Towards Deployments Contracts in Large Scale Clusters & Desktop Grids

Françoise Baude, Denis Caromel, Alexandre Di Costanzo, Christian Delbe, Mario Leyton

Towards Deployment Contracts in Large Scale Clusters & Desktop Grids

Françoise Baude, Denis Caromel, Alexandre di Costanzo, Christian Delbé, and Mario Leyton

INRIA Sophia - I3S - CNRS - Université de Nice Sophia Antipolis
INRIA, 2004 Route des Lucioles, BP 93
F-06902 Sophia Antipolis Cedex, France
First.Last@sophia.inria.fr

Abstract

While many dream and talk about Service Level Agreement (SLA) and Quality of Service (QoS) for Service Oriented Architectures (SOA), the practical reality of Grid computing is still far from providing effective techniques enabling such contractual agreements.

Towards this goal, this paper provides an overview of the techniques offered by ProActive to set and use contractual agreements. Based on the identification of roles, application developer, infrastructure manager, application user, the actors of a Grid environment can specify what is required or what is provided at various levels. The results are both flexibility and adaptability, matching the application constraints and the environment characteristics with various techniques.

1 Introduction

Traditionally the programming and execution of a distributed application has been handled by a single individual. The same individual programs the application, configures the resources, and performs the execution of the application on the resources. Nevertheless, the increasing sophistication and complexity of distributed applications and resource infrastructures has led to the specialization of expert roles.

On one side we find the *developers* of distributed applications, and on the other side the *infrastructure managers* who maintain resources such as Desktop machines, Servers, Cluster and Grids. Between both of these expert roles we can identify the *users* who take the applications and execute them on a distributed infrastructure to solve their needs.

The separation of these roles raises the issue of how programmers and infrastructure experts relate to solve the needs of the users. The complexity of this issue is emphasized when considering that the programmers and infrastructure managers are unacquainted. That is to say, that a user has to deploy and execute an unfamiliar application on unfamiliar resources without knowing the requirements of either.

In this paper we address the issue of reaching contractual agreement between distributed applications and resource infrastructures during deployment. Throughout this paper we propose the deployment time as the key moment to reach an agreement between the infrastructure and the application. Using the contracts, users are able to perform the deployment and execution of an unfamiliar application on unfamiliar resources effortlessly.

Section 2 presents the deployment principles and architecture of the ProActive middleware, including the deployment of applications on JVMs managed by the peer-to-peer infrastructure. Section 3 introduces several ways to setup contracts and agreements between the three major roles in Grid computing: application developer, infrastructure manager, and application users. Section 4 shows an example of how a contract can be reached between a master-slave application and an infrastructure. Section 5 discusses the related work. Section 6 summarizes the concluding remarks and presents the future work.

2 ProActive and Deployment

ProActive is a Grid programming middleware which provides, among others, a Grid infrastructure abstraction using deployment descriptors [3], and an active object model using transparent futures [7].

Active objects are remotely accessible via method invocations, automatically stored in a queue of pending requests. Each active objects has its own thread of control and is granted the ability to decide in which order incoming method calls are served (FIFO by default). Method calls

1-4244-0910-1/07/$20.00

on active objects are asynchronous with automatic synchronization (including a rendezvous). This is achieved using automatic *future objects* as a result of remote methods calls, and synchronization is handled by a mechanism known as *wait-by-necessity* [5].

2.1 Deployment Framework

The ProActive Deployment Framework completely extracts all infrastructure details from the source code [3].

The first key principle is to *fully* eliminate from the source code the following elements:

- Machine names
- Creation protocols
- Registry and lookup protocols
- Communication protocols

The goal of the deployment framework is to deploy any application anywhere without having to modify the source code. The resources acquired through the deployment process are called *nodes*. Nodes are the containers of active objects, and are created by starting the ProActive runtime on the infrastructure resources.

The second key principle is the capability to abstractly describe an application, or part of it, in terms of its conceptual activities.

To summarize, in order to abstract away the underlying execution platform, and to allow a *source-independent deployment* a framework has to provide the following elements:

- An abstract description of the distributed entities of a parallel program or component.
- An external mapping of those entities to real *machines*, using actual *creation, registry*, and *lookup* protocols.

To answer these principles, the ProActive deployment framework relies on XML deployment *descriptors* to hold the infrastructure configuration. Descriptors introduce the notion of *virtual-node*:

- A virtual-node is identified as a name (a simple string).
- A virtual-node is used in a program source.
- A virtual-node, after deployment, is mapped to one or to a set of *actual ProActive Nodes*, following the mapping defined in an XML descriptor file.

A virtual-node is a concept of a distributed program or component, while a node is a deployment concept that hosts active objects. There is a correspondence between virtual-nodes and nodes which is the relation created in the deployment descriptor: the mapping. This mapping is specified in the deployment descriptor. There is no direct mapping between virtual-nodes and active objects: the active objects are deployed by the application onto nodes related with a virtual-node. By definition, the following operations can be configured in the deployment descriptor:

- The mapping of virtual-nodes to nodes and to Java Virtual Machines.
- The mechanism (protocol) to create or to acquire Java Virtual Machines, such as: local, ssh, gsissh, rsh, rlogin, lsf, pbs, sun grid engine, oar, prun, globus (GT2, GT3 and GT4), unicore, glite, and nordugrid-arc.
- The mechanism (protocol) to register or to lookup Java Virtual Machines, such as: RMI, HTTP, RMI-ssh, Ibis, and SOAP.

In the context of the ProActive middleware, nodes designate resources of an infrastructure. They can be created or acquired. The deployment framework is responsible for providing the nodes, mapped to the virtual-nodes, to the application. Nodes may be created using remote connection and creation protocols. Nodes may also be acquired through lookup protocols, which notably enable access to the ProActive Peer-to-Peer infrastructure as explained in Section 2.2.

2.2 Principles: Peer-to-Peer

We propose in [9] a P2P infrastructure of computational nodes for distributed communicating applications. The proposed P2P infrastructure is an unstructured P2P network, such as Gnutella [14]. In contrast to others P2P approaches for computing, which are usually hierarchical or master-salve, our approach is original in the way that an unstructured P2P network commonly used for file sharing can be also used for computing.

The P2P infrastructure has three main characteristics. First, the infrastructure is decentralized and completely *self-organized*. Second, it is flexible, thanks to parameters for adapting the infrastructure to the location where it is deployed. Finally, the infrastructure is portable since it is built on top of Java Virtual Machines (JVMs). Thus, the infrastructure provides an overlay network for sharing JVMs.

The infrastructure allows applications to transparently and easily obtain computational resources from grids composed of both clusters and desktops. The application deployment burden is eased by a seamless link between applications and the infrastructure. This link allows applications

to be communicating, and to manage the resources' volatility. The infrastructure also provides large scale grids for computations that would take months to achieve on clusters.

The proposed P2P infrastructure is an unstructured P2P network. Therefore, the infrastructure resource query mechanism is similar to the Gnutella communication system, which is based on the Breadth-First Search algorithm (BFS). The system is message-based with application-level routing. Messages are forwarded to each acquaintance, and if the message has already been received (looped), then it is dropped.

Applications use the P2P infrastructure as a pool of resources. The main problem for applications to use those resources is that resources are returned via a best-effort mechanism; there are no guaranties of that the number requested resources can be satisfied. Recently we have improved the resource query mechanism by adding the possibility of filtering requested resources on three operating system properties: the system name, version, and the system architecture. Those properties are provided by the Java system properties. The filtering mechanism is indeed done by peers of the infrastructure; when a peer gets a resources query, first checks if it is free and then checks OS property constraints.

3 Contracts and Agreements

The three roles that we have identified: *programmers*, *infrastructure managers*, and *users* are related with the *applications*, *descriptors*, and *deployment/execution* respectively. The programmer writes the application, the infrastructure manager writes the deployment descriptor, and the user performs the deployment and execution of the application on the infrastructure using the deployment descriptor.

To begin with, application and descriptor must agree on the name of the virtual-node. Nevertheless, the virtual-node name is not the only agreement problem that the application and descriptor have. More importantly, the application and descriptor must agree on the required and provided technical services such as: fault-tolerance, load-balancing, etc.

Modifying the application or the descriptor can be a pain-full task, specially if we consider that the user may not be the author of either. To complicate things further, the application source may not even be available for inspecting the requirements and performing modifications. Figure 1 illustrates the issue. The user is not aware of the application or descriptor requirements.

In the rest of this section we analyze different scenarios where the roles of programmers, users and infrastructure managers are combined or separated into different people, and explain different approaches that are able to solve these scenarios.

3.1 Infrastructure Technical Services

The concept of non-functional requirements, i.e. technical services, was first introduce in the field of component models. Such models allows a clear separation between the functional code written by the developer and the non-functional services provided by the framework. In [16] a technical service must be developed by an expert of the field. For example, an expert in fault-tolerance must implement the fault-tolerance service, because he can provide a good quality-of-service for a large range of applications.

A technical service is a non-functional requirement that may be dynamically fulfilled at runtime by adapting the configuration of selected resources [6]. The infrastructure manager is aware of the technical services that can be provided by the infrastructure, and can configure a deployment descriptor to specify the available technical services.

For example, to configure fault-tolerance, a `services.FaultTolerance` class is provided. This class defines how the configuration is applied to all active objects hosted on the specified node. The deployment descriptor specifies the fault-tolerance in the following way:

```
<technical-service id = "myService" class="services.
    FaultTolerance">
  <arg name="proto" value="cic"/>
  <arg name="server" value="rmi://host/FTServer"/>
  <arg name="port" value="6060"/>
</technical-service>
```

The configuration parameters of the service are specified by `arg` tags in the deployment descriptor. Those parameters are passed to the `init` method as a map associating the name of a parameter as a key and its value. The `apply` method takes as parameter the node on which the service must be applied. This method is called after the creation or acquisition of a node, and before the node is used by the application.

Figure 2 summarizes the deployment framework provided by the ProActive middleware. Deployment descriptors can be separated in three parts: mapping, infrastructure, and non-functional aspects. The virtual-node is the shared abstraction between applications and descriptors. The virtual-node is referenced from inside the application code, and is also mapped to nodes in the deployment descriptors.

The infrastructure manager knows the most adequate fault-tolerance mechanism depending on the environment, and can configure this mechanism as a technical service in the deployment descriptor. However, the developer of the application knows on which virtual-nodes to apply the adequate technical service. Therefore, we introduce in the next section the concept of virtual-node-descriptor.

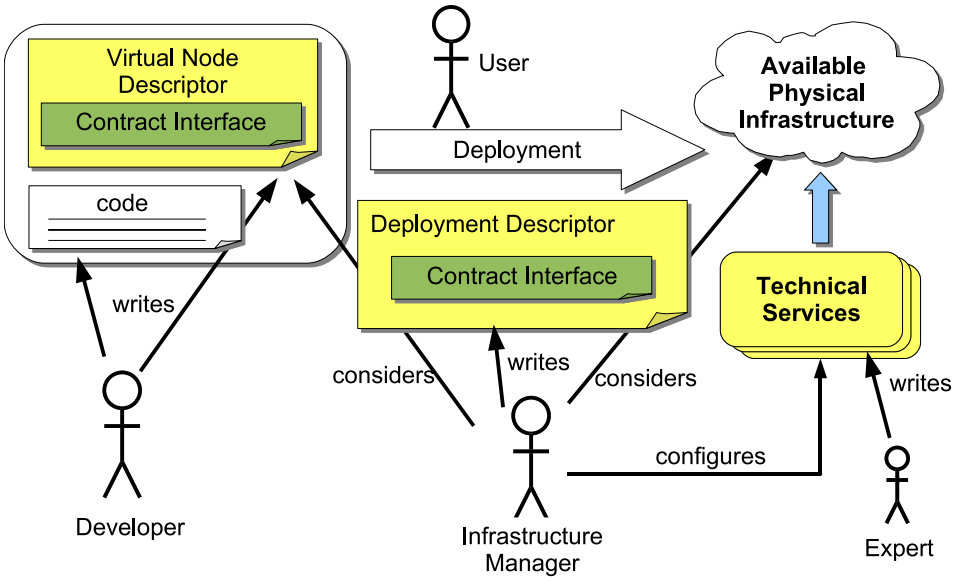

Figure 1. Deployment roles and artifacts

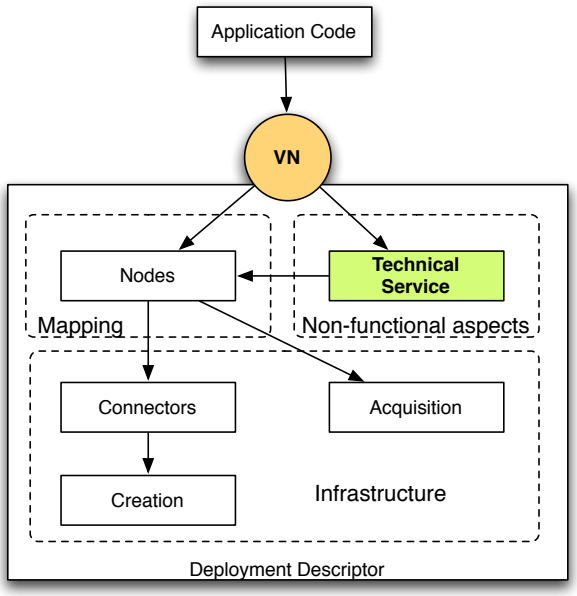

Figure 2. Deployment descriptor model

3.2 Application Virtual Node Descriptor

Virtual-node-descriptor is a mechanism for specifying the environmental requirements of the applications [8]. The requirements of the application are specified by the programmer in a virtual-node-descriptor. The virtual-node-descriptor is packaged with the application when distributing it to the users.

The virtual-node-descriptor is expressed in a dedicated XML file packaged with the application such as:

```
<virtual-nodes>
  <virtual-node name="VN-Slaves">
    <technical-service type="services.FaultTolerance"/>
    <nodes required="500" minimum="100" />
    <processor architecture="x86"/>
    <os name="linux" release="2.6.15"/>
  </virtual-node>
<virtual-nodes>
```

Non-functional requirements are expressed in a simple way. The tag `technical-service` specifies the technical service required by the application. Developer can also expresses the total and/or the minimum number of nodes required by the application. Additionally, other requirements can also be specified such as hardware or software.

In order to deploy, these requirements must be fulfilled by the deployment framework for the indicated virtual-node. Also, the technical service must fit, i.e. extends or implements, the type specified in the virtual-node-descriptor.

Using virtual-node-descriptors, the user does not have to be aware of the application's design and implementation. By simple inspection of the virtual-node-descriptor, the user can know the requirements of the application.

By default there is no contract management module, such as in [18], nor deployment planner such as in [17]. Indeed, virtual-node-descriptors are verified when retrieving nodes from the physical infrastructure, resulting in runtime errors if the requirements are are not satisfied. This ensures a simple framework in terms of specification and verification, eludes resource planning issues, and could still be

plugged to a resource allocator framework such as Globus's GARA [13].

Nevertheless, developers dont know on which infrastructure the applications will be deployed, and the infrastructure may not support some specific requirement of the application. Therefore, in Section 4 we propose to describe the infrastructure with a mechanism based on coupling contracts, which is described in the next section.

3.3 Coupling Contracts

Coupling Contracts proposes capturing the properties of *how* information agreement takes place between parties, specifically between applications and descriptors [4]. To achieve this, each party provides an interface holding a set typed clauses. The clauses specify *what* information is required and provided by each party, and the type specifies how an agreement on this information is reached. If the interfaces are compatible, the coupling of the interfaces yields a contract with the agreed values for the clauses.

3.3.1 Concepts: Contracts, Interfaces and Clauses

Typed Clauses correspond to the information that both parties must agree on. A clause is defined by a type, a name and a value. The clauses are typed with one of the alternatives shown in Table 1. As an example, the *Application* type specifies that the value of the clause can only be set by the application. The descriptor specifies a value as required, forcing the application to provide a value. Another example corresponds to the *Descriptor-Priority* type which specifies that a default value can be provided by the application, and that the value can be overridden by the descriptor. Additionally, parties can enforce constraints on the value of the clauses such as maximal and minimal value, choices, etc. The default constraint corresponds to non-emptiness.

Interfaces represent a grouping of clauses that are exposed by each party. An interface is defined by a name and a set of clauses.

Coupling Contracts are the results of coupling two interface. The contract holds the clauses and their values. The values of the clauses are resolved using the specific type for each clause. If there is a conflict of types, or the value does not satisfy the constraints, then the contract is invalid and the coupling is not allowed. When a contract is valid, then both parties can query the contract to get the value of the agreed clauses.

Typed clauses can also be used to perform advertisement and matchmaking in the Condor style [19]. Both parties can expose their interfaces (advertisements) to a matchmaker or broker. To determine if the two parties are a suitable match, the coupling contract can be generated and validated.

The clauses belonging to the interfaces will specify *what* information is shared (provided or required) for the matchmaking. And the type of the clauses will specify *how* the information is shared for the coupling.

4 Deployment Contracts Example

In this section we show how the concepts introduced in Section 3 can be merged and applied to provide full separation of roles: developer, infrastructure manager and user. Specifically, we aim at creating deployment contracts between the applications and the deployment descriptors using the Grid middleware ProActive. We will show how the deployment framework can benefit from the use of: technical services, virtual-node-descriptors, and coupling contracts to deploy unfamiliar applications with unfamiliar infrastructures.

The example presented in this section uses the fault-tolerance mechanism provided by ProActive [2]. The fault-tolerance is based on rollback recovery. Several parametrized protocols can be used, regarding the application's requirements and the characteristics of the infrastructure.

The application specifies its provisions and requirements in the virtual-node-descriptor. Figure 3 shows an example for a master-slave application. Symmetrically, Figure 5 shows the provisions and requirements of the descriptor. The coupling contract is composed of the clauses specified in both, and the values of this contract will be used in the virtual-node-descriptor (Figure 3), application (Figure 4), and in the deployment descriptor (Figure 6).

VN_MASTER & VN_SLAVES are of type `Descriptor` type. These clauses will hold the required and provided names of the virtual-nodes.

NUM_NODES is of type `Application-Priority`. The virtual-node-descriptor specifies that the application requires `16` nodes. The descriptor-interface specifies that this value must be grater than zero, and smaller than the maximum number of allowed nodes.

FT_PROTOCOL is of type `Descriptor-Priority`. The virtual-node-descriptor specifies that the application requires the fault-tolerance protocol to be either `cic` or `oml`, suggesting `cic` as the default value. On the other hand, the descriptor-interface specifies that the protocol must be one of: `pml`, `cic`, `oml`, and overrides the virtual-node-descriptor by choosing `oml`.

ARCH is of type `Application-Priority`. The virtual-node-descriptor specifies that the architecture

Table 1. Types

Type Name —	Provides Value —	Requires Value —	Set constraints —	Priority
Application	App	Desc	Desc	App
Descriptor	Desc	App	App	Desc
Application-Priority	App,Desc	Desc	App,Desc	App,Desc
Descriptor-Priority	Desc,App	App	Desc,App	Desc,App
Environment	Env	Desc,App	Desc,App	Env

must be configured to `x86` because it provides specific binary code for this architecture. The descriptor-interface provides the following architectures: `x86`, `sparc`, `ppc`, and `any`.

OS is of type `Application-Priority`. The virtual-node-descriptor specifies that the operating system must be configured to `Linux` because it provides specific binary code for this operating system. The descriptor-interface provides the following operating systems: `Linux`, `MacOS`, `Solaris`, and `any`.

In the virutal-node-descriptor, the developer activates the fault-tolerance technical service for the master virutal-node, since it represents a single point of failure in the application. The protocol used for fault-tolerance will correspond to the agreed value of the coupling contract, which in the example corresponds to `oml`. The developer also specifies the required number of nodes, which is validated using the virtual-node-descriptor against the allowed minimum. On the other hand, the infrastructure manager specifies in the descriptor the optimistic maximum number of nodes that the infrastructure can provide, and validates the application's required number of nodes using the clause's constraints.

The architectures and operating systems that are supported by the infrastructure are specified in the descriptor using typed clauses. The application's requirements are also specified as clauses, but in the virtual-node-descriptor. this is useful for applications that have binary code which runs only on a specific operating system with a specific infrastructure. When the coupling contract is generated, both descriptor and application have reached an agreement on the characteristic of the resources. In the example the agreement corresponds to: `Linux`, `x86`.

5 Related Work

The problem of finding suitable resources for a given application have already been addressed by techniques such as matchmaking in Condor [19, 20], collections in Legion [10], or using resource management architectures like Globus[11].

```
<virtual-nodes>
  <clauses>
    <interface name="application-master-slave-interface">

      <Descriptor name="VN_MASTER" />
      <Descriptor name="VN_SLAVES" />

      <ApplicationPri name="NUM_NODES" value="16"/>

      <DescriptorPri name="${FT_PROTOCOL}" value="cic">
        <or>
          <equals>cic</equals>
          <equals>oml</equals>
        </or>
      </DescriptorPri>

      <ApplicationPri name="ARCH" value="x86"/>
      <ApplicationPri name="OS" value="Linux"/>
    </interface>
  </clauses>

  <virtual-node name="${VN_MASTER}">
    <technical-service type="service.FaultTolerance"/>
  </virtual-node>

  <virtual-node name="${VN_SLAVES}">
    <processor architecture="${ARCH}"/>
    <os name="${OS}"/>
    <nodes required="${NUM_NODES}" minimum="10"/>
  </virtual-node>
</virtual-nodes>
```

Figure 3. Application: VN Descriptor

```
//If the application and descriptor can not be coupled
     an exception will be thrown
ProActiveDescriptor pad = ProActive.
     getProactiveDescriptor("descriptor.xml", "vn-
     descriptor.xml");

//Retrieving Clauses from the Contract
CouplingContract cc = pad.getCouplingContract();
String vnMasterName = cc.getValue("VN_MASTER");
String vnSlavesName = cc.getValue("VN_SLAVES");

VirtualNode vnMaster=pad.getVirtualNode(vnMasterName);
VirtualNode vnSlaves=pad.getVirtualNode(vnSlavesName);
...
```

Figure 4. Application Code

```
<clauses>
  <interface name="descriptor-master-slave-interface">
    <Descriptor name="VN_MASTER" value="vn-master"/>
    <Descriptor name="VN_SLAVES" value="vn-slaves"/>

    <Descriptor name="MAX_NODES" value="100"/>
    <ApplicationPri name="NUM_NODES" value="1">
      <and>
        <biggerThan>0</biggerThan>
        <smallerThan>${MAX_NODES}</smallerThan>
      </and>
    </ApplicationPri>

    <DescriptorPri name="${FT_PROTOCOL}" value="oml">
      <or>
        <equals>pml</equals>
        <equals>cic</equals>
        <equals>oml</equals>
      </or>
    </DescriptorPri>

    <ApplicationPri name="ARCH" value="any">
      <or>
        <equals>x86</equals>
        <equals>ppc</equals>
        <equals>sparc</equals>
        <equals>any</equals>
      </or>
    </ApplicationPri>
    <ApplicationPri name="OS" value="any">
      <or>
        <equals>Linux</equals>
        <equals>MacOS</equals>
        <equals>Sun</equals>
        <equals>any</equals>
      </or>
    </ApplicationPri>
  </interface>
</clauses>
...
```

Figure 5. Deployment Descriptor Interface

```
...
  <virtualNodesDefinition>
   <virtualNode name="${VN_MASTER}" serviceId="ft-serv"/>
   <virtualNode name="${VN_SLAVES}"/>
  </virtualNodesDefinition>
...
<technicalServiceDefintions>
  <service id="ft-serv" class="services.FaultTolerance">
    <arg name="proto" value="${FT_PROTOCOL}"/>
    <arg name="server" value="rmi://host/FTServer"/>
    <arg name="TTC" value="60"/>
  </service>
</technicalServiceDefinitions>
...
```

Figure 6. Deployment Descriptor

However, the approaches presented in this work not only focus on acquiring resources, but also on generating contractual agreements during the deployment process.

Therefore, our approach pertains more to Service Level Agreement, and more specifically, *how* to manage the negotiation, in order to end up with an agreement between what is usually called customers and providers: e.g. with the help of software agents to coordinate the negotiation, as in [15], or orchestrated along a specific algorithm in the MetaSchedulingService described in [21].

Another related approach corresponds to the Web Services Agreement (WS-Agreement) Specification[1], which is about to become a draft recommendation of the Global Grid Forum[12]. The WS-Agreement is a two layer model: Agreement Layer and Service Layer. Many of the concepts introduced in our work find their reflection in the Agreement Layer. According to the specification "an *agreement* defines a dynamically-established and dynamically-managed relationship between parties", much like the proposed coupling contracts. Also, the proposed coupling interfaces can be seen as *agreement templates* in WS-Agreement, since they are both used to perform advertisement. Additionally, in the same way that interfaces and contracts are composed of clauses, in WS-Agreement templates and agreements are composed of *terms*. Finally, the concept of constraints is present in both approaches.

The similarity of our proposed approach and WS-Agreement Specification is encouraging when we consider that both were conceived independently. On the other hand, the main difference in the approaches is that the definition of a protocol for negotiating agreements is outside of the WS-Agreement Specification scope.

From the WS-Agreement perspective, typed clauses can be seen as an automated negotiation approach because they provide an automated mechanism for accepting or rejecting an agreement.

6 Conclusions and Future Work

In this paper we have addressed the separation of roles: application developer, infrastructure manager, and user. We have identified that agreements must be made between these different roles in order to execute the application on a distributed infrastructure: Desktop Machines, Clusters and Grids.

We have argued that the key moment to perform an agreement corresponds to the deployment time. During the deployment, the application and infrastructure must reach a contractual agreement. The contract will allow the execution of the application on distributed resources by specifying, among others, the technical services.

To generate the deployment contract we have described the application's provisions and requirements using virtual-

node-descriptors, and symmetrically, we have specified the infrastructure's provisions and requirements in deployment descriptor interfaces.

In the future we would like to simplify the coupling contracts to allow negotiation with typeless clauses, using constraint satisfaction instead. We would also like to investigate dynamic renegotiation of contracts after the deployment.

References

[1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification (ws-agreement). Draft Version 2005/09. http://forge.gridforum.org/projects/graap-wg.

[2] F. Baude, D. Caromel, C. Delbé, and L. Henrio. An hybrid message logging-cic protocol for constrained checkpointability. In *Proceedings of Europar 2005*. Springer-Verlag, 2005.

[3] F. Baude, D. Caromel, L. Mestre, F. Huet, and J. Vayssière. Interactive and descriptor-based deployment of object-oriented grid applications. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, pages 93–102, Edinburgh, Scotland, July 2002. IEEE Computer Society.

[4] J. Bustos-Jimenez, D. Caromel, M. Leyton, and J. M. Piquer. Coupling contracts for deployment on alien grids. In *Proceedings of the International Euro-Par Workshops*, Lecture Notes in Computer Science, Dresden, Germany, August 2006. Springer-Verlag. To appear.

[5] D. Caromel. Towards a Method of Object-Oriented Concurrent Programming. *Communications of the ACM*, 36(9):90–102, September 1993.

[6] D. Caromel, C. Delbe, and A. di Costanzo. Peer-to-peer and fault-tolerance: Towards deployment based technical services. January 2006.

[7] D. Caromel, C. Delbe, A. di Costanzo, and M. Leyton. Proactive: an integrated platform for programming and running applications on grids and p2p systems. *Computational Methods in Science and Technology*, 12, 2006.

[8] D. Caromel, A. di Costanzo, C. Delbé, and M. Morel. Dynamically-fulfilled application constraints through technical services - towards flexible component deployments. In *Proceedings of CompFrame 2006, Component and Framework Technology in High-Performance and Scientific Computing*, Paris, France, June 2006. IEEE.

[9] D. Caromel, A. di Costanzo, and C. Mathieu. Peer-to-peer for computational grids: Mixing clusters and desktop machines. *Parallel Computing Journal on Large Scale Grid*, 2007. To appear.

[10] S. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw. Resource management in legion. Legion Winter Workshop, 1997.

[11] K. Czajkowski, I. T. Foster, N. T. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pages 62–82, London, UK, 1998. Springer-Verlag.

[12] G. G. Forum. http://www.gridforum.org.

[13] I. Foster, A. Roy, and V. Sander. A quality of service architecture that combines resource reservation and application adaptation. In *Proceedings of the Eight International Workshop on Quality of Service (IWQOS 2000)*, pages 181–188, June 2000.

[14] Gnutella. http://www.gnutella.com.

[15] D. Greenwood, G. Vitaglione, L. Keller, and M. Calisti. Service Level Agreement Management with Adaptive Coordination. In *Int. conference on Networking and Services (ICNS'06)*, 2006.

[16] J. Kienzle and R. Guerraoui. Aop: Does it make sense? the case of concurrency and failures. In *ECOOP*, pages 37–61, 2002.

[17] S. Lacour, C. Pérez, and T. Priol. Generic application description model: Toward automatic deployment of applications on computational grids. In *6th IEEE/ACM International Workshop on Grid Computing (Grid2005), Seattle, WA, USA*. Springer-Verlag, november 2005.

[18] O. Loques and A. Sztajnberg. Customizing component-based architectures by contract. In *Second International Working Conference on Component Deployment (CD 2004)*, volume 3083 of *Lecture Notes in Computer Science*, pages 18–34, Edinburgh, UK, May 2004. Springer-Verlag.

[19] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *In Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, 1998.

[20] R. Raman, M. Livny, and M. Solomon. Policy driven heterogeneous resource co-allocation with gangmatching. In *Proc. of the 12th IEEE Int'l Symp. on High Performance Distributed Computing (HPDC-12)*, 2003.

[21] P. Wieder, R. Yahyapour, O. Wäldrich, and W. Ziegler. Improving workflow execution through sla-based advance reservation. Technical Report CoreGRID TR-053, CoreGRID Network of Excellence, 2006.