Towards a Service-Oriented Ad Hoc Grid

Matthew Smith, Thomas Friese, Bernd Freisleben
Department of Mathematics and Computer Science, University of Marburg
Hans-Meerwein-Strasse, D-35032 Marburg, Germany
Email: {matthew, friese, freisleb}@informatik.uni-marburg.de

***Abstract*— In its current state, service-oriented grid computing focuses on the unification of resources through virtualization, to enable on demand distributed computing within a preconfigured environment. Organizations or inter-organizational communities willing to share their computational resources typically create a centrally planned grid, where dedicated grid administrators manage the nodes and the offered grid services. In this paper, we present the idea of a spontaneously formed, service-oriented *ad hoc grid* to harness the unused resources of idle networked workstations and high-performance computing nodes on demand. We discuss the requirements of such an ad hoc grid, show how the service-oriented computing paradigm can be used to realize it and present a proof-of-concept implementation based on the Globus Toolkit 3.0. The features of this system are peer-to-peer based node discovery, automatic node property assessment, hot deployment of services into a running system and added inter-service security.**

***Index Terms*— Grid Computing, Ad Hoc Computing, Service Orientation, Service Deployment, Peer-to-Peer Discovery, Inter-Service Security.**

I. Introduction

The grid computing paradigm [1], [2] is attracting a growing number of users developing larger distributed computing projects than ever before. The initial vision of the grid encompasses the fusion of different high-performance computing centers into a common infrastructure that allows uniform access to those heterogeneous systems.

Currently, most grid projects are in the hands of large research organizations, companies or governments such as the NASA (Information Power Grid) [3], the US Department of Energy & IBM (Science Grid) [4] and the European Union (EGEE) [5]. These institutions have dedicated staff who manage the grid and configure it specifically for their needs. The installation of a production quality large scale grid is far from trivial, making these administrators vital to the task.

If the grid is to fulfill the vision of becoming the next-generation Internet (as described in [5], [6], [7]), the complexity of installing and maintaining the grid must be reduced significantly. The Internet boom was made possible by making access to the Internet intuitive and transparent to the users. As a consequence, the number of users increased exponentially, which further increased the support for and the acceptance of the new medium.

The introduction of the service-oriented computing paradigm and the corresponding web service standards such as WSDL [8] and SOAP [9] in the field of grid computing through the Open Grid Services Architecture (OGSA) [10] is a major step towards reducing the complexity of grid use, operation and maintenance. While the OGSA describes the higher-level architectural aspects of service-oriented grid computing, the Open Grid Services Infrastructure (OGSI) is a fine-grained description of the infrastructure required to implement the OGSA model. Service-oriented grid computing offers the potential to provide a fine grained virtualization of the available resources to significantly increase the versatility of a grid. It can be employed to create a broader user base as a catalyst for new grid developments by extending the initial vision of the grid – connecting the world's supercomputing centers – to also incorporate the much larger community of desktop computers. The influx of users will allow the grid to offer the possibility of harnessing the unused CPU cycles (or other resources) of idle workstations, as found in practically every organization, by combining them on demand to spontaneously form an *ad hoc grid* without a preconfigured fixed infrastructure.

To achieve this goal, a number of new challenges must be taken into account. For example, through the extension of the grid by non-dedicated resources, the complexity of the grid is greatly increased. Currently, a handful of administrators with specialist knowledge manage their grid infrastructure, configure the separate nodes and preinstall all grid services which are required. When a large number of nodes is added to the grid on a dynamic basis, central administration is no longer feasible. The heterogeneity of the system is increased and the reliability of the nodes is decreased due to reboots or crashes caused by the regular users of those nodes. The system itself must be capable of coping with the dynamic topology changes of the underlying network and the heterogeneity of the nodes to form an ad hoc grid autonomously. Security is also of vital importance to such an extended grid system. Since the number of users within a system is increased, new security mechanisms are needed to ensure that malicious code can not harm legitimate services running on the grid.

In this paper, we present the main problems involved in realizing a service-oriented ad hoc grid to provide computing resources on demand to every participant. Our solutions to these problems are based on peer-to-peer node discovery, automatic node property assessment using property scouts to check how well a node meets a certain set of requirements, hot deployment of services into a running system without disrupting other services already running there and added inter-service security by ensuring that each service runs within its own sandbox and has no direct access to the running code

This work is partially supported by Siemens AG, Corporate Technology München, Germany

of other services. The important parts of a proof-of-concept implementation based on the Globus Toolkit 3.0 (GT3) [11] will be described. All components developed for building an ad hoc grid can be inserted non-intrusively by simple configuration changes and do not require any alteration of GT3 code. The service-oriented ad hoc grid environment introduced in this paper opens up a whole new range of resources to be tapped and expands the potential user base of the grid paradigm significantly.

The paper is organized as follows. In section II, we introduce our notion of an ad hoc grid. In section III, we present the requirements that must be met by a service-oriented ad hoc grid environment. Our proof-of-concept implementation is described in section IV. Related work is discussed in section V. Section VI concludes the paper and outlines areas for future research.

II. The Ad Hoc Grid

An ad hoc grid is a spontaneous formation of cooperating heterogenous computing nodes into a logical community without a preconfigured fixed infrastructure and with only minimal administrative requirements. The main goal of an ad hoc grid is to provide computing resources on demand to every participant. Unlike traditional grid systems, the number of non-dedicated grid nodes is much higher, demanding non-intrusive operation of the ad hoc grid middleware.

Thus, our view of an ad hoc grid environment goes beyond the preconfigured, dedicated grid infrastructures existing today to encompass frequent dynamic additions to the grid. This includes workstations within organizations as well as scattered personal computers, similar to the basic idea of many distributed computing projects like SETI@Home [12].

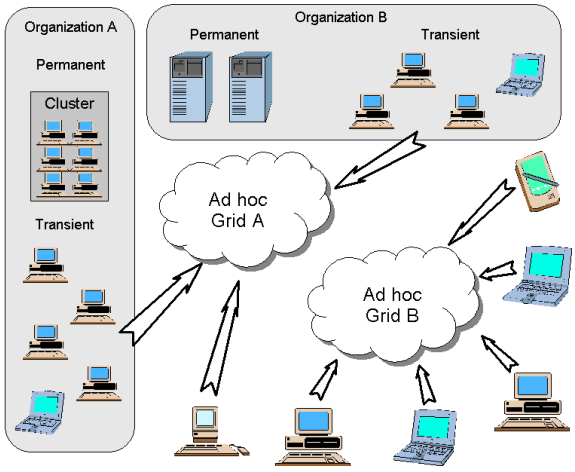

Fig. 1. Ad hoc grid architecture overview

Figure 1 shows how two separate ad hoc grids are composed. The first ad hoc grid spans two organizations, the second is created from scattered nodes on the Internet. Both grid communities form a virtual organization using the existing Internet infrastructure. While ad hoc grid A encompasses transient nodes (e.g. non-dedicated workstations), it also includes dedicated high-performance computers. In contrast, ad hoc grid B is made up solely of transient individual nodes. While ad hoc grid A bears a greater resemblance to traditional grid systems, ad hoc grid B illustrates the shift to a personal grid system, built without the resources of a large organization.

In the next section, we discuss the general challenges of building an ad hoc grid environment.

III. Challenges

The main steps which need to be taken to allow ad hoc grid computing in a heterogenous environment are as follows.

- **Node Discovery:**
  In an ad hoc grid environment, the network topology is dynamic (i.e. rebooting of workstations, movement of laptops, replacement of computers) and thus a node detection solution geared towards frequent node arrivals and departures is required. While arrival or departure of nodes should be discovered as quickly as possible, a balance needs to be found between keeping the topology information up to date and flooding the network with discovery messages.
- **Node Property Assessment:**
  OGSA and OGSI define virtualization of available resources at the system-independent level of resource access to allow uniform access to a heterogeneous system. The hardware and operating system or underlying implementation of a service is hidden from the caller of a service. Support of the standard grid service interfaces is guaranteed and sufficient to allow access to the resource. Even the instantiation of a deployed grid service is system-independent, since it is specified to be handled by a gatekeeper (service factory).
  To enable autonomous deployment, meta-information from the grid participant must be available to the deployment service, so it can reliably operate in a heterogenous environment. While the underlying system of each node is guaranteed to support the grid service interface, the way it implements this is not specified by OGSA or OGSI. When deploying services to newly discovered nodes, it is, however, necessary that the service implementation is supported by the grid platform of the node. Information about the underlying hardware and operating system is particularly important when deploying legacy code, because tight integration of system resources is a common occurrence there.
  Typical information needed is operating system type, available hardware resources and availability of required libraries. Furthermore, information on the reliability of the nodes can be taken into account when services are to be deployed, so nodes with long up times are given priority over nodes which frequently reboot or crash.
- **Service Deployment:**
  Vital to the invocation of services on various nodes in the grid system is the availability of those services. For a large scale ad hoc grid, the time consumption for manual deployment is prohibitive, and management is difficult due to the fluctuating availability of the nodes. Even with the availability of advanced grid programming toolkits,

deployment of services has been identified as a critical issue [13]. Furthermore, the number of grid services will steadily rise with the number of users, further increasing the management cost of the grid environment. In a dynamically changing environment, deployment is even more critical as there is no single deployment cycle that reaches all machines. Instead, services need to be deployed and instantiated on demand on machines as they become available.

Service deployment becomes part of an ad hoc grid application instead of being handled by a system administrator as a precondition to the use of a service. Instead of only providing predefined services, the computational nodes of the bare grid become a resource in themselves. This resource can be tapped by applications using the hot deployment service to leverage spare resources into their computational group. When a node becomes available that meets the requirements for the deployment of a service, the application can autonomously carry out the deployment and use the newly available machine for its application flow.

In a production environment, every operation needs to be non-intrusive, i.e. it does not interfere with the execution of other services already running on the grid. The ability to introduce or remove a service without interruption of other operations is vital for the vision of a highly flexible ad hoc grid.

- **Service Security**
  Security is a major aspect in all distributed systems, since there is always the possibility that a node introduces malicious code. In an ad hoc grid, several new aspects must be dealt with beyond the standard security requirements existing in previous grid systems. In traditional systems, installing a service requires security certificates allowing the operations. Services usually can only be installed by a very small number of people and trust can be assumed between all parties. In a large ad hoc system on the other hand, it is possible that users unknown to each other operate on the same node. This gives rise to new security issues. One major new security threat is that a trusted node is running further unknown services. Here, inter-service security must be offered, since fair play is not guaranteed any more.

IV. Implementation of a Service-Oriented Ad Hoc Grid

In this section, we present a proof-of-concept implementation of a service-oriented ad hoc grid, dealing with each of the challenges mentioned above.

As a basis for our work we chose the Globus Toolkit 3.0 (GT3) deployed in Tomcat 5 [14], since it is the most widely used implementation of the Open Grid Services Infrastructure. Furthermore, since GT3 is deployed as an Axis Service in Tomcat, it offers a number of access points into the system via configuration of the Axis Service, allowing easy integration of new features.

A. Node Discovery

To enable automatic service deployment in an ad hoc grid environment, the participating nodes must be discovered first. Due to the potentially large size of future grids, manual discovery as practiced in existing grid environments (see section V) is not an option. Since the grid can cross the boundaries of organizations, a simple multicast or broadcast will not reach all potential participants without proper preconfiguration of the network infrastructure which is unacceptable in an ad hoc environment. An automatic discovery mechanism is needed to find nodes willing to participate in the grid. A central registry system is easy to install but does not scale well and introduces a single point of failure. For ad hoc grids, a decentralized discovery mechanism is vital to cope with the fluctuating topology and large number of participants.

The peer-to-peer community has spent significant effort to solve the node discovery problem in large, heterogenous and unreliable networks. Peer-to-peer and grid computing systems have a number of similarities. Both systems aim to bring together distributed resources. In general, peer-to-peer systems are designed to fulfill a single task (e.g. file sharing), while grids are multi-purpose and offer greater flexibility for distributed application design. The advantage of peer-to-peer systems is that they are easier to install, configure and administer. Typically, there is no central coordination needed at all. Current grid systems are relatively small encompassing several thousands of nodes, while peer-to-peer systems can connect millions of nodes [15], [16], [17] using only the limited resources of personal computers. In [7], Iamnitchi and Foster present a more detailed comparison of the two technologies. The authors also state that peer-to-peer applications are becoming more complex, offering general distributed computing capacities. At the same time, grid systems are growing bigger and thus the differences between the two paradigms are likely to disappear over time. Although a number of papers [18], [19], [20] discuss the benefits offered by the confluence of peer-to-peer and grid computing, to the best of our knowledge, most of the projects suggest to integrate peer-to-peer computing ideas only for particular aspects of grid computing rather than integrating a fully fledged peer-to-peer solution at its core.

In our prototype implementation of an ad hoc grid, we have integrated the JXTA [21] peer-to-peer system into the grid environment. As a first application, we have implemented a peer-to-peer based discovery mechanism to find all nodes running in our grid system. Peers within this system form *peer groups* of cooperating nodes, allowing node discovery scoped by group membership.

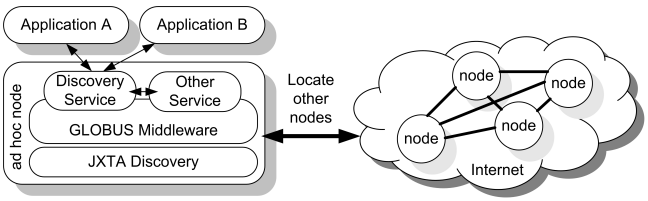

Fig. 2. JXTA-based discovery components.

Figure 2 shows the relationship of the discovery components within our system. The discovery service itself is a grid service hosted by GT3, thus staying within the bounds of the service-oriented grid. Exposing the peer-to-peer discovery mechanism with a grid service interface makes the discovery functionality accessible to other services and applications, providing a useful service in of itself, without locking the functionality inside the middleware layer. Figure 3 shows the interface of the discovery mechanism. The first method returns all active nodes which can be found in `hop` hops, up to a maximum `max`. The second method only returns the nodes which also match the requirements `req` (see section IV-B). The `NodeMap` contains a list of nodes attributed with the information on how well the requirements were matched.

```
NodeMap discoverAllNodes(int hop, int max)
NodeMap discoverNodes
            (int hop, int max, String req)
```

Fig. 3. Interface of the discovery mechanism

By using a JXTA-based discovery mechanism, we leverage the ability of peer-to-peer systems to cope with large and dynamically changing networks into the grid environment. Since the discovery mechanism is a loosely coupled grid service itself, it can be replaced at any time, allowing us to exchange different peer-to-peer protocols or even use a central registry when needed.

Based on this mechanism, our ad hoc grid system is now capable of discovering all reachable nodes running in the grid environment. The next step necessary is to assess the capabilities of these nodes in respect to the requirements of the services to be executed on them.

B. Node Property Assessment

A grid service in GT3 consists of a language-independent grid service description and the accompanying implementation. In a preconfigured grid environment, the developers of the grid service or the administrator of the grid manually install the service on the nodes, thus ensuring the service is only installed on nodes meeting its requirements. In an ad hoc grid environment, where nodes enter and leave the grid on a regular basis, an automated deployment mechanism is required. Since the ad hoc grid consists of heterogenous nodes, it is no longer given that all nodes are capable or well suited to run a certain service. Thus, as a prerequisite to automated deployment, the properties of the nodes within the grid need to be analyzed and then be matched to the requirements of the grid service. Operating system type, language support, processor power, available memory and up-time are properties of a node which can be taken into account when a service is to be deployed.

In our ad hoc grid prototype implementation, we developed a grid service called *property scout* which checks how well the node on which it is running meets a certain set of requirements. An example set of requirements is shown in Figure 4.

The grid service using this requirements block needs a Windows XP operating system, a Java Runtime Environment version 1.2 or higher and a preinstalled custom DLL.

```
<check>
  <OS name=windows version=XP
     modifier=none/>
  <Language name=JRE version=1.2
     modifier=orHigher/>
  <Library name=custom.dll/>
</check>
```

Fig. 4. Set of requirements

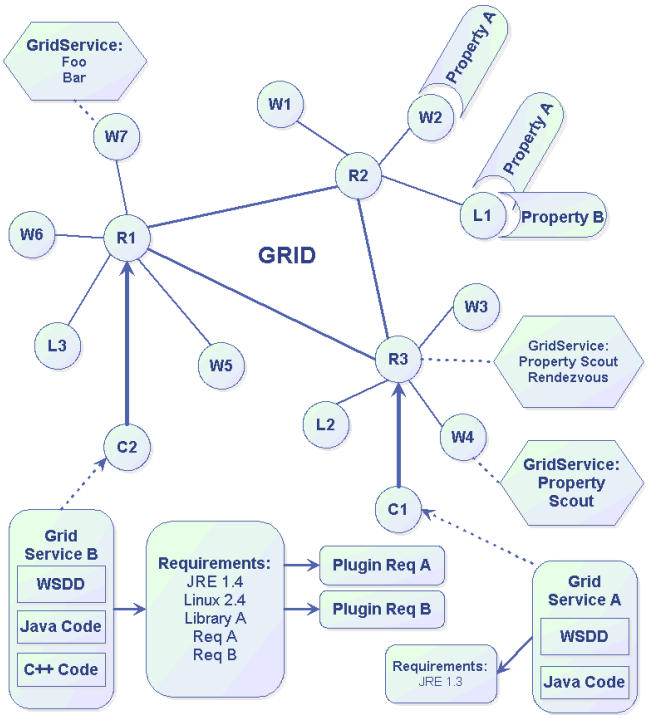

Fig. 5. Peer-to-peer based ad hoc grid with property scouts

Figure 5 shows an ad hoc grid based on a peer-to-peer network consisting of three JXTA rendezvous nodes (R1 to R3), three Linux workstations (L1 to L3), seven Windows workstations (W1 to W7) and two clients (C1 and C2). Regular lines represent the peer-to-peer connections between the nodes. Client C1 wants to deploy grid service A which requires the Java runtime environment version 1.3. Client C2 wants to deploy grid service B which requires the Java runtime environment 1.4, Linux with a kernel version 2.4, a preinstalled library A and two custom requirements A and B. The thick arrows point from the client wishing to deploy a service to the rendezvous used for deployment. Each grid node is running the property scout service. Furthermore, the rendezvous nodes are equipped with property scout rendezvous services introduced later in this section.

Clearly, it is not possible to anticipate all requirement types and evaluation metrics since each grid service can have unique needs specific to its task. For instance, if a grid service needs to cooperate with a remote machine in real-time, a low network latency between the grid service node and the remote machine is required. To be able to incorporate such application specific

requirements, a plug-in architecture was developed to give the property scouts the necessary flexibility to cope with an ad hoc environment and unanticipated requirements. Custom analyzers conforming to the plug-in interface can be created by the developers of a grid service specifically tailored to their needs. The requirements description file is extended to include the name of the analyzer capable of parsing and checking a set of requirements. This coupling of requirement specification and evaluation code offers developers the capability to test an unknown platform for the specific needs of their application. For instance, instead of merely checking for high processor speeds, a custom analyzer could actually perform some specific operations and check the system performance using actual operational code. The requirements in Figure 6 show how a custom analyzer can be set to check whether its method `calculateX` is completed in less than 2000 ms, and whether 100 MB of test data can be written to the hard disk in less than 3000 ms. The methods implementing the test are supplied by the developer and placed in the class specified in the requirements file.

```
<?xml version="1" encoding="ISO-8859-1"?>
<GridServiceRequirements>
<name> SampleGridService </name>
<req>
  <analyzerClass>
      jxtagrid.systemanalyzer.
      CustomSystemAnalyzer
  </analyzerClass>
  <check>
    <method name=calculateX
      inLessThan=2000ms/>
    <HDAccess size=100MB
      inLessThan=3000ms/>
  </check>
</req>
</GridServiceRequirements>
```

Fig. 6. Requirements for a custom analyzer

This flexibility is particularly important in a heterogenous environment, since simply comparing CPU cycles or similar static attributes is difficult due to architectural differences of the nodes. For instance, a Centrino with 1.4 GHz is not slower than a Pentium P4 with 2 GHz.

The decision whether a node is *capable* of hosting a grid service can be made locally on the node based on the requirements specification. Whether a node is the best match for a given service, can only be decided globally. This poses a special challenge in an ad hoc grid, as new nodes can enter at any time, changing the overall situation.

Since our node discovery is handled by JXTA, we integrated a *property scout rendezvous* service into the JXTA rendezvous nodes. The property scout rendezvous services gather the data produced by the property scouts and if required can correlate them with each other to create a global view of the grid capabilities. If, for instance, a grid service needs to run on 100 nodes, a custom property scout rendezvous can be created to discover which 100 nodes currently available are the best match for the service's requirements instead of simply installing it on the first hundred nodes capable of hosting the service. Figure 7 shows a simple peer-to-peer based node property assessment. Client C1 wants to deploy a grid service on as many nodes fulfilling requirements A and B as possible (in this case only nodes L1 and W4). A request is sent to the local rendezvous R1 which in turn sends it to its connected peers and the other rendezvous nodes R2 and R3. Rendezvous nodes R2 and R3 then forward the requirements to their peers. Nodes L1 and W4 match the requirements, so they send a message to R1 that they are capable of hosting the grid service of C1. If they can communicate directly with R1 they do so, otherwise they use their local rendezvous as a bridge to the required endpoint. The responses are sent to the rendezvous nodes to avoid flooding the requesting client with responses. The thick arrows between C1, R1, L1 and W4 show where the requirement information and the accompanying service is transmitted. The normal lines only transmit the requirement data. Here, the full capabilities of JXTA can be utilized for peer-to-peer communication.

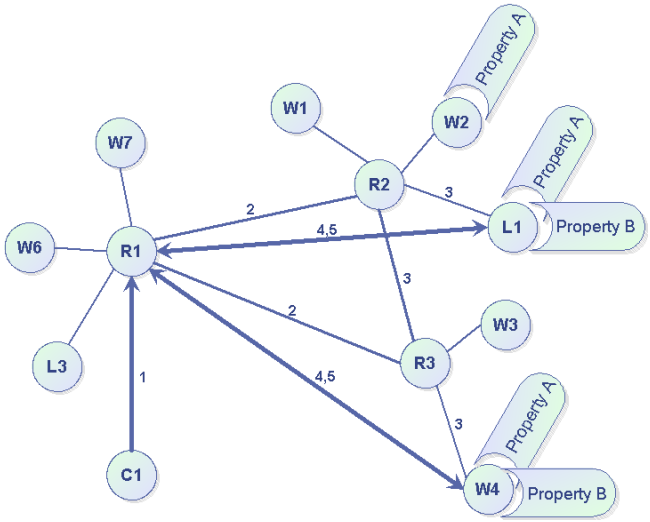

Fig. 7. Peer-to-peer based node property assessment

Just like the discovery service, the property scouts and the property scout rendezvous are also grid services and thus can be accessed as such.

C. Service Deployment

Automatic and non-disruptive service deployment is one of the most important requirements for an ad hoc grid. With increased adoption of the grid paradigm, the number of developers wanting to use the grid will increase, thus giving each developer administrative rights for all grid nodes is not feasible, especially in an ad hoc grid environment where personal computers are also members of the grid. We introduce a hot deployment service (HDS) allowing remote deployment of grid services without requiring the developer to have an administrative account on the system. A further vital benefit of our HDS is that deployment of a service can be done

non-disruptively, i.e. without requiring a restart of the OGSA platform after deployment. This is absolutely essential, since it is not acceptable that every time a new service is installed or an existing service is updated, all other services running in that environment are restarted as well – possibly loosing substantial amounts of work in progress. It would be just as undesirable as to restart an entire web server every time a web page is added, but currently this is common practice in service-oriented grid environments. If the grid is to become the next-generation Internet, hot deployment is indispensable.

Currently, GT3 does not offer any mechanism to hot deploy a grid service either remotely or locally. To remedy this, our hot deployment service undertakes the following steps to deploy a grid service into a running GT3 environment.

- Register the service description with the AXIS/OGSA request handlers.
- Make the schema files available to the OGSA environment.
- Make the service class files available to the class loader.

The pseudocode in Figure 8 describes these steps carried out to register the service dynamically:

```
with service.gar do {
  with wsdd_document do {
    root = registry.getRootNode()
    service_descriptor =
      wsdd_document.getServiceEntry()
    foreach property of service_descriptor
      do { serviceOptions.put(property) }
    serviceEntry =
      createServiceEntry(serviceOptions)
    root.bind(service_descriptor.name,
      serviceEntry)
    axisEngine.deploy(wsdd_document) }
  copy schema to schema_location
  copy jar to jar_location.service }
```

Fig. 8. Deployment algorithm

To register the newly deployed service with the OGSA application, the HDS needs to dynamically update the in-memory service registry of the OGSA application (the code within `with wsdd_document do`). Usually, these object structures are built when parsing the server configuration file upon startup of the application. This mechanism is not suitable in an ad hoc grid environment, since this would require a restart of the application.

Then, the schema files are copied to the location that was registered with the web application context by the OgsiServlet upon startup of the system (`copy schema to schema_location`). Finally, the OGSA application needs to be enabled to find and load the new classes that make up the service (`copy jar to jar_location.service`).

Additionally, it must be possible to refresh the implementation of these classes. In long running grid environments, it is likely that grid services will need to be updated at some point. The ability to do this in a running system is just as important as the ability to deploy services in the first place. However, the need to load additional classes and dynamically replace them was not anticipated or governed by the OGSA specification or its GT3 implementation. In the standard implementation of GT3, all service classes are loaded in one and the same ClassLoader. It is impossible to update a class definition as the Java ClassLoader does support reloading of classes. This effectively makes it impossible to update a previously installed service unless all contained classes are renamed or the GT3 environment is restarted. To enable a non-disruptive reloading of service classes, a ClassLoader hierarchy is introduced into our proof-of-concept realization of the HDS. Each grid service is loaded into its own disposable ClassLoader and if the grid service needs to be redeployed, the old ClassLoader is disengaged from the system and a new ClassLoader is put in its place. The new ClassLoader is then capable of loading the new class definitions and integrating them into the running system. It is also possible to run different versions of the same grid service on the same grid node, simply by using different ClassLoaders for each version.

Within GT3, a `FactoryCallback` implementation is responsible for loading the service classes. We provide the class `de.fb12.grid.deploy.HotFactoryCallbackImpl` as our own implementation of the `FactoryCallback` interface in order to leverage our own class loading mechanism into GT3. The factory callback is responsible for managing the ClassLoader hierarchy; it distinguishes different instances of the ClassLoaders by acquiring the service context from the AXIS engine inside the GT3 web application.

This is a non-intrusive way to introduce our own class loading mechanism into GT3, because the factory callback implementation can be specified in the service descriptor for every individual service. A service wishing to be hot deployable merely needs to be configured to use the `HotFactoryCallbackImpl`. This is the only change required to make a service hot deployable and reloadable. Hot deployable and standard services can be run side by side by using the different `FactoryCallback` implementations.

Figure 9 shows a snapshot of the ClassLoader hierarchy in the system, after three different grid services were instantiated. The GT3 environment and our hot service deployment mechanism are loaded by the Tomcat WebAppClassLoader. The remote clients call the deployment service where the HotFactoryCallBackImpl creates a DisposableClassLoader for each service creation request received. The GridServiceHandle is then returned to the clients. After the instantiation of the third service “C”, its implementation was updated and reinstantiated as “C*”.

This central component of our ad hoc grid now allows services to be installed on demand on nodes running the HDS.

D. Service Security

The standard security mechanism provided by GT3 offers access control mechanisms to guarantee that only authorized users can call grid services. We use this mechanism to restrict calls to the hot deployment service. As long as there is still a central authority granting or denying access which is trusted

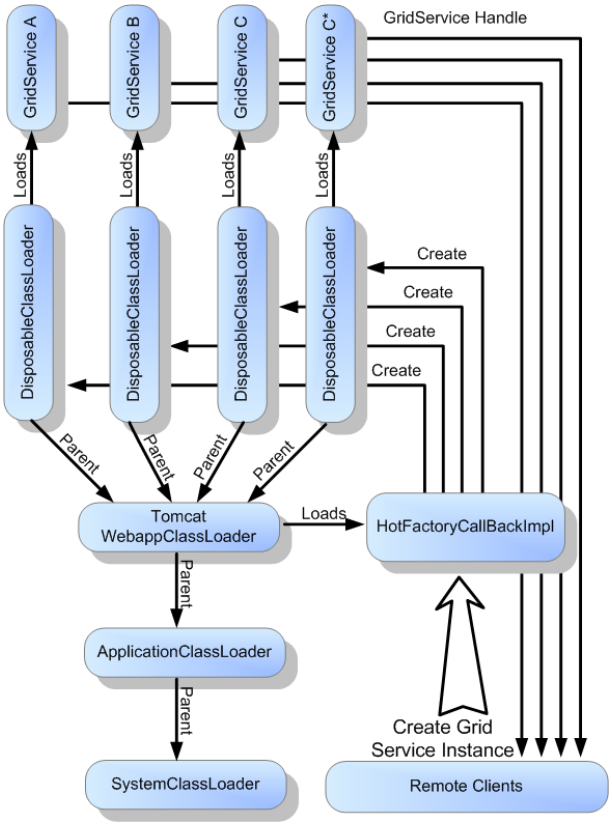

Fig. 9. ClassLoader hierarchy

by all, this mechanism suffices to secure the deployment side of the ad hoc grid. If, however, there is no central authority or trust can not be assumed between all participants of the grid (i.e. if A trusts B and A trusts C but B does not trust C), then further security mechanisms are required to offer intra-node security.

The current GT3 environment does not address these issues at all, since a central management and trust among the participants is assumed. To allow secure operation of grid services from different organizations on the same grid node, we offer an inter-service security model to ensure even if a malicious service is introduced into the grid, it can not harm other services running there.

Since the separate grid services running on one grid node are all hosted by GT3, they run within the same JVM. As a consequence, interaction between the classes is possible, thus offering malicious code the possibility to harm running services. For example, two possible scenarios of how a malicious service can cause harm to the system are:

- **Foreign Data Attack:** If the service targeted for attack uses singletons [22] (or any other method to access internal objects which does not require specific object references), it is possible to introduce a service which can modify these objects simply by using the same package as the target service and calling the objects normally. Using this method, the internal state of an object belonging to a different service can be modified.
- **Foreign Code Attack:** If the target service loads classes on demand (the standard procedure in Java), it is possible to introduce foreign code into the service by preempting the load procedure. If, for example, the target service has a Class A which loads Class B at a certain time during its operation, a malicious service can use that as an entry point for an attack. This is achieved by defining a Class B* with the same fully qualified name and with the same method signature as Class B and then loading that class within the attacking service. If this is done prior to the loading of B in the target service, malicious code has been successfully introduced into the system. When A tries to load B, the ClassLoader will see it has already loaded a class with the fully qualified name of B (namely B*) and loads the class from its cache. As a consequence, A now executes B* instead of B.

This form of inter-service attack is made possible by the fact that GT3 loads all grid services within the same class loader. The basic idea of our solution to this problem is to use the ClassLoader hierarchy introduced to enable the dynamic loading and reloading of classes to also provide inter-service security. In its most basic form, each grid service is loaded within its own ClassLoader and as such its classes and resources are private and can not be accessed by any other service. This ensures that services using singletons can not be hijacked by malicious services, and foreign code can not be smuggled into the program flow. Inter-service communication is confined to using web service calls and thus also ensures that proper authentication between services must be observed. If it is necessary that two services be able to communicate directly, they must place communication classes in a further ClassLoader which is a parent ClassLoader to each of the service ClassLoaders. These communication classes can now be accessed by both services. It is also possible to create different security spaces by creating different parent class loaders which have different rights within the system. For instance, using this method it is possible to restrict certain grid services from accessing the hard disk of a node.

The combination of the standard security model of GT3 in addition to the ClassLoader-based sandboxing of services offers the necessary security for our service-oriented ad hoc grid environment.

V. Related Work

Currently, several implementations of the Open Grid Services Infrastructure are being developed, including: Globus Toolkit 3.0 (GT3) [11], PyOGSI [23], OGSI::Lite [24], MS.Net Grid [25], OGSI.net [26] and ICENI [27].

The Globus Toolkit 3 (GT3) offers mechanisms to create service instances on a grid node via service factories. The GT3 does not offer a flexible mechanism for hot factory deployment (i.e. the deployment and removal of a grid service without restarting the OGSA web application inside the web application container). Instead, a service is deployed using two subsequent Apache Ant [28] tasks. The files making up a grid service are copied into the OGSA directory structure before the entire OGSA application is copied into the application container. To effect the changes, the application container must restart the OGSA application, causing a restart of every service

running in its context. The GT3 does not offer support for node discovery or node property assessment.

The PyOGSI project aims to make GT3 accessible through a Python interface. It does not address the problems of service deployment, node discovery or node conformity assessment.

OGSI::Lite attempts to provide an OGSI implementation in the Perl language. Every OGSI::Lite service is run as a Perl CGI module inside Apache and can therefore be dynamically replaced by replacing the file on the server. The use of Perl, however, creates some deficiencies, such as hard to understand and reuse code, only rudimentary support for WSDL creation – the authors suggest to write Java interfaces and generate WSDL definitions for the Perl services from them – lack of feature implementations in SOAP::Lite and the missing support of the multitude of Java web service developments (e.g. web service security). Furthermore, node discovery or node property assessment is not supported by OGSI::Lite.

Both MS.NETGrid and OGSI.net use the Microsoft Internet Information Server (IIS) as their hosting container. Dynamic deployment of services is not addressed as deployment requires the alteration of the container's configuration file, copying of service descriptions as well as service assemblies (service DLLs) and the restart of the IIS. Both do not offer node discovery or node property assessment.

Finally, the ICENI [27] project is aimed at creating a service-oriented architecture for grid computing. It was originally [29] built on Jini technology [30] and was recently ported to JXTA and OGSI. The OGSI implementation of ICENI uses GT3 only as a communications provider rather than extending the functionality of the OGSI implementation. ICENI services are exposed as OGSI grid services via a complex process using introspection and the generation of proxy classes. ICENI does not combine the features of JXTA, OGSI and Jini, it simply uses them as alternative solutions.

VI. Conclusions

In this paper, we introduced a service-oriented ad hoc grid as a spontaneous fusion of cooperating heterogenous computing nodes into a logical community without a preconfigured fixed infrastructure and with only minimal administrative requirements. The main goal of an ad hoc grid is to provide computing resources on demand to every participant. We have discussed the main issues involved in building such an ad hoc grid. Approaches to solve the problems of node discovery, node property assessment, hot service deployment and inter-service security were presented. Furthermore, a proof-of-concept realization of these components in an ad hoc grid based on GT3 was described. All components can be inserted non-intrusively by simple configuration changes and do not require any alteration of the GT3 code.

There are several areas for future work. For example, an interesting issue is to provide support for versioning, allowing multiple versions of a grid service to be deployed and run side by side. Another area is support for deploying legacy applications, thereby advancing the already provided support for native platform code included in a service implementation. Furthermore, the introduction of predefined security spaces allowing different levels of access to a node's resources is a topic for future work. Finally, a goal driven self-healing automation of the entire deployment process required to install and maintain a fully fledged grid application capable of dealing with dynamic node failures should be realized.

References

[1] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[2] F. Berman, G. Fox, and T. Hey, *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, 2003.

[3] NASA Advanced Supercomputing Division, "NASA Information Power Grid," 2002, http://www.nas.nasa.gov/About/IPG/ipg.html.

[4] U.S. Department of Energy Office of Science, "Department of Energy and IBM Science Grid," 2001, http://doesciencegrid.org.

[5] EGEE, "Enabling Grids for eScience in Europe: Executive Summary," 2004, http://egee-intranet.web.cern.ch/egee-intranet.

[6] "IBM Making a Commitment to the Next Phase of the Internet," http://marianne.in2p3.fr/datagrid/documents/Ibmgrid.pdf.

[7] I. Foster and A. Iamnitchi, "On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing," in *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003, pp. 118–128.

[8] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language," 2001, http://www.w3.org/TR/wsdl.

[9] The World Wide Web Consortium, "Simple Object Access Protocol (SOAP)," 2003, http://www.w3.org/TR/soap/.

[10] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," in *Open Grid Service Infrastructure WG, Global Grid Forum*, 2002.

[11] "The Globus Toolkit 3.0," http://www-unix.globus.org/toolkit/.

[12] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky, "SETI@home - Massively Distributed Computing for SETI," *Computing in Science and Engineering*, vol. 3, no. 1, p. 79, 2001.

[13] D. Gannon, R. Ananthakrishnan, S. Krishnan, M. Govindaraju, L. Ramakrishnan, and A. Slominski, "Grid Web Services and Application Factories," in *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and T. Hey, Eds. Wiley, 2003.

[14] "Apache Tomcat 5.0," http://jakarta.apache.org/tomcat/index.html.

[15] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-Like P2P Systems Scalable," in *Proc. of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM Press, 2003, pp. 407–418.

[16] "Limewire," http://www.limewire.org/.

[17] N. S. Good and A. Krekelberg, "Usability and Privacy: A Study of Kazaa P2P File-Sharing," in *Proceedings of the Conference on Human Factors in Computing Systems*. ACM Press, 2003, pp. 137–144.

[18] A. Iamnitchi, I. Foster, and D. Nurmi, "A Peer-to-Peer Approach to Resource Discovery in Grid Environments," in *Proc. of the 11th Symposium on High Performance Distributed Computing*, 2002, p. 419.

[19] A. Iamnitchi and I. Foster, "On Fully Decentralized Resource Discovery in Grid Environments," in *International Workshop on Grid Computing*, 2001, pp. 51–62.

[20] M. Murshed and R. Buyya, "Using the GridSim Toolkit for Enabling Grid Computing Education," in *Proc. of the Int. Conf. on Communication Networks and Distributed Systems Modeling and Simulation*, 2002.

[21] *Project JXTA v2.0: Java Programmers Guide*, 2003.

[22] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*, 1997.

[23] Lawrence Berkeley National Laboratory, "pyOGSI," 2003, http://www-itg.lbl.gov/gtg/projects/pyOGSI/.

[24] "OGSI::Lite," http://www.sve.man.ac.uk/Research/AtoZ/ILCT.

[25] D. Byrne, A. Hume, and M. Jackson, "Grid Services and Microsoft .NET," in *UK e-Science All Hands Meeting*, 2003, pp. 129–136.

[26] G. Wasson, N. Beekwilder, M. Morgan, and M. Humphrey, "OGSI.NET: OGSI-Compliance on the .NET Framework," to appear in ccGrid 2004.

[27] N. Furmento, J. Hau, W. Lee, S. Newhouse, and J. Darlington, "Implementations of a Service-Oriented Architecture on Top of Jini, JXTA and OGSI," in *Proc. Second Across Grids Conference*, 2004.

[28] "The Apache Ant Project," http://ant.apache.org.

[29] N. Furmento, J. Hau, W. Lee, S. Newhouse, and J. Darlington, "ICENI: An Open Grid Services Architecture Implemented with Jini," in *Proc. of UK e-Science All Hands Meeting*, 2003, pp. 703–120.

[30] "Jini Network Technology," http://www.sun.com/software/jini.