AGRADC: An Architecture for Autonomous Deployment and Configuration of Grid Computing Applications

Luciano Paschoal Gasparý, Weverton Luís da Costa Cordeiro, Sidnei Roberto Selzler Franco, Marinho Pilla Barcellos, Gerson Geraldo Homrich Cavalheiro

1Institute of Informatics
Federal University of Rio Grande do Sul, Brazil
2Postgraduate Programme on Applied Computing
University of Vale do Rio dos Sinos, Brazil
3Institute of Physics and Mathematics
Federal University of Pelotas, Brazil

{paschoal, weverton.cordeiro}@inf.ufrgs.br, sidneif@turing.unisinos.br, marinho@acm.org, gerson.cavalheiro@ufpel.edu.br

Abstract

Deployment and configuration of grid computing applications are exhaustive and error-prone tasks, representing a weak link of the lifecycle of grid applications. To address the problem, this paper proposes AGRADC, an architecture to instantiate grid applications on demand, which incorporates features from the Autonomic Computing paradigm. This architecture improves the grid application development process, providing tools to define (a) a deployment flow, respecting dependencies among components that compose the application, (b) configuration parameters, and (c) actions to be executed when adverse situations like faults arise. The result of this process, materialized in the form of a set of documents, is delivered to an instantiation engine, which autonomously conducts and manages the deployment and configuration process.

1. Introduction

Grid computing applications have expanded gradually and consistently over the last years. Grid computing infrastructures are built from the sharing of distributed resources [7], and have as main objectives the provision of higher performance for the solution of computing intensive problems, while abstracting the computing infrastructure to a single and powerful computer, among others [10]. With the evolution of grid technologies, one can notice that they incorporate new functionalities and enhance the services offered. One of the advances pursued is the automation of tasks associated with grid middleware, enabling the deployment, configuration, and management of resources and services in an easier way.

To execute a large scale application – with a large number of distributed resources and services – it is essential that the environment be properly deployed and configured, thus meeting the application's needs. This task tends to be exhaustive and subject to errors, and can be regarded as the “weak link” of the lifecycle of a grid application development process. The problem is aggravated by the desire to execute a wider range of increasingly specific applications, requiring the deployment and configuration of rather particular frameworks. Clearly, carry out such a cumbersome process manually is not a suitable approach.

In order to deal with the problem, this paper proposes AGRADC (Autonomic Grid Application Deployment & Configuration Architecture), an architecture to instantiate grid applications that allows the necessary infrastructure to be deployed, configured, and managed upon demand. In the proposed architecture, the available tools allow the developer to define (a) a deployment flow, respecting dependencies among components that comprise the application, (b) configuration parameters, and (c) actions to be executed in adverse situations such as faults. The result of this process, materialized in the form of a set of text files, is delivered to an instantiation engine, which then autonomously conducts and manages the deployment and configuration process.
The main contributions of the present paper are twofold. First, it proposes a novel service to conduct the instantiation process of grid applications – the instantiation engine – which incorporates features of the Autonomic Computing paradigm. The engine is able to perform circumventing (configuration) procedures in response to problems found along the process. Second, it introduces a software architecture fully compliant with related specifications, namely the CDDLM (Configuration Description, Deployment, and Lifecycle Management) specification [1].

The remaining of the paper is organized as follows: Section 2 discusses related work. Section 3 presents a conceptual overview of AGrADC and describes its components. Section 4 deals with the prototype development as a proof-of-concept. An experimental assessment of the architecture is presented in Section 5. Section 6 closes the paper with concluding remarks and prospective directions for further work.

2. Related work

This section discusses related work on area of dynamic deployment and configuration of services and applications, by first addressing the subject in general, and then in the context of grid applications.

In [16] authors present a comparative analysis of three approaches for the deployment of services: scripts, languages, and models. The paper assesses their scalability, complexity, and expressivity, then concludes that the most promising solutions are based on models, since they deal well with scalability and complexity (which are crucial aspects for large scale applications). The dynamic deployment of services was also investigated in contexts other than grid computing, such as J2EE [13] and Web Services [2]. Rauch et al. proposed in [12] the cloning of system partitions for the deployment of software, an admittedly expensive computer approach because it requires the replacement of the whole image of the operating system. Following the same direction, Keahey et al. proposed in [5] the use of virtual machine technology to deploy environments with diverse services and configurations.

The first related work focusing on grid computing was presented in [15, 18, 11]. The authors in [15] proposed a safe solution to the dynamic deployment of WSRF services (Web Services Resource Framework). In [18] authors introduced an architecture based on Tomcat to incorporate the functionality of dynamic service deployment to Globus Toolkit version 3 (GT3), with the restriction to operate at a container level. The third paper, [11], suggested changes in the core of Tomcat to make its deployment lighter and at a service level. As a result, the authors obtained an infrastructure able to deal with highly dynamic environments without compromising the availability and fulfillment of requests. Parallel to these research initiatives, the GGF (Global Grid Forum) has made efforts to standardize the CDDLM specification [1], which constitutes a model based approach for configuration, deployment and management of grid application lifecycle.

In summary, the advances made in the area concentrate on mechanisms meant to interact individually with target stations, providing a systematic form (and interoperating, as the case of CDDLM) in order to instantiate components of a grid application. However, existing approaches do not show how to orchestrate the deployment of a complete application involving several distributed resources and services. Further, they do not address the challenge of coping with adverse situations such as configuration errors and the unavailability of target stations. These two aspects are the focus of the architecture presented in this paper.

3. AGrADC architecture

This section describes the architecture proposed for the instantiation of grid computing applications. First, architecture elements are discussed, as well as the flows of information among them. Next, the description language of the instantiation scenarios is presented. At last, the operation of the instantiation engine is specified, highlighting the functionalities introduced in order to provide self-configuration and self-recovery in the process of application instantiation.

3.1. Conceptual elements and their interactions

AGrADC is composed of four conceptual elements: (a) management application; (b) component repository; (c) instantiation engine; and (d) instantiation services. Figure 1 gives an overview of the architecture instantiated in a grid infrastructure comprised of three administrative domains, namely A, B, and C.

The management application allows the developer to define the application components, specify the steps for deployment and configuration, and request the instantiation of the application in the grid infrastructure. The components and deployment scripts are stored in a repository. The instantiation engine receives invocations from the management application and orchestrates the instantiation of the environment requested. Finally, instantiation services – lodged in all grid stations – supply interfaces enabling them to execute deployment, configuration, and management of components.
The interaction among architecture elements takes place as follows. First of all, using the CDL language, the developer defines the components that participate in the application (e.g., database, http server, and grid service), the deployment sequence to be respected, and the configuration parameters (flow 1 in Figure 1). The result of this phase is the generation of a set of CDL files and components that are stored in the repository (flow 2). The next phase is the application instantiation request to the instantiation engine (3), which is accompanied by the identifier of the application description file location. Upon receiving the request, the instantiation engine retrieves this file (4), interprets it, and initiates the instantiation process.

As the engine identifies each component in the description file, it retrieves the corresponding component from the repository (flow 4 in Figure 1). Based on the information provided by the grid scheduler (5) with relation to the resources available, the engine decides in which stations each component will be instantiated and interacts with the instantiation services of the selected stations (6). The interaction foresees operations for the deployment, configuration, and management of the components. The result of these operations – success or failure – is informed to the engine via notifications generated by the instantiation services (7). Based on policies expressed at the engine (explained in Section 3.3), it automatically reacts advancing the instantiation process or executing a circumventing procedure. Finally, if the instantiation process runs successfully, the execution environment of the grid is ready to execute the application.

The representation of instantiation scenarios is crucial for the architecture, given the diversity of applications, and above all, the execution requirements demanded by the applications. Without such scenarios (and the infrastructure to dynamically instantiate them), an application would be either limited to the services previously available at the grid stations or would demand a great operational (and ad-hoc) effort to prepare the desired environment. The following section deals with the functionalities of the CDL language for the specification of the mentioned scenarios.

3.2. Representation of instantiation scenarios

The description of an instantiation scenario is done through the use of the configuration description language (CDL) [17], and consists of the identification of the components and their configuration parameters, as well as the determination of the deployment flow to be respected. To describe the functionalities of the language, an example of a scenario is presented in Figure 2. The scenario is comprised of three components: (a) database (Hsqldb); (b) web-based Java application server (Tomcat); and (c) grid application (GridApp). The example illustrates the dependencies among such components, indicating that Hsqldb and Tomcat may be instantiated in parallel, and that only at the end of this step GridApp can be instantiated.

The specification is organized in two blocks: definition and configuration of components (cdl:configuration), and definition of the order of components instantiation (cdl:system), as follows. In the definition and configuration block (Figure 2, lines 4-28) the components of the grid application are defined, which will be later associated with the resources available. The components are defined using XML blocks, and configuration parameters are associated to each one. Besides representing services (e.g., Tomcat, lines 10-15, and Hsqldb, lines 16-21), components may indicate parameters to be shared by other components, which is the case of DBCConnection, lines 5-9).
Each component has a particular set of parameters, according to its needs. There are three ways to associate values to a parameter: (a) defining the value at the moment of parameter declaration (e.g., line 13, <port>8080</port>); (b) informing that the value will be defined at the moment of instantiation (e.g., line 12, <hostname cdl:lazy="true"/>); or (c) determining that the parameter value must be inherited from the indicated reference (e.g., line 38, <dbport cdl:ref="Hsqldb:/port"/>).

In the block that defines the component instantiation order, the ordering is determined by the hierarchy in which components are declared in the block cd:system (Figure 2, lines 30-51). The hierarchy among components is established through the following rules:

- **Sequence**: indicates that components should be instantiated according to lexical order. For example, it is specified in lines 31-45 that the instantiation of components Tomcat and Hsqldb must be completed before the instantiation process of GridApp is initiated.

- **Inverse**: it is employed to indicate the order of removal of components in a process of undeploy (lines 46-50).

- **Flow**: used when no order among components is required, that is, there are no dependencies. In Figure 2 (lines 32-41), Tomcat and Hsqldb may be instantiated in any order.

- **Switch**: it allows for the instantiation flow to be altered depending on the values associated with condition variables (not used in the example).

### 3.3. Instantiation engine

The instantiation engine provides a service that orchestrates the instantiation of the software infrastructure needed for the execution of grid applications. The engine spares human intervention in the deployment, configuration, and management of components that compose the applications, bridging a critical gap not addressed in previous related work.

Invoked by the management application, the engine offers two methods: Deploy and Undeploy. The identifier of the location of the description file of the application to be deployed accompanies the invocation of the former while the latter is followed by the EPR (End-Point Reference) of the application to be removed. On the other hand, the instantiation engine employs the methods supplied by the Deployment API [8] to interact with the instantiation services.

The installation of an application consists of the instantiation of its components, respecting the order informed in the description file. The instantiation of each component, in its turn, is executed respecting steps defined in the CDDLM specification. In order to conduct this complex process in a controlled manner, the engine maintains a state machine for each component, reflecting the current situation of the corresponding instantiation.

Figure 3 illustrates a snapshot of the instantiation process of the application previously introduced in Section 3.2. Notice that three state machines are employed, and the one controlling the instantiation of Hsqldb is highlighted. The referred state machine was already uploaded in the target station and configured. At the moment the engine has just invoked the run() method, it asynchronously receives a notification message of type RunFault, informing that the operation was not successful. Simultaneous to the instantiation attempt of the Hsqldb component, the engine executes the same procedure for the Tomcat component, since the CDDL specification authorizes such parallelism.
The current state of components and the notification messages received may be used to express how the instantiation engine should behave in adverse situations. Such behavior—specified by the application developer or through action policies [6]—confers the architecture features of Autonomic Computing, such as self-configuration and self-recovery.

Action policies dictate actions that must be executed whenever the system is found in a certain state, represented in the form ON (State) IF (Condition) THEN (Action). Policies must cover each relevant state defining actions according to pre-established conditions, so that the instantiation engine presents a rational behavior. Figure 4 illustrates a set of policies defined to govern the engine behavior. These policies cover the four main lifecycle states of a component: undefined (lines 1-19), instantiated (21-30), initialized (32-41), and running (43-52). For instance, when an engine tries to deploy a component (undefined state), some situations may arise:

- the scheduler may not have stations to offer (NonAvailableHost), condition which makes the engine interrupt the deployment component (terminate()) and, as a consequence, the instantiation of all applications;
- the engine may not be able to contact the station determined by the scheduler (HostUnreachable), condition which makes the engine attempt to deploy the component in the same station or in an alternative one;
- the engine may face difficulties interacting with the instantiation service which is being executed in the station (SoapFault), condition which induces the engine to repeat the attempt.

```
ON UndefinedState {
  IF (event.NonAvailableHost)
  THEN terminate()
  IF (event.HostUnreachable && event.HostUnreachable:ContFailed < 3)
    THEN create()
  IF (event.HostUnreachable && event.HostUnreachable:ContFailed >= 3)
    THEN terminate()
  IF (event.SoapFault && event.SoapFault:ContFailed < 3)
    THEN create()
  IF (event.SoapFault && event.SoapFault:ContFailed >= 3)
    THEN terminate()
  ...
}
ON InstantiatedState {
  IF (event.ConfigurationFault && event.ConfigurationFault:ContFailed < 3)
    THEN initialize()
  IF (event.ConfigurationFault && event.ConfigurationFault:ContFailed >= 3)
    THEN terminate()
  ...
}
ON InitializedState {
  IF (event.RunFault && event.RunFault:ContFailed < 3)
    THEN run()
  IF (event.RunFault && event.RunFault:ContFailed >= 3)
    THEN terminate()
  ...
}
ON RunningState {
  IF (event.TestFault && event.TestFault:ContFailed < 3)
    THEN ping()
  IF (event.TestFault && event.TestFault:ContFailed >= 3)
    THEN terminate()
  ...
}
```

Figure 3. State machines maintained by the instantiation engine

Figure 4. Policy representation
4. Implementation

A prototype of the proposed architecture was developed with the objective of assessing its technical feasibility. This section describes features related to its implementation, including information of each element that makes up the architecture.

The management application was implemented in Java, and allows one to describe the components that comprise a grid application (and its dependencies) and to verify, with the support of JDOM API, the correctness of the resulting XML-CDL description files. The management application also allows the generation of GAR (Grid Archive) component packages – compatible with Globus Toolkit version 4 [4], identifying the instantiation engine, as well as invoking and accompanying the instantiation of an application. The description files and the components are stored on a web application server (Apache Tomcat).

The instantiation services are web services that implement the interfaces and the mechanisms defined in the CDDLM specification. As the aim of this work was to investigate techniques to orchestrate the instantiation of complete applications, comprising resources and distributed services, we chose to propose a solution aligned to the referred specification. However, we temporarily adopted a deployment service supplied by Globus Toolkit version 4, in addition to functionalities that allow component deployment, configuration, and management, emulating the CDDLM specification.

Finally, the instantiation engine has as its main feature the conformance with the CDDLM specification. The engine, also developed in Java, is a web service that implements the state machine mechanism shown in Figure 3. The interface exposed to the management application to invoke and accompany the instantiation of applications follows the WSDM (Web Services Distributed Management) specification [3], and was developed using the Muse framework v2.0 [9]. The JDOM API is employed to read, interpret, and store in memory the application descriptions. To interact with the instantiation services, the current prototype version uses the API supplied by Globus Toolkit version 4 – and not the Deployment API – for the reasons mentioned previously. The engine also implements a listener to receive notification messages generated by the instantiation services.

The action policies are represented in the format described in Section 3.3 and stored in a configuration file. This feature allows the policies to be created, changed, and dynamically removed without having to recompile the instantiation engine.

5. Experimental evaluation

This section discusses an experimental evaluation carried out with the architecture prototype. AGRADC was deployed in a real grid environment (executing Globus version 4) in order to evaluate its ability to instantiate applications on demand and react autonomously to induced adverse situations.

The application used to evaluate the architecture was based on the application shown in Figure 2. The description file of the application was adapted by determining the instantiation of the Hsqldb component in a station and the Tomcat and GridApp components in three other distinct stations. GridApp consists of a bag-of-tasks application to perform computations on a deterministic model of intracellular viral kinetics [14]. In the described scenario, the three instances of GridApp were configured to use the same Hsqldb database. In order to execute the experiment, we employed five stations: four to host the applications, and one for the management application, the component repository and the instantiation engine.

The instantiation of the defined application was invoked from the management application. Throughout the process, we observed that the architecture behaved exactly as it was expected, being faithful to what was specified in the configuration file and successfully concluding the instantiation. The execution of the entire process was verified by (a) capture and analysis of generated traffic, (b) characterization of entries in log files, (c) verification of installed components, and (d) well-succeeded invocation of the application.

In another experiment, the application description was relaxed, allowing the instantiation of the application to be carried out in stations recommended by the grid scheduler (using the lazy constructor). The scheduler, on its turn, was configured to suggest a sequence of known stations. The first and third stations of the list did not exist. When, by the second time, the
instantiation of the application was invoked, it was possible to verify that the architecture made use of the defined policies to conduct the engine behavior. When notification messages HostUnreachable were received, the engine autonomously asked the scheduler for new station recommendations and, even facing these adversities, it concluded that the instantiation of the application was successful.

6. Conclusion and future work

This paper presented AGRADC, a software architecture for deployment, configuration, and lifecycle management of grid computing applications. In contrast to previous investigations, we advance further towards a solution to orchestrate the instantiation of complex grid computing applications. For such, we propose a ground-breaking service whose role has been performed until now by human operators. Allowing the specification of policies to regulate its behavior, the service is able to execute configuration procedures to deal with problems faced throughout the instantiation process, providing support for auto-configuration and auto-recovery. A prototypical implementation that is able to conduct the instantiation of an application was used as a proof-of-concept. The results obtained suggest that the proposed architecture can be employed in real scenarios.

Note that the focus of the present paper lies in the installation and configuration of the software infrastructure required to deploy an application, including its binaries. The invocation of the application is beyond the scope of the paper, and therefore was not addressed. It might be carried out by grid workflow tools, whose main objective is to invoke, in a coordinated way, the application components.

As future work, we intend to (a) develop a broader set of experiments to measure the times involved when instantiating applications in larger scale scenarios, and (b) explore alternatives concerning the representation of autonomous behavior, such as objective policies and utility functions, aiming at providing a higher level of formalism for the grid infrastructure manager.

References


Conferences containing my papers

Only papers from conferences that have not ended are shown, but you can also see all papers. Dates listed are deadlines for submitting manuscripts for registered papers. You can only upload papers that have at least one author. All deadlines are America/New_York time.

<table>
<thead>
<tr>
<th>Conference</th>
<th>Paper title (details)</th>
<th>Manuscript deadline</th>
<th>Add and delete authors</th>
<th>Upload</th>
<th>Copyright</th>
<th>Visa letter</th>
<th>Withdraw</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2P’09</td>
<td>Choking Polluters in BitTorrent File Sharing Communities</td>
<td>past</td>
<td>+</td>
<td>--</td>
<td>--</td>
<td></td>
<td>![x]</td>
</tr>
<tr>
<td>ISCC’09</td>
<td>AGRADC: An Architecture for Autonomous Deployment and Configuration of Grid Computing Applications</td>
<td>past</td>
<td>+</td>
<td>--</td>
<td>--</td>
<td></td>
<td>![x]</td>
</tr>
</tbody>
</table>

pending (no manuscript), active (under review), discuss, withdrawn, rejected, minor revision, major revision, accepted, published

http://edas.info/listConferencesAuthor.php
IEEE Symposium on Computers and Communications (ISCC'09)
July 5 - 8, 2009, Sousse, Tunisia

UPDATE: The author instructions have been updated with the final paper submission instructions.
New: Authors must complete the IEEE Copyright form available at http://www.ieee.org/web/publications/rights/copyrightmain.html and send the completed form to Dr. Adel Elmaghraby by email (adel@louisville.edu) or fax (1-502-852-4713).

About ISCC

The IEEE symposium on Computers and Communications started in 1995 in response to the growing interaction between the fields of computers and communications. ISCC takes place annually, typically in the beginning of July. Symposium locations include Egypt, Greece, France, Tunis, and Italy with plans for other sites in the Mediterranean region.

Sessions will include tutorials in addition to presentations on new research results. Papers describing original work are invited in any of the computers and communications areas described in the call for papers will be considered. Accepted papers will be published in Proceedings of IEEE ISCC 2009 and in the IEEE digital library. Acceptance will be based on quality, relevance, and originality. Best Paper Award and Best Simulation Paper Award will be presented.

Sponsors

IEEE
IEEE Communications Society
IEEE Computer Society

Supporters

AT&T
Tunisair
Transporteur officiel