

Distributed Ant: A System to Support Application Deployment in the Grid

Wojtek Goscinski and David Abramson
{wojtekg,davida}@csse.monash.edu.au

School of Computer Science and Software Engineering, Monash University

Abstract

e-Science has much to benefit from the emerging field of grid computing. However, construction of e-Science grids is a complex and inefficient undertaking. In particular, deployment of user applications can present a major challenge due to the scale and heterogeneity of the grid. In spite of this, deployment is not supported by current grid computing middleware or configuration management systems, which focus on a super-user approach to application management. Hence, individual users with limited resource control deploy applications manually which is not a grid scalable solution. This paper presents our motivation, design and implementation of a grid scalable, user-oriented, secure application deployment system, Distributed Ant (DistAnt). DistAnt extends the Ant build file environment to provide a flexible procedural deployment description and implements a set of deployment services.

1. Introduction

The computational grid has the capability of providing a powerful infrastructure for ubiquitous computational resource sharing [1]. Existing grid middleware provides essential services for executing grid applications, such as resource management, information services and data management. Virtual Organizations [2] might span different machines, platforms, architectures, and even organizational and international boundaries. The development of applications for this wide landscape is difficult and software tools and environments are necessary to help in the development of these applications. In particular it has become apparent to us that challenges in deployment and configuration of applications over such a diverse landscape introduces a barrier to the adoption of a truly dynamic grid.

The motivation for our research is based upon experiences in organizing and developing large scale computational grid experiments. At the IEEE Supercomputing Conference (SC2003) in Phoenix, Arizona, together with colleagues from the University of California, San Diego (UCSD), our research group per-

formed a large e-Science experiment using a computational grid [3]. The experiment was executed using parametric sweep software developed by our research group, Nimrod/G [4], which coordinated around 50,000 instances of the GAMESS quantum chemistry package at up to 30 super computers in 10 countries. This major e-Science experiment taught us a great deal about the difficulty in using large distributed infrastructure, such as the Grid, for daily e-Science. While the computation ran for five days, two research assistants worked day and night for two months before the conference to set up the infrastructure. In particular, application deployment represented a huge proportion of both the technical and organizational effort required to implement this experiment, demonstrating the complete lack of support for application deployment within grid middleware. The lack of middleware support for application deployment presents a challenge to scientists utilizing computational grid infrastructure.

The focus of this paper is the development of a new tool called Distributed Ant (DistAnt), an automatic distributed application deployment system with a focus on grid applications. DistAnt provides a simple, scalable and secure deployment environment. It is unique because it approaches application deployment from a grid perspective. The grid can be characterized as an environment where users have specific resource requirements, but are not necessarily provided with high levels of resource authority or functionality to implement those requirements. Accordingly, DistAnt provides a system that allows users with limited resource authority to easily deploy applications. This is in contrast to typical configuration management systems which assume a system administration, super-user oriented view of resource management which doesn't authorize resource users with application deployment.

For the purposes of this paper, we define application deployment as the range of actions, including file transport, installation and configuration, required to setup a program on a particular resource, so that it is ready for instantiation. We do not limit application deployment to 'grid services' because we believe the development of traditional grid applications will grow in parallel with the growth in development of grid services. Furthermore, this

work recognizes the continued importance of legacy applications and the need for tools and mechanisms to 'grid enable' them.

This paper is structured as follows: Section 2 presents more information about the motivation for DistAnt and details background and research in this area. Section 3 describes a proposal for the logical design of a grid deployment system. Section 4 characterizes OGSA and Ant, the two major technologies used in the construction of DistAnt and discusses the implementation of DistAnt. Section 5 draws conclusions from this work and introduces further research.

2. The Need for a Grid Application Deployment System

The development of the e-Science experiment discussed in Section 1 demonstrated the need to provide a mechanism for easy software deployment. The grid constructed for the experiment was representative of grids in general; being multi organizational and technically heterogeneous. This caused obvious challenges:

- It included multiple test beds managed by different organizations, such as the Pacific Rim Applications and Grid Middleware Assembly (*PRAGMA*), The Australian Grid Forum (*AusGrid*) and the US-based *TeraGrid*. Organizational heterogeneity commonly implies different resource administration and operating procedures. This complicated application deployment due to different resource access, user permissions, accounts and installation procedures.
- The experiment included a range of architectures (x86, ia64, sparc, MIPS) and operating systems (varying flavors of Linux, SunOS and IRIX). Technical heterogeneity complicates application deployment due to different build & executable files, library dependencies, file systems and installation procedures.

In the lead up to our experiment, programs were installed using the technologies available; `ftp` to transport files and `ssh` to perform a manual installation and configuration. This installation method is not grid-scalable and is far from user friendly. The research assistants performing the installations were computer scientist, not e-Scientists. Unless setting up such experiments becomes more efficient, grids will fall short of their potential for direct use by scientists.

The Grid Resource Allocation Manager (GRAM) [15] successfully minimizes the complicating effect of technically and organizationally heterogeneous systems by providing a single application instantiation interface and instantiation request specification. In turn, application deployment can benefit from a similar approach. Providing a single well known interface to application deployment reduces problems raised due to organizational

heterogeneity and technical heterogeneity. The problems incurred through technical heterogeneity are further reduced through use of a single system-independent procedural deployment description.

Automated application deployment is generally viewed as a part of configuration management, which encompasses the wide range of actions of taking a blank machine and performing the necessary steps to turn it into a functional system. This might include operating system installation and configuration, application installation and configuration, and updating configuration changes. A representative handful of resource and configuration management systems are summarized below, in particular we describe examples of: a commonly used cluster management system, NPACI Rocks; declarative configurations systems LCFG and SmartFrog; and research specifically directed towards configuration management within a grid environment, GridWeaver.

NPACI Rocks [5] provides node installation from a bare machine and a software configuration mechanism based on package dependencies amongst software. RedHat kick start files and Linux RPMs are used to automate the system installation process. NPACI Rocks provides a "no-nonsense" approach to resource management by completely reinstalling nodes from the operating system up when they have failed or when there is a requested change in their installation or configuration.

The LCFG (Local Configuration) [6] system provides a mechanism for node installation from the operating system up, specifically targeting Linux. Each node is assigned a centrally located `lcfg` profile which defines operating system and software. Nodes run a LCFG client which consults its profile for packages it should install. It is a declarative system, meaning users write declarative system descriptions rather than procedural descriptions of the actions required to construct that system.

SmartFrog [7] defines a framework for constructing configurable software systems. It defines the declarative SmartFrog language which describes: individual components; their hierarchies, relationships, and locations; in what order they are run; and how they combined to form a complete system. This language is interpreted by the SmartFrog distributed runtime engine which performs actions to realize the description.

GridWeaver [8] is a prototype system constructed using the SmartFrog framework and LCFG. The system assumes that SmartFrog and GridWeaver are mutually supportive technologies. LCFG provides a mechanism to completely rebuild a machine, while SmartFrog provides a component management framework which assumes existing underlying resources.

The characterized configuration systems are focused on or originate from a cluster management perspective. They implement a centralized mechanism to provide central management and configuration over systems

within institutional control for the purposes of system administration. Conversely, our introduction of a grid perspective to application management requires that users have authority to deploy applications on resources they do not necessarily control. The grid focus on sharing of resources implies the loss of complete central control. Resource users on the computational grid will commonly need specific applications and relying on complete central control to provide these applications is not scalable. For example, using NPACI Rocks, a change in the system description would result in a complete rebuild of the system, which is not desirable for a system that might be shared by many users.

In addition to configuration systems, grid middleware systems providing service hosting are an interesting comparison to our own system goals due to their service oriented view of grid applications. Of particular interest is the mechanism these systems implement (or do not implement) to deploy and manage services. The systems characterized are the Globus Toolkit 3 [10], H2O [13] and HARNESS [14].

Globus Toolkit 3, discussed in greater detail in the implementation section, provides a service hosting container. Like the configuration management systems discussed, GT3 provides a central service management mechanism. It allows users to lookup and instantiate already deployed services using a factory model. The GT3 service hosting environment is static, meaning it requires a restart to refresh deployed or updated services.

H2O defines a Java based container and service model using the terms kernel and pluglet. H2O is the progression of the HARNESS project. Both environments are interesting because unlike GT3 they allow clients to deploy services. H2O builds on this functionality by allowing the resource provider to set container policy on who may deploy pluglets, the types of pluglets they may deploy and priorities. H2O implements the same user oriented view of grid application deployment that we're interested in providing. However, like GT3 it also provides a complete Java service container environment and thus forces application developers to implement a specific Java interface or wrap their code. Furthermore, we believe not all future grid applications will be implemented as grid services, reliant on an application hosting environment. Therefore, any application deployment system should cater to a wide range of applications.

Unlike all the systems characterized, our research is solely focused on deployment; we make the assumption that operating system and grid middleware installation and configuration has already been performed. In this sense, our work could be integrated with a central configuration system which manages the underlying grid fabric or a grid service container which hosts deployed applications.

We want to utilize and integrate into stable grid environments, with a large user base, such as the Globus Toolkit. However, heterogeneity in underlying grid middleware, system configuration and system configuration management systems is representative of the grid. Therefore, a deployment system for the grid should be flexible and sufficiently generic to complement existing grid middleware and configuration management systems.

A mechanism for application deployment is also important from a software development perspective. This is particularly true of grid computing where programs might be developed for very specific high-demand resources with specific characteristics, or generic computational resources to run opportunistically on any available location. In either case, complete development requires deployment and testing on remote resources. The entire process leading up to and including deployment requires a number of steps: compilation; linking; packaging; file transport; un-packaging / installation; and configuration. And finally, typically, this process leads to application instantiation to perform testing. It is the goal of our work to provide an overall mechanism to describe and automate all these steps. The first steps; compilation, linking and packaging, are easily described and automated through a make system such as Make or Ant. Un-packaging, installation and configuration can also be easily provided through a build file system on the local machine. However, build file systems typically focus on providing functionality on the local system and do not extend their functionality to remote locations. It is the aim of this work to define a procedural deployment description analogous to a standard build file system.

The GridAnt [19] system takes a similar approach to DistAnt; it utilizes Ant extensibility and workflow to provide a client side grid workflow system. GridAnt implements Ant tasks as an interface to common grid activities such as file transfer, job submission, information query and checkpointing. GridAnt and DistAnt share the characteristic of consolidating multiple client side actions into workflows and the use of Ant. However, unlike GridAnt, our work is specifically focused towards application deployment; addressing both client and server side requirements and issues such as remote configuration and resource heterogeneity.

3. DistAnt Design

This section describes the architecture and functionality of DistAnt, an application deployment service for the grid, which fulfills the considerations raised in Section 2. In addition this section describes how our system approaches system heterogeneity and security considerations.

The logical architecture for a grid application deploy-

ment system consists of three components:

- A flexible front end procedural deployment description to allow users to describe remote file transport, configuration and instantiation in the same way that they would describe compilation, linking and packaging. The range of possible grid applications might include: newly developed Java or .NET programs which support a simple copy and paste deployment; to legacy programs which might require very specific installation and configuration steps. This wide range of possible grid applications requires a flexible procedural deployment description. Therefore, it is logical to utilize an existing build file system, providing the advantages of building on an established system with a wide range of functionality and an existing user base.
- A deployment client to co-ordinate the deployment with remote deployment servers. A deployment might involve transporting multiple files to one or more locations or performing deployment actions on one or more locations. A client side application is required, to initiate requests, in parallel, to remote locations, and receive back results.
- A deployment server which sits at remote locations and services deployment requests; providing each deployment with an individual file system space.

The front end build file system should implement the following functionality, which we will generally refer to as 'deployment actions':

- File transfer, to provide a means of transporting files to remote locations.
- Remote actions to execute procedural actions on remote machines. This allows the user to write routines which unpack, install and configure applications at a remote location.
- Application instantiation to provide a mechanism for instantiating the deployed application.

Together, these deployment actions allow a user to move application files, install, configure and finally instantiate a deployment. To identify a particular installation, we propose an *application namespace* which is a user defined string, identifying a particular application from others that may have already deployed.

Providing a means of fully handling technical & system heterogeneity is beyond the scope of this paper. However, the design of DistAnt provides mechanisms to minimize the effect of system heterogeneity. In particular, the definition of a single deployment mechanism reduces problems associated with resource access, while the definition of a flexible procedural deployment description means the user would typically only need to write one deployment description for many different resources.

DistAnt undertakes minimizing the differences associated with file system heterogeneity by using a *deployment space* and *deployment references*. Each application

deployment is performed within its own *deployment space*, which is a directory created and managed by the deployment server. *Deployment references* are used to reference a particular file or application at one or more locations, without knowing the specific details of that file or applications location. Each individual deployment server is able to substitute *deployment references* for their actual local file system location. References allow the writer to write generic procedural deployment actions to execute on different resources, much like they would write generic code functions using variables.

The application deployment service presented acts as a proxy, performing actions on behalf of the user. In turn, the deployment service becomes aware of its installed programs, while the user can ignore certain aspects of that installation. However, application instantiation (for example, through the Grid Resource Allocation Manager (GRAM)) requires the client have very specific knowledge of an applications deployment characteristics, such as the executable location. Automatic deployment makes writing a Resource Specification Language (RSL) document (GRAM execution request) difficult without specific information about the application installation. We provide a solution to this by proposing a application instantiation task. This task uses a combination of the *application namespace* and *deployment references* to reference a particular deployment and file for application instantiation. This provides a simple mechanism for instantiation and allows users to quickly test applications directly after deployment. This mechanism is not meant as a replacement or reinvention of widely used allocation mechanisms such as GRAM, but rather an interface to those systems. This provides interoperability with the existing resource management service and utilizes specific advantages that those systems offer.

Security is of particular concern in a remote deployment system. It is not this system's goal to protect the system against authorized users installing malicious applications. However, it is the responsibility of a deployment system to ensure that application deployment is authenticated to a user and performed within the existing organizational security infrastructure. Authentication can be achieved using a public private key infrastructure, while executing deployment actions under the authenticated user ensures that deployments occur within the existing organizational security infrastructure.

4. DistAnt System Implementation

This section describes the implementation of DistAnt. Section 4.1 describes and justifies the two technologies on which DistAnt is built, GT3 and Ant. Section 4.2 describes the technical architecture and implementation.

4.1 Background Technologies

The Open Grid Services Architecture (OGSA) [9] was developed to provide a common, homogeneous external view of the wide ranging heterogeneous resources that make up the grid. OGSA is defined over the Open Grid Services Infrastructure (OGSI) specification, which defines a virtualization of grid resources by formalizing a lowest level, universal view of grid entities, a Grid Service. This defines a common view and means of communication regardless of the heterogeneity of underlying hardware and software. Globus Toolkit 3 is the open source reference implementation of OGSA, written in Java over a standard Web Service engine, Axis. In addition to implementing the OGSA specification GT3 provides system-level and base-level services. System-level services are general use services for administrative and development purposes. Base-level services provide specific grid oriented functionality:

- Program execution, GT3 GRAM [17] provides job execution and management;
- Data management, GT3 provides an Grid Service interface to GridFTP [12] for file transport; and
- Information services, GT3 provides services to publish commonly used resource information.

The OGSA / GT3 infrastructure is used in DistAnt as the basis of our deployment client / server architecture. It provides DistAnt with a common, secure, services based middleware for building and hosting grid services. It is developed for heterogeneous resources and provides functionality DistAnt requires, such as program execution and file transport.

Due to the release of the WS-Resource [16] Framework, it is not expected GT3 will be as widely supported as its predecessor GT2. However, it is expected that the GT3 specific DistAnt clients and services developed will easily be ported to the WS-Resource Framework or Web Services in general. To minimize dependence on one particular infrastructure and to provide for implementations within other environments, DistAnt was implemented generically, with a GT3 wrapper.

Ant [11] is an XML based procedural build-file system, written in Java. Ant build files consist of Tasks, Targets, Properties and a Project:

- Tasks define an action or element of functionality. Common types of tasks includes: compile; archive; execution and file related tasks [11].
- Targets group together Ant tasks procedurally and define an overall logical action, such as compilation and linking, packaging or cleaning. Ant implements a workflow through definition of dependencies amongst targets.
- Properties, every project has a set of properties which consists of Ant properties, Java properties, system properties and user defined properties.

- Project, the root XML structure of an Ant makefile, wraps all targets and properties.

Ant is the most appropriate for our purposes due to its platform independence and extensibility. Other build file systems, such as *make* or *gnumake*, which could be considered for this project are shell based and therefore are dependant on a particular operating system. Ant sacrifices the ability to perform complicated shell constructs for the advantages of platform independence using XML and Java. It provides extensibility by allowing custom tasks.

We use Ant as our procedural application deployment description mechanism because it defines a workflow and a wide set of build file oriented functionality. We extend Ant functionality in two ways: Defining custom tasks as an interface to deployment functionality; and defining properties for use as deployment references to remote files and deployed applications.

4.2 Development of a Grid Application Deployment system

This section describes the architecture and implementation of DistAnt. In particular, we present the implementation of the DistAnt architecture, deployment actions and deployment references. We also provide an example DistAnt build file and discuss the implementation of security mechanisms.

As introduced in Section 3 DistAnt consists of three major components (see figure 1):

- Custom Ant tasks – Extensions to the Ant build file system, including the three deployment actions introduced in section 3.
- The DistAnt Client – A local client to co-ordinate requests to one or more remote deployment services.
- The DistAnt Service – A GT3 remote deployment service on each remote resource to accept, process requests, and manage deployments.

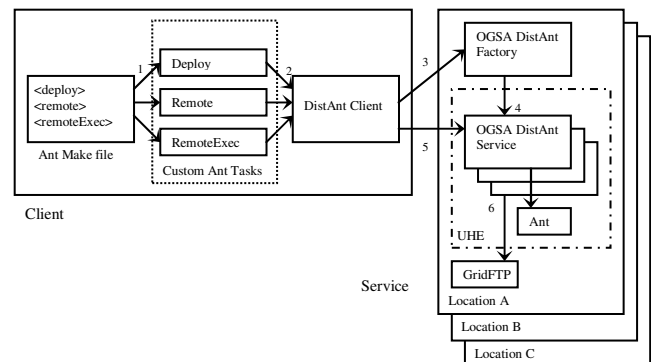


Figure 1. DistAnt Architecture.

The following describes a typical sequence of events within the DistAnt architecture when processing a

deployment action, with references to figure 1. This is provided as an introduction to both the DistAnt architecture and the implementation of specific DistAnt deployment actions, which are described later.

1. Ant parses the build file and when appropriate it executes the appropriate DistAnt custom task.
2. The DistAnt custom task creates an instance of the DistAnt Client, which remains for the entire instantiation of Ant. The custom task passes information about the deployment action request to the DistAnt Client, including: the specific action; remote server locations; and any parameters specific to that particular action.
3. When it receives a request from a custom task, the DistAnt Client contacts each address location (which represent a remote DistAnt Factory) requesting an instance of a DistAnt Service.
4. The DistAnt Factory instantiates the DistAnt Service within a User Hosting Environment (UHE), running under the requesting user's user account, and passes the address back to the requesting client. The instantiated DistAnt Service remains to service further requests until the DistAnt Client is terminated.
5. The DistAnt Client requests the DistAnt Service directly with the action request. The DistAnt Service uses the *application namespace* provided by the client and the requesting user authenticated details to map to a specific *deployment space*. This mapping is stored in an XML document the DistAnt Service uses to provide persistent storage about deployments.
6. The DistAnt Services interfaces directly with Ant or GridFTP to perform the requested action.

Requests for deployment actions are asynchronous and with each request the DistAnt Client receives back a request identifier. The DistAnt Client registers to the DistAnt Servers service data to receive request results and uses the unique identifier to differentiate requests.

The three deployment actions available are: *deploy*, *remote* and *remoteExe*. They are all implemented from the same DistAnt *genericRemoteTask* class, which defines a basic level of functionality required to execute a particular action on one or more remote resources.

Figure 2 shows an example of a DistAnt Ant file and how these three actions are used by the user. The example transports a file to two remote locations (*deploy*), un-tars the file at the two locations (*remote*) and then executes the un-tarred executable (*remoteExec*). *deplRef1* demonstrates deployment references; the deploy task sets *deplRef1* as the deployment reference representing the deployed file, and it is then used in the remote untar task. *appNamespace1* demonstrates the *application namespace* used to identify this deployment.

The following describes the implementation and functionality of the three DistAnt tasks, deployment

references and dependencies.

deploy provides a mechanism for deploying a file or directory structure to one or multiple resources. File transport could have been achieved through the standard Ant ftp functionality. However, the standard ftp functionality is limiting:

- We require a file transport task which has been developed with the greater goal of application deployment. *deploy* provides support for multiple destinations and allows the user to define a deployment reference to the remote file, which can then be used as a reference within future actions.
- ftp is limiting in a grid environment. *deploy* uses GridFTP which provides a secure data transport mechanism which has been designed specifically for a grid environment. However, other transport mechanisms can be plugged in if necessary. These might include standard ftp or http.

```
<?xml version="1.0"?>
<project name="DeploymentExample" default="remoteExe"
basedir=".">
  <property name="appNamespace1"
location="www.monash.edu.au/exampleInstallation1" />
  <target name="deploy">
    <deploy
src="packagefile.tar"
dest="{ appNamespace1}"
handle="deplRef1"
transport="gridftp"
localGridFTPLocation
="gsiftp://memnon.csse.monash.edu.au:21/">
    <urlList>
    <urlDataType url="http://memnon:8080/ogsa/
services/installer/registry/InstallerRegistry"/>
    <urlDataType url="http://meerkat:8080/ogsa/
services/installer/registry/InstallerRegistry"/>
    </urlList>
  </deploy>
</target>
<target name="unpack" depends="main">
  <remote>
    <urlList>
    <urlDataType url="http://memnon:8080/ogsa/
services/installer/registry/InstallerRegistry"/>
    <urlDataType url="http://meerkat:8080/ogsa/
services/installer/registry/InstallerRegistry"/>
    </urlList>
    <untar src="{ deplRef1 }"
dest="{ appNamespace1}">
    </untar>
  </remote>
</target>
<target name="execute" depends="unpack">
  <remoteExec
deploymentNamespace="{ appNamespace1}"
executable="packagefile.exe"
rs1Save="localRSLFile.xml">
    <urlList>
    <urlDataType url="http://memnon:8080/ogsa/
services/installer/registry/InstallerRegistry"/>
    <urlDataType url="http://meerkat:8080/ogsa/
services/installer/registry/InstallerRegistry"/>
    </urlList>
  </remoteExec>
</target>
</project>
```

Figure 2. Example DistAnt build file, showing examples of the three DistAnt actions; *deploy*, in the deploy target; *remote*, in the unpack target; and *remoteExec* in the execute target. Due to space restrictions Ant class definitions have been removed.

remote provides an Ant custom action for wrapping Ant functionality to be performed at a remote host. Any

ant task or tasks which would normally be performed locally can be wrapped in the *remote* task to note that it should be performed remotely. This allows the user to write ant procedures which perform remote configuration.

remote is implemented as a nesting custom task, meaning other tasks can be nested inside. Ant initializes *remote* by passing it all its nested tasks. *remote* then initiates the process of executing itself on the remote resources listed. *remote* passes information to the DistAnt Client, which initiates requests to the listed DistAnt Services. Making this request, it passes a number of parameters which are required to service a *remote* request:

- The Ant build file. The entire Ant build file is passed to the DistAnt Service which saves it to disk and executes Ant to run the requested *remote* target. The entire file is sent because the section of Ant build file requested might call other targets. DistAnt does not preprocess the build file to determine which sections need be sent and which can be ignored.
- Ant user properties. Any properties set by the user, up until the execution of *remote*, are passed to the remote resource.
- The name of the *remote* target.

The DistAnt Service processes the request and instantiates Ant to execute the requested target locally within the deployment space.

Writing *remote* tasks imposes two restrictions on the user: *remote* tasks must be written directly within their own target; and the user should reference files using deployment references. Referencing specific files, though not prohibited, might result in a general Ant 'file unknown' exception, which is passed back to the local client and user. More generally, all remote exceptions are serialized and passed back to the DistAnt Client.

Executing an Ant target on a remote machine will result in user output. This output is captured by the DistAnt Service and passed back to the DistAnt Client using the DistAnt Services service data. In future, we plan to stream output back to the DistAnt Client.

remote utilizes the power of the existing procedural Ant build file functionality because the user is able to remotely execute the many Ant tasks available locally. This includes the condition tasks which allow the user to perform functionality based on conditions. The *os* condition is of particular interest because it allows users to perform conditions based on the operating system, architecture and operating system version.

remoteExec provides a mechanism to remotely instantiate a deployed application by providing an Ant interface to resource allocation systems. The current implementation interfaces with GT3 GRAM, hiding the specifics of generating an RSL document request. *remoteExec* takes a number of parameters including the *application namespace*, executable location (which can be constructed using a *deployment reference*) and an option

to instantiate the application automatically or save the generated RSL document for the user to utilize manually. *remoteExec* initiates the following sequence of events:

1. *remoteExec* is executed by Ant and passes the request to the DistAnt Client.
2. The DistAnt Client passes the request to the particular DistAnt Service, which generates a GRAM RSL document based on the request, filling in specific information that GRAM requires but the client might be unsure of, including the specific installation directory. It passes back both the generated RSL document and a reference to the GRAM factory the client should use.
3. The DistAnt Client receives back the RSL document and can initiate a job request to the remote GRAM service. Otherwise, it can save the RSL document to disk for the user to submit manually.

DistAnt implements *deployment references* as Ant properties. Deployment references are defined by the user, who can use that reference in later Ant tasks. When executing a *deploy* action, the DistAnt Service saves the user defined *deployment reference*, along with the actual file location. Subsequent *remote* or *remoteExe* actions at that location can reference the deployed file using the *deployment reference*. The DistAnt Service stores the relationship between *deployment reference* and specific local file location. When servicing remote and remoteExe requests, the DistAnt Service exchanges references with their real local file system location. An example deployment reference is demonstrated in figure 2.

Ant includes a mechanism for defining dependencies amongst targets. However, dependencies are more complicated in DistAnt, due to the fact that a single Ant action might succeed at a particular location, but fail at another location. Dependencies are defined in DistAnt in the same way as normal Ant files; using the *depends* keyword in target definitions (see figure 2 for an example). However, internally the DistAnt Client manages dependencies individually per location. If a particular DistAnt target fails at a particular location, causing a remote exception, its dependant tasks will not continue executing at that location. However, The DistAnt Client does not cease the entire Ant workflow unless all locations have failed.

The current implementation of DistAnt synchronizes all locations at the end of each deployment action, meaning only individual deployment actions are performed in parallel. Planned future work includes parallelism for sequences of deployment actions.

Using Ant as the basis of DistAnt provides major advantages:

- It allows users to easily write deployment descriptions based on their knowledge of how their program should be installed;
- It is as powerful and flexible as the underlying Ant system; and

- It allows users to integrate the process of deployment and testing with general development.

Users write application deployment descriptions using their knowledge of an existing build file system, with a small set of powerful extensions.

The design of DistAnt calls for a User Hosting Environment architecture to provide security to the deployment process. A UHE ensures the deployment occurs within the stable institutional based user permissions infrastructure. DistAnt reuses the User Hosting Environment provided by GT3 GRAM. User authentication occurs through the GSI [18] security infrastructure and the DistAnt Factory spawns a UHE executing under the account of the authenticated local user. The DistAnt Service executes within this hosting environment, performing all user requests under the local user. This implementation ensures that users are authenticated to a local resource user and perform deployments within the space and permissions allocated to that user.

5. Conclusion

Providing mechanisms to support user application deployment is an essential part of the fabric that supports a simple ubiquitous grid. The multi-organizational and technically heterogeneous nature of the grid provides a complicated environment for end users. End user support is essential to advance usability and promote wide scale adoption of the field of grid computing. In particular, end user deployment presents a challenge to grid usability and is unsupported by existing grid middleware. DistAnt provides a significant contribution to the goal of developing usable computational grids because it fills a gap in end user support in grid middleware. DistAnt provides a dedicated secure mechanism for decentralized application deployment. Thereby, it simplifies the end user process of constructing computational grids and e-Science experiments.

This paper presents research in progress. Therefore, DistAnt is not a finished implementation and we have plans for future work. DistAnt has undergone initial testing. More in depth testing will present an interesting opportunity to gauge the effect of DistAnt. We expect this will prove interesting for future publication.

DistAnt introduces new research questions which are of interest to our project. These include integrating DistAnt more closely with existing grid middleware or configuration management systems. Furthermore, DistAnt promotes application deployment on heterogeneous resources but cannot guarantee that a particular application will execute on any particular resource. This project provides an opportunity to continue research into how to best fully support system heterogeneity in an application deployment system.

References

- [1] I. Foster, C. Kesselman. "Computational Grids", Chapter 2 of "The Grid: Blueprint for a New Computing Infrastructure", Morgan-Kaufman, 1999.
- [2] Ian Foster, Carl Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal of Supercomputer Applications, Sage Publications, 15(3), 2001.
- [3] Sudholt, W., Baldrige, K., Abramson, D., Enticott, C. and Garic, S. "Parameter Scan of an Effective Group Difference Pseudopotential Using Grid Computing", New Generation Computing 22, 2004, 125-135.
- [4] D. Abramson, J. Giddy and L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?", International Parallel and Distributed Processing Symposium (IPDPS), Cancun, Mexico, 2000, 520- 528.
- [5] G. Bruno, P. Papadopoulos and M. Katz., "Npaci rocks: Tools and techniques for easily deploying manageable linux clusters". Cluster 2001, 2001.
- [6] P. Anderson and A. Scobie. "LCFG: The Next Generation", UKUUG Winter Conference, 2002.
- [7] P. Goldsack, J. Guijarro, A. Lain, G. Mecheneau, P. Murray, P. Toft, "SmartFrog: Configuration and Automatic Ignition of Distributed Applications", HP Labs, Bristol, UK, 2003, <http://www.hpl.hp.com/research/smartfrog/>
- [8] P. Anderson, P. Goldsack, J. Paterson, "SmartFrog meets LCFG - Autonomous Reconfiguration with Central Policy Control", Proceedings of the 2002 Large Installations Systems Administration (LISA) Conference, 2003
- [9] I. Foster, C. Kesselman, J. Nick, S. Tuecke., "Grid Services for Distributed System Integration". Computer, 35(6), 2002.
- [10] I. Foster, C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", International Journal of Supercomputer Applications, 11(2):115-128, 1997.
- [11] "The Apache Ant Project", <http://ant.apache.org/>
- [12] W. Allcock, "GridFTP Protocol Specification", Global Grid Forum Recommendation GFD.20. 2003.
- [13] D. Kurzyniec, T. Wrzosek, D. Drzewiecki, and V. Sunderam, "Towards self-organizing distributed computing frameworks: The H2O approach", Parallel Processing Letters, 13(2), 273-290, 2003.
- [14] M. Migliardi, V. Sunderam, A. Geist, and J. Dongarra, "Dynamic reconfiguration and virtual machine management in the Harness metacomputing system", Lecture Notes in Computer Science, volume 1505, Springer Verlag, 1998.
- [15] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, et al, "Resource Management Architecture for Metacomputing Systems". IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998.
- [16] K. Czajkowski, I. Foster, J. Frey, et al. "The WS-Resource Framework, Version 1.0", 2004, <http://www.globus.org/wsrf/>
- [17] "GT3 GRAM Architecture", <http://www-unix.globus.org/developer/gram-architecture.html>
- [18] V. Welch, F. Siebenlist, I. Foster, et al, "Security for Grid Services", HPDC-12, IEEE Press
- [19] G. von Laszewski, B. Alunkal, K. Amin, S Hampton, and S. Nijssure, GridAnt-Client-side Workflow Management with Ant. Whitepaper, 2002, <http://www-unix.globus.org/cog/projects/gridant/>