perfect as possible in terms of scalability, efficiency and safety. But they do not focus enough on the problem of the formal analysis of the accuracy of such systems. In this sense, some research has already paid attention to the formal verification of publish-subscribe systems [12]–[14].

In [12], [13], the authors propose an approach for modeling and validating Pub-Sub systems. This approach is based on an architecture of components that react to events. In these works, the components are specified with state-transition diagrams of UML. Formal verification is achieved through model checking (using SPIN). But instead of using the formulas of linear temporal logic (LTL) of SPIN, the authors have interpreted the properties as automata. According to them, this will represent more complex properties needed to validate the modeled system.

Although our modeling approach is also based on component reacting on events, we do prefer to enjoy the advantages of temporal logic for the formal verification step, in particular by using the ASK-CTL library.

In [14], the authors describe a generic framework dedicated to modeling and formal verification of publish-subscribe mechanisms. Their system is based on a model of state machine providing management of events during the execution of the publication-subscription protocol. The framework takes as input a set of components and a set of properties of publish-subscribe mechanism. The matching of the two sets is subsequently validated using model checking tools.
This system is regarded primarily as a generic framework in which there is always the risk of not providing a model and an audit tailored to each specific case of publish-subscribe mechanism. Our approach is successful modeling and formal verification the most suitable for BonjourGrid while isolating the publish-subscribe mechanism.

In [17], [18], the authors, motivated by the benefits of formal analysis, build coordination protocol for a formal model using colored Petri nets. To evaluate the accuracy of their model and as a result of their protocol, they checked the behavioral properties and formally implemented a mechanism of CTL model checking. Our work is built around the protocol proposed by the authors. From our side, we also capitalize on the use of colored Petri nets and CTL logics.

In [15], the authors present a new approach to modeling and formal verification, this time dedicated to software components. Their methodology is based on a software architecture-driven and the reuse of Petri nets models. Their contribution is rather a new approach for visual composition, formal verification and validation of software systems. The work is built primarily around software components; we expect to benefit most ideas proposed around the tasks of formal verification and validation.

VI. APPLICATION ISSUES

The Nexedi company created a distributed Cloud Computing operating system called SlapOS. SlapOS is an OW2 project. It is already used in production in France. SlapOS unifies IaaS, PaaS and SaaS through a simple API inspired by grid computing and by open source ERP5. SlapOS is currently a centralized component based on a sequential algorithm. Our team is currently working with Nexedi in the Resilience project in order to distribute the SlapOS services.

The application area is as follows. Users on Internet host 2 servers and the Vifib 1 company pay for the Internet subscription. The deal for the user is to accept to host some services on the 2 PCs. In the Resilience project, services are Linux packages for different Linux distributions. A Ubuntu user requests for the building of a new software. He publishes a request for the availability of libraries X, Y and Z. Then a distributed compile step occurs and the new software becomes available on the Internet under the form of a SaaS. The BonjourGrid protocol serves to coordinate the activity of multiple users requesting for building new services.

Because SlapOS includes billing support, it is very useful to turn any software into SaaS or PaaS in a matter of hours. It also opens the mind of people toward business aspects of Cloud. Because SlapOS is distributed in nature, it is much more resilient than traditional Cloud such as Amazon EC2 or PaaS such as Salesforce. We have already seen a lot of interest in SlapOS by large companies in Korea, Germany and Japan because we create a cloud of a new type. The formalization of Services done in this paper is a good step ahead to better understand the complexity of the SlapOS software.

To summarize, Grid technologies used in the Resilience project are basically the coordination of computing resources and applications inspired by BonjourGrid protocol. Web-technologies are also used for mastering the business level (accounting) and we have chosen the Tiolive technology, an open source solution for Communication (Email, Telephone, Chat), Backoffice (Contacts, Documents, Accounting, ERP, CRM) and e-Business (Web Site, e-Commerce).

VII. CONCLUSION AND PERSPECTIVES

In this paper, we introduced the context of our work around the coordination of resources using the publish-subscribe paradigm and the usefulness of modeling and formal verification of such a specific mechanism for the BonjourGrid system dedicated to the management of multiple instances of Desktop Grid middleware.

We recall that our work is related to reverse engineering. We do not aim to prove that our ‘last’ model complies with the implementation but we are abstracting mechanisms for the publish-subscribe paradigm.

This work is a step before starting the study of the scaling of BonjourGrid from a formal point of view. We want to say that the next task will be to inject more of computing resources in the system and to show that the protocol is correct. Furthermore, this effort has been consolidated in this paper with another facet of the problem, namely the definition of a mechanism for composition of the basic scheme introduced in this paper with any protocol that is written with the publish-subscribe paradigm in mind. Recall that BonjourGrid is only a case study that allows us to discuss the problems and devise solutions. Finally, once the complete model made, we will seek to define a device for automatic code generation from modeling. The target language may take the form of popular language on the Internet such as XMPP 2 or Bonjour (more exactly Wide Area Bonjour). Many libraries support the publish-subscribe mechanism are being deployed (e.g. Jetty 3), Hookbox 4 including the forthcoming HTML5 standard 5.

We thank Kais Klai, associate professor at LIPN for his advices and remarks during this work as well as Hassna Louadah, former LIPN intern, for her work about Petri Nets.

REFERENCES


1URL: http://www.vifib.com

2URL: http://xmpp.org/

3URL: http://jetty.codehaus.org/jetty/

4URL: http://hookbox.org

5URL: http://www.w3.org/TR/html5/


[26] Boinc. URL: http://boinc.berkeley.edu/
