Abstract

This paper presents a unique collaboration between two grid middleware systems: the Minimum Intrusion Grid (MiG) and WebCom. Although both systems implement grid middleware properties, and provide a fully functional grid system, they are fundamentally different by design and implementation, and target different end-users.

A strategy that is mutually beneficial is obtained by supporting MIG and WebCom interactions that on the one hand provides WebCom users access to special sandboxed environments on a computing platform composed of the resource richness of the PRC model. On the other hand, MIG users can exploit the ease of use of the Visual programming model offered by the dynamic workflow execution environments offered by WebCom.

Keywords: MiG, WebCom,

1 Introduction

When first introduced, one of the big selling points of Grid Computing [1], was to make distributed computer resources as easily accessible as electricity in a power grid. Today, more than a decade later, the prohibitive barrier between grid application developers and efficient exploitation of the huge pool of aggregated resources still exists; developers still face several challenges when attempting to interact and utilize the Grid and a deeper insight to the underlying technologies is required. Providing uniform access to distributed heterogeneous resources to end users in the same manner that we get electricity from a socket in the wall remains an illusion.

Many Grid solutions exist, most of them are built on, or direct descendants of the Globus toolkit [2]. These systems have several shortcomings [3] that would greatly complicate a solution similar to the methodology presented here, most importantly

their lack of strong scheduling, poor scalability, firewall dependencies on connected resources and client, and their big resource and client software packages.

Gaining widespread acceptance as an important computing platform depends on making the technology accessible to general exploitation, not only for grid developers but also for non specialists. In practice, hiding the underlying complexities of a Grid system not only requires grid applications to be freed from all architectural details of the computational resources connected to the Grid, but also the set of tools needed to develop the applications and interact with the Grid system.

The Minimum intrusion Grid (MiG) [3] and WebCom [4] are two separate and independent middleware implementations, developed at different universities for different target groups. Both systems implement grid middleware properties, and provide a fully functional grid system, yet they are fundamentally different by design and implementation. Despite their differences, both middleware solutions were designed for user transparency with the vision of the ability to facilitate a grid operating system that would fully leverage the grid potential. This report presents a collaborative effort between these two grid middleware systems.

MiG strives for lowering the entry cost for users and resource providers by moving as much functionality as possible into a fat, centralized black-box grid system. MiG leverages the concept of 'sandboxes' to provide a non-intrusive and highly dynamic Grid computing infrastructure. Once a sandbox job is submitted to MiG, a sandbox is instantiated on a resource providers hardware. This sandbox then fetches and executes the application. Once this sandbox has completed its task, results are returned to MiG, and the sandbox is subsequently terminated. Thus every application in such a sandbox is guaranteed to execute in a fresh environment. This facilitates a great degree of non-intrusiveness on host resources and has proved to close the gap between Grid Computing and Public Resource Computing (PRC) models.

WebCom focuses on providing a suite of tools encompassing application development through to execution. Applications are expressed as workflows constructed from tasks of varying complexities such as grain-size, target execution environments and resource requirements. These applications are constructed from a 'palette' of services and submitted to a WebCom instance for execution. This mechanism allows for users with little knowledge of programming to create and execute applications of varying complexity on a vast range of distributed computing architectures from Networks of Workstations, to Clusters, to Grid computing infrastructures.

Whilst it is clear that both systems target different audiences and application execution models, a strategy that is mutually beneficial is obtained by supporting MiG and WebCom interactions. Both middlewares are highly capable of providing specialised features unique to their own domain. However, by utilising these features, we present a methodology that on the one hand provides WebCom users access to special sandboxed environments on a computing platform composed of the resource richness of the PRC model, and on the other hand, MiG users can exploit the ease of use of the Visual programming model offered by WebCom by embedding their applications within a dynamic workflow execution environment. This also eases utilization of the

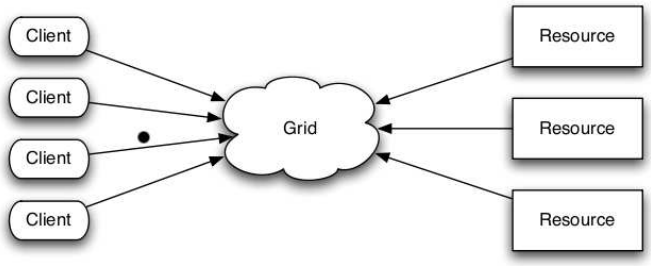

Figure 1: The abstract MiG model.

MiG sandboxes significantly thus effectively expanding the user group of the MiG system.

The following section gives a deeper introduction to each of the two middleware systems. Next, Section 3 describes in technical details the implementation of the presented approach. A motivating test application is presented in Section 4, and the report is closed by conclusions and future work in Section 5.

2 Grid Systems Introduction

2.1 Minimum intrusion Grid

MiG is a stand-alone approach to Grid that does not depend on any existing systems, i.e. it is a completely new platform for Grid computing. The philosophy behind the MiG is to provide a Grid infrastructure that imposes as few requirements on users and resources as possible. The overall goal is to ensure that users only need a signed X.509 certificate, trusted by Grid, and a web browser that supports HTTP and HTTPS. A fully functional resource only needs to create an MiG user on the system and to support inbound ssh and outbound HTTPS.

By keeping the Grid system disjoint from both users and resources, as shown in Figure 1, this model allows the Grid system to appear as a centralized black-box to both users and resources, and all upgrades and trouble shooting can be performed locally within the Grid without intervention from neither users nor resource administrators. Thus, all functionality is placed in a physical Grid system.

The basic functionality in MiG starts with users submitting jobs to MiG and resources sending requests for jobs. A resource then receives an appropriate job from MiG, executes the job, and sends the result to MiG that can inform the user of the job completion. Thus, MiG provides full anonymity; users and resources interact only with MiG, never with each other.

2.2 MiG Sandboxes

Typically, the resource farm of a grid system is comprised of unix/linux computers from many different compute centres. While the computational power provided by these may be significant, it is in no way comparable to the increasingly popular Public Resource Computing (PRC) models, such as BOINC [6], a middleware for volunteer computing[1]. Disguised as a screen saver, the PRC client software harnesses the idle time CPU cycles from millions of privately owned Internet-connected PCs across the globe.

However, there is a huge gap between such systems and Grid Computing in that the entry cost in terms of workload for application developers is far from the vision of a grid operating system[2]. In order to close this gap, MiG introduced sandboxes capable of tapping the enormous amount of unused processing power from standard desktop computers. First and foremost, minimizing the workload required by resource owners for installing and managing a Grid resource, and ensuring the integrity of the donated resource is utterly important. Therefore, all grid application are sandboxed in a virtualized environment.

Due to the renewed popularity of virtualization over the last few years, virtual machines are being developed for numerous purposes and therefore exist in many designs, each of them in many variants with individual characteristics. Despite the variety of designs, the underlying technology encompasses a number of properties beneficial for Grid Computing [7]:

Platform Independence Since moving application code around as freely as application data is an intrinsic part of a grid system, it is highly profitable to enable applications to be executed anywhere in the grid. Virtual machines bridge the architectural boundaries of computational elements in a grid by raising the level of abstraction of a computer system, thus providing a uniform way for applications to interact with the system. Given a common virtual workspace environment, grid users are provided with a compile-once-run-anywhere solution.

Furthermore, a running virtual machine is not tied to a specific physical resource; it can be suspended, migrated to another resource and resumed from where it was suspended.

Host Security To fully leverage the computational power of a grid platform, security is just as important as application portability. Today, most grid systems enforce security by means of user and resource authentication, a secure communication channel between them, and authorization in various forms. However, once access and authorization is granted, securing the host system from the application is left to the

[1] In 2006, the BOINC projects reached 400 TFlop/s, while BlueGene/L topped the top 500 list of supercomputers with 280 TFlop/s

[2] For instance, according to the BOINC website, it takes three man-months to port an existing application to the BOINC framework and about $5000 for hardware used for project maintenance

operating system.

Ideally, rather than handling the problems after system damage has occurred, harmful - intentional or not - grid applications should not be able to compromise a grid resource in the first place.

Virtual machines provide stronger security mechanisms than conventional operating systems, in that a malicious process running in an instance of a virtual machine is only capable of destroying the environment in which it runs, i.e. the virtual machine.

Application Security Conversely to disallowing host system damage, other processes, local or running in other virtualized environments, should not be able to compromise the integrity of the processes in the virtual machine.

System resources, for instance the CPU and memory, of a virtual machine are always mapped to underlying physical resources by the virtualization software. The real resources are then multiplexed between any number of virtualized systems, giving the impression to each of the systems that they have exclusive access to a dedicated physical resource. Thus, grid jobs running in a virtual machine are isolated from other grid jobs running simultaneously in other virtual machines on the same host as well as possible local users of the resources.

Resource Management and Control Virtual machines enable increased flexibility for resource management and control in terms of resource usage and site administration. First of all, the middleware code necessary for interacting with the Grid can be incorporated in the virtual machine, thus relieving the resource owner from installing and managing the grid software. Secondly, usage of physical resources like memory, disk, and CPU usage of a process is easily controlled with a virtual machine.

Performance As a virtual machine architecture interposes a software layer between the traditional hardware and software layers, in which a possibly different instruction set is implemented and translated to the underlying native instruction set, performance is typically lost during the translation phase. Despite of recent advances in new virtualization and translation techniques, and the introduction of hardware-assisted capabilities, virtual machines usually introduce performance overhead and the goal remains achieving near-native performance only. The impact depends on system characteristics and the applications intended to run in the machine.

To summarize, virtual machines are an appealing technology for Grid Computing because they solve the conflict between the grid users at the one end of the system and resource providers at the other end. Grid users want exclusive access to as many resources as possible, as much control as possible, secure execution of their applications, and they want to use certain software and hardware setups.

At the other end, introducing virtual machines on resources enables resource owners to service several users at once, to isolate each application execution from other users of the system and from the host system itself, to provide a uniform execution

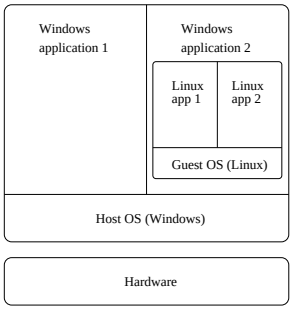

Figure 2: Full virtualization; applications running in the guest OS in the virtual machine are isolated and can only compromise the VM, not the host system

environment, and managed code is easily incorporated in the virtual machine.

2.3 The MiG Sandboxes

Taking advantage of these virtualization properties, the MiG sandbox solution effectively combines Grid Computing and PRC computing. Contributors download a screen saver, a virtual machine of their own choice, and a custom made MiG Linux image. Upon screen saver activation, the virtual machine boots the MiG Linux image in which all grid jobs will be running. As soon as the screen saver deactivates, the virtual machine is shut down, and all resources are given back to the local user.

As shown in Figure 2, the virtual machine is just a normal process running in the host OS, which typically would be Windows. The virtual machine emulates the underlying physical hardware, thus creating a secure sandbox environment that allows an application written for one OS to be executed in another.

While the MiG sandboxes are quite similar to the 'normal' native resources in the MiG, a few things differ. Most importantly, since most typical desktop computers reside behind NAT routers it is not possible to achieve the MiG requirement of inbound ssh access on resourcers. Therefore, to stay in the philosophy of minimum intrusion, sandboxes use a strict pull-based model where all communication is initiated from the resource. For further details, see [5]. For MiG users who wish to deploy their applications on the virtualized sandbox platform, all that is necessary is to specify the 'sandbox' keyword in the job description file.

2.4 WebCom

Problem solving for parallel systems traditionally lay in the realm of message passing systems such as PVM and MPI on networks of distributed machines, or in the use of specialised variants of programming languages like Fortran and C on distributed shared memory supercomputers. The WebCom System[8] relates more closely to message passing systems, although it is much more powerful. Message passing ar-

chitectures normally involve the deployment of a codebase on client machines, and employ a master or server to transmit or *push* "messages" to these clients. WebCom supports this level of communication, but is unique in bootstrapping its codebase by *pulling* it from the server as required.

Technologies such as PVM, MPI and other metacomputing systems place the onus on the developer to implement complete parallel solutions. Such solutions require a vast knowledge on the programmer's part in understanding the problem to be solved, decomposing it into its parallel and sequential constituents, choosing and becoming proficient in a suitable implementation platform, and finally implementing necessary fault tolerance and load balancing/scheduling strategies to successfully complete the parallel application. Even relatively trivial problems tend to give rise to monolithic solutions requiring the process to be repeated for each problem to be solved.

WebCom removes much of these traditional considerations from the application developer; allowing solutions to be developed independently of the physical constraints of the underlying hardware. It achieves this by employing a two level architecture: the computing platform and the development environment. The computing platform is implemented as an Abstract Machine (AM), capable of executing applications expressed as Condensed Graphs. Expressing applications as Condensed Graphs greatly simplifies the design and construction of solutions to parallel problems. The Abstract Machine executes tasks on behalf of the server and returns results over dedicated sockets. The computing platform is responsible for managing the network connections, uncovering and scheduling tasks, maintaining a balanced load across the system and handling faults gracefully. Applications developed with the development environment are executed by the abstract machine. The development environment used is specific for Condensed Graphs. Instructions are typically composed of both sequential programs (also called atomic instructions) and Condensed nodes encapsulating graphs of interacting sequential programs. In effect, a Condensed Graph on WebCom represents a hierarchical job control and specification language. The same Condensed Graphs programs execute without change on a range of implementation platforms from silicon based Field Programmable Gate Arrays[9] to the WebCom metacomputer and the Grid.

2.4.1 Architecture Overview

The WebCom abstract machine is constructed from a set of modules that plug into a core module called the *backplane*. Each module in the WebCom system falls into one of two categories: it is either a *Core* module or a *User* module. Modules are loaded based on a initial configuration, thus bootstrapping the computational platform. Core modules cover the *Compute Engine*, *Fault Tolerance*, Scheduling and Load balancing via the *Distributor*, *Security*, *Communications*, *Job Management* and *Information Management*. User modules can be provided to add additional functionality to the WebCom abstract machine. This architecture is outlined in Fig. 3.

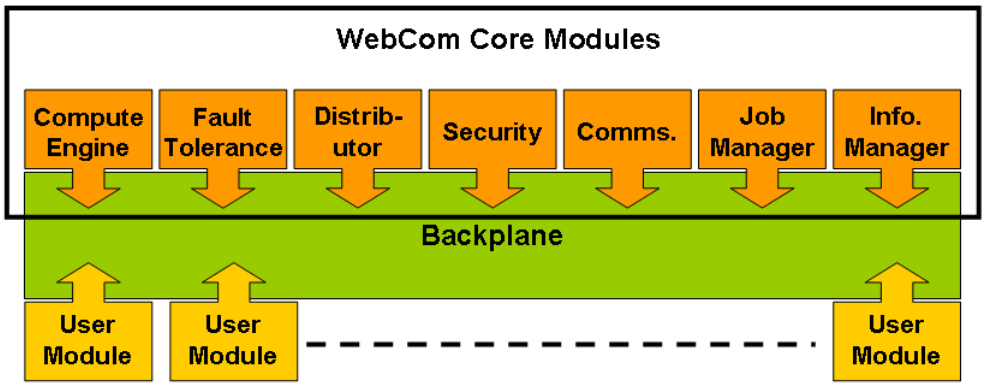

Figure 3: WebCom Abstract Machine Architecture showing plugin architecture.

Compute Engine: The compute engine is responsible for low level task execution. Task composition varies depending on the compute engine in use. The default Condensed Graphs compute engine is responsible for executing applications expressed as Condensed Graphs.

Fault Tolerance: The fault tolerance module detects and corrects faults that occur within the Abstract Machine. Mechanisms in place for fault tolerance range from simply rescheduling failed tasks to employing a unique processor replacement strategy [10].

Distributor: The distributor is responsible for allocating work to clients. The distributor operates according to a set of installed policies. Typically, a system-wide default policy exists that allows the distributor to select clients for task execution. These policies are plugable and hierarchal in nature and specify items such as the selection algorithm and associated configuration parameters. Nodes within a Condensed Graphs application are executed according to the installed policy. In addition, nodes themselves can specify their own distribution policy. Such nodes will be allocated to clients based on that policy. This provides great flexibility within the AM, as an application is not tied to one particular scheduling algorithm. Different nodes in a single application can be scheduled using different algorithms. So, it is possible that multiple applications executing on a single abstract machine can execute using multiple selection algorithms within the distributor.

Security: The security manager is responsible for authenticating the tasks, results and other messages transmitted throughout the WebCom infrastructure. The current security manager is based on the *Keynote*[11] standard.

Communications Manager Module: The communications manager is responsible for communicating tasks to clients. Once the distributor has decided where to allocate a task, it is placed in a queue for the selected client. The communication manager module serves this queue, transmitting the task to the client. Tasks are sent based upon a *Pull Request* mechanism. When a client is willing to accept work, it issues a pull request. A WebCom will respond to this request by *pushing* the task to the client.

Job Manager: Each application within the WebCom AM executes within its own job space. The job manager [12] can be used to monitor the progress of job execution, pause and restart jobs and also suspend jobs.

2.5 Combined Potential

By utilising the strengths both the MiG and WebCom platforms, we have identified a number of areas where this collaboration can provide significant advances for both user communities. We will elaborate on some of these advantages in the remainder of this section.

For the MiG user community: MiG users submit through a script. Once the job enters the MiG system, the user has to periodically poll it to enquire about the jobs status. In most cases, the user will have a rough idea as to how often they need to issue this poll, however, it is possible that they may not issue this poll in a timely manner. Using WebCom as a submission engine for MiG, the user will be notified when the job has been completed. This is determined by WebCom issuing the poll for the job status at regular intervals. At worst, the maximum time the user will be waiting for such a notification would be equivalent to the polling interval the MiG Compute Engine employs.

A second potential benefit to the MiG user community is that this system provides a platform whereby users can construct scientific workflows using an intuitive interface. Just drag and drop the required nodes, connect the arcs and execute the resulting application. As the work presented here describes our initial investigations, it is apparent that a more complete suite of nodes would be required. However, this is easily achievable.

A third benefit to the MiG user community could be to support the PRC model of computing at the resource level. Resource donators can choose which projects their particular sandbox will support. With an ever increasing number of projects being supported by MiG, WebCom's targeting mechanism can be used to provide a more flexible approach to application execution, targeting applications to particular sandboxes. This would happen transparently to the user.

For the WebCom user community: The sandbox model may be of particular interest to users of WebCom. Creating jobs in a sandboxed environment allows for

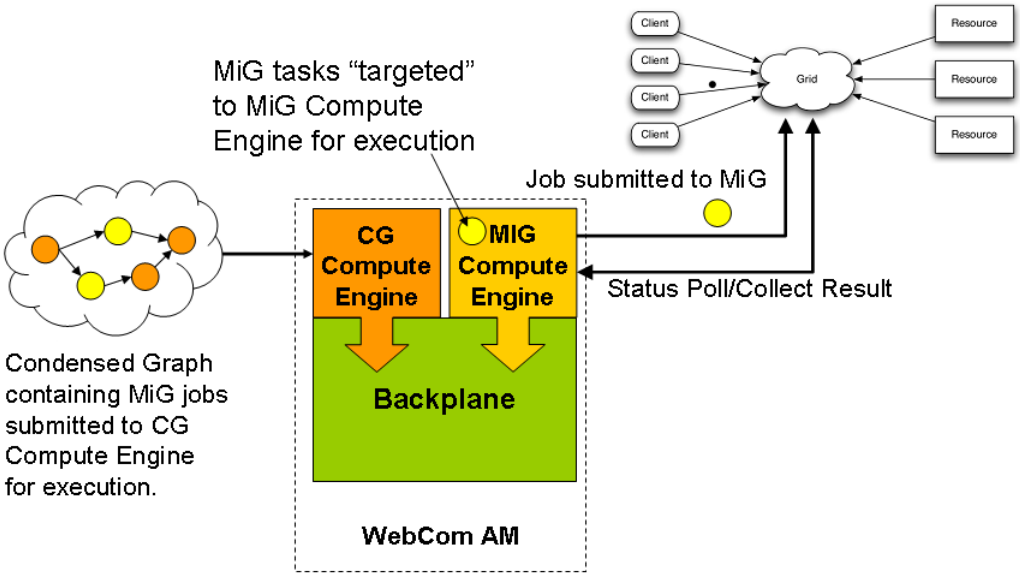

Figure 4: High Level computation showing MiG compute engine

greater scope in deployment of WebCom. WebCom itself could be pre-packaged with a particular configuration. MiG could be used to deploy and instantiate any number of these packages. When executed, a dynamic WebCom platform would be created, allowing users to execute applications within a fresh, secure environment. This has potential for creating multiple WebCom instances whose lifetime may only exist for the duration of a single application.

WebCom's targeting can be use to include MiG jogs in a higher level workflow. Scientific computations are typically better suited to native execution. Native MiG computations could be easily invoked to leverage the underlying resources.

3 Implementation

At the basis of supporting MiG interaction from WebCom lies the client side of the MiG system shown on the left in Figure 1. When incorporating a MiG client in another system, the 'minimum intrusion' property of MiG is highly valuable, as the only requirements to the client is a valid X.509 certificate trusted by MiG and outbound HTTP and HTTPS. There are several ways for users to interact with the MiG system: Either through a standard web browser or by several types of scripts. A job is defined by a minimal Resource Specification Language(mRSL) text file. The user holding the certificate automatically gets a home directory and must upload all grid-related files to this home directory. Then, all file names mentioned in the job description are relative to the user's home directory, just as if it were local.

Initial support for MiG interpretability is provided at two levels: the first being the

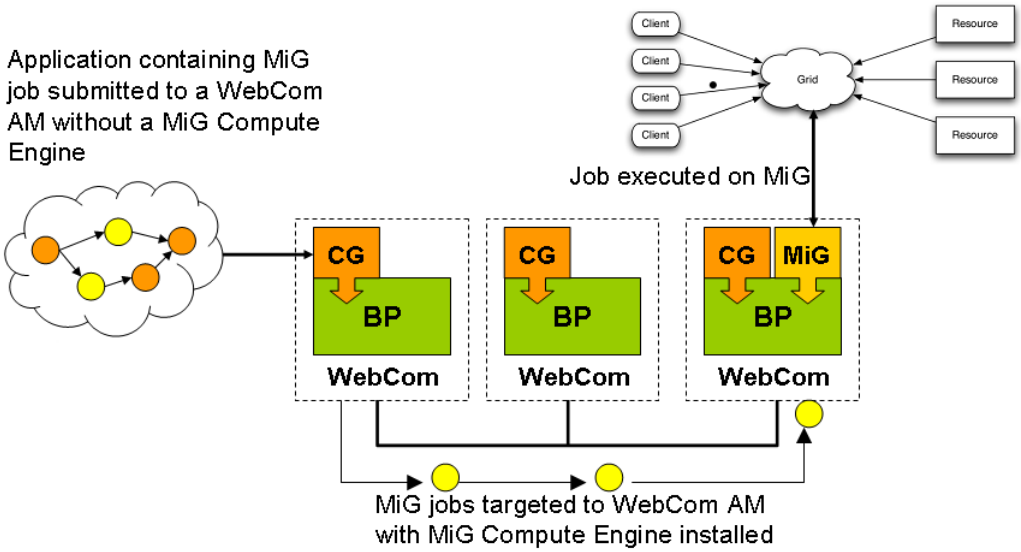

Figure 5: Job Targeting, target MiG jobs.

provision of a new Engine Module called the MigEngine and the second the provision of a suite of nodes that can be included in the WorkFlow being developed.At the simplest level a node that executes a MiG job description file is provided. When the MiG compute engine receives an appropriate instruction, it carries out the operations required for that instruction.

The instruction to execute a job submission script file goes through a number of stages: First the appropriate mRSL file is saved to some predetermined location, next the MiG compute engine invokes the MiG platforms submission script. The result of this submission is a job id. This id can be used to obtain information about the jobs progression such as its current status and what result was received. Typically, within the MiG system, these operations are conducted periodically by the user.

WebCom's MiG Compute Engine can be used to remove this task from the user. When the compute engine submits the task for execution it then proceeds to poll the mig system at regular intervals to obtain the status of a job.

In the initial version of the MiG Compute engine, once a job finishes it returns the jobs status to the user. At this stage the user can manually retrieve any output files, error files generated by the job.

On an extended WebCom network, Fig. 5, it is possible for a user to construct workflow applications containing MiG jobs without having to install or join the MiG platform. When such a workflow is submitted to a WebCom not capable of executing it directly (that WebCom does not have a MiG Compute Engine installed locally), it advertises the need for another WebCom that has a a MiG Compute Engine plugged in.

Other WebCom instances on the network that match the requested criteria will re-

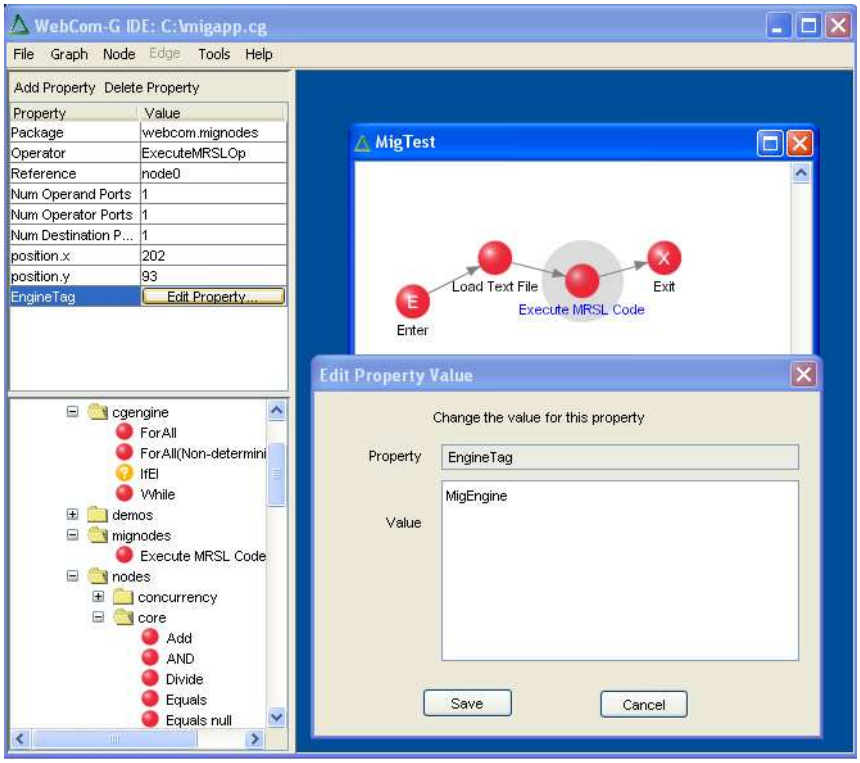

Figure 6: Simple Application

spond. When a response is received the task will subsequently scheduled to a responder and WebComs targeting mechanism is then invoked to ensure the task is delivered to the selected responder. In the case where there are multiple responders to a request, the first responder is the one that will be selected. In the case where there are no responders (no WebCom has a MiG Compute Engine plugged in), the task is put into a waiting queue. Periodically, tasks in this queue are passed to the distributor for re-allocation. This results in a further request for a WebCom with a MiG compute engine. Tasks will continue to wait, possibly indefinitely, until a suitable responder is found.

4 Test Application

To illustrate the process of executing a MiG application, a simple application was created. This is shown in Fig. 6. The test application simply loads a mRSL file and feeds that as input to the `Execute MRSL Code` task. The final result is then passed back to the user. Typically nodes in a condensed graph will be targeted for execution on a Condensed Graphs compute engine. However, the `Execute MRSL Code` node shown in the figure must be targeted for execution on a MiG Compute Engine. This targeting information can be seen in the *Edit Property Value* dialog that is visible. This

is necessary, because as described earlier it is the MiG Compute Engine that posses the knowledge as how to execute this type of node. When the `Execute MRSL Code` node finishes the status is passed on to the `X` node.

The test configuration for this application included one computer that acted as a MiG gateway: This computer had the MiG submission system and a WebCom instance installed. The WebCom system installed on this machine had both the Condensed Graphs and MiG execution engine modules installed.

A second machine with WebCom was used for constructing, and executing the application. Once the application was submitted, execution proceeded as expected: The specific MiG nodes were targeted to the WebCom instance with the MiG compute engine installed.

5 Conclusions & Future Work

This paper presented details of our initial investigation into supporting interactions between two different computing systems: The Minimum Intrusion Grid (MiG) and WebCom. Initial investigations concentrated on providing added functionality to the WebCom system that facilitate the targeting and execution of MiG jobs. Applications consisting of MiG jobs are created within the WebCom IDE and executed on the WebCom computational platform. The initial implementation proved successful although there is significant scope for further research.

Further research can be conducted in to the number of potential benefits to both systems user communities: such as more fine grained application targeting for MiG users, deploying and establishing dynamic WebCom compute environments, investigating the use of more scientific computations such as Fast Fourier Transformations for example, extending the palette of nodes available to the users when creating the workflows.

In addition to the items outlined above, further research and gathering of results generated by MiG can be conducted. The current prototype implementation just returns the status code to the user, along with the job id. Currently, the user has to manually retrieve the results.

Additional nodes can be provided for the complete suite of MiG commands, each command would then have a direct correspondent when creating visual applications.

Acknowledgements

The work presented here was part funded by the Boole Centre for Research in Informatics, Science Foundation Ireland under the WebCom-G project and the Danish NABIIT program committee.

References

[1] I. Foster, C. Kesselman: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, Elsevier Press (2004) 593-620

[2] I. Foster, C. Kesselman:The Globus project: a status report. Proceedings of the Seventh Heterogeneous Computing Workshop, March 1998, IEEE Computer Society Press, pp. 4–19

[3] B. Vinter: The Architecture of the Minimum intrusion Grid: MiG. Proceedings of Communicating Process Architectures 2005

[4] J.P. Morrison, B. Clayton, D. Power, A. Patil: WebCom-G: Grid Enabled Metacomputing- The Journal of Neural, Parallel and Scientific Computation. Special Issue on Grid Computing. Editors H.R. Arabnia, G.A. Gravvanis, M.P. Bekakos, Vol. 12(3), PP. 419-438, September, 2004

[5] R. Andersen, B. Vinter: Harvesting Idle Windows CPU Cycles for Grid Computing. Proceedings of GCA-2006

[6] D.P. Anderson: BOINC: a system for public-resource computing and storage. Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)

[7] R. Figueiredo, P. Dinda, J. Fortes: A Case for Grid Computing on Virtual Machines. Proceedings of the International Conference on Distributed Computing Systems (ICDCS), May 2003

[8] J.P. Morrison, D.A. Power, J.J. Kennedy: An Evolution of the WebCom Metacomputer. The Journal of Mathematical Modelling and Algorithms: Special issue on Computational Science and Applications, 2003(2), pp 263-276

[9] J.P. Morrison, P.J. O'Dowd. P.D. Healy: Searching RC5 Keyspaces with Distributed Reconfigurable Hardware. Proceedings of ERSA 2003, Las Vegas, June 23-26, 2003

[10] J.J. Kennedy: Design and Implementation N-Tier Metacomputer with Decentralised Fault Toerance. PhD Thesis, University College Cork, Ireland, May 2004

[11] M. Blaze et al: The Keynote Trust Management System, Version 2. Internet Request for Comments 2704. September 1999

[12] N. Cafferkey, P.D. Healy, D.A. Power, J.P. Morrison: Job Management in WebCom. In Proceedings of the 6th International Symposium on Parallel and Distributed Computing (ISPDC) - IEEE Press, Hagenberg, Austria, July 4th-8th, 2007