Multi-disciplinary Design Optimization (MDO) application is being looked at as a case study for the GMIS framework. This MDO application requires integration of disciplinary analysis modules. These disciplinary analyses are typically distributed over Internet. Integrating such analysis modules is usually quite complex and addressed in UH MDO framework [Saran 2005, Nilakanta 2006]. This framework would require a platform which provides distributed computing environment over a WAN. Globus was used as a distributed environment for the framework. However GMIS can be used as a distributed environment for the UH MDO framework. GMIS provides all services required for UH MDO framework.

This chapter gives an introduction to Multidisciplinary Design Optimization, MDO requirements and then discusses about some of the frameworks available. It discusses about the UH MDO framework architecture and how GMIS components support UH MDO framework functionality. It shows how activation graphs can be created in GMIS viewer with an example. Finally another application called “Virtual Cluster” formation using GMIS is discussed.

6.1 Introduction to Multidisciplinary Design Optimization (MDO)

Multi-Disciplinary Design Optimization (MDO) is gaining popularity and is increasingly being used to design complex engineering systems. This requires integration of disciplinary analysis modules. Disciplinary analysis typically evolves without any centralized coordination and is often distributed over network.

System level analysis can be performed using MDO framework. Such a system level analysis includes each sub-system analysis and various sub-system interactions. For the purpose of performing system analysis various analysis modules are to execute in some sequence. For large number of modules it becomes humanly difficult to iterate, the generated input files and parsing output files to extract useful information. Therefore
a software integration system which helps user to tie in all disciplinary analysis modules is required.

The UH MDO framework [Saran 2005, Nilkanta 2006] supports process execution in distributed heterogeneous computers. Globus was used as a distributed environment for the framework when it is developed. The framework provides intuitive GUI and is extensible. It handles large size problems, by dividing it into small modules, automates the execution and movement of data between different modules over grid. The framework provides the database management features for storing and provides data exchange between analysis modules.

Multi-disciplinary analysis process contains:

1. Build the data dictionary.
2. Collect all the analysis modules
   i. Build wrappers for all analysis modules
   ii. Register in the framework
3. Design the system problem
   i. Specify data dependency
   ii. Providing execution sequence
4. Specify optimization problem
5. Solve the problem
6. Visualize results

6.2 MDO Requirements

The previous MDO framework designs are based on distributed architectures like Common Object Request Broker Architecture (CORBA). CORBA based programs are interoperable irrespective of the type of computer, network, operating system or programming language. In addition to remote execution, CORBA provides services for naming, transaction monitoring, error handling, etc. This makes CORBA ideally suited for distributed computing. Most of the frameworks that are developed so far and are being developed are using CORBA for network communication.
However, all these activities are concentrated to make frameworks work on Local Area Networks (LANs). To make these activities over a Wide Area Network (WAN) require a distributed architecture that is suitable for the requirement. Grid computing is a technology where distributed computing over WAN is possible. So grid computing has used to build a MDO framework called UH MDO framework, so that it will be suitable for computing over WAN.

The purpose of UH MDO framework is to provide support for multidisciplinary design optimization application development and execution. This section lists a set of requirements for an ideal framework. The requirements are presented from the following points of view:

- Architectural design
- Problem formulation and construction
- Problem execution
- Information access

**Architectural Design:**

- Provide a intuitive GUI
- Adapt object-oriented principles
- Extensible and support for developing interfaces for adding new programs
- No unreasonable amount of overheads
- Handle large size of problems
- Support collaborative design

**Problem Formulation and Construction:**

- Configure complex branching and iterative problem formulations easily.
- Easily reconfigure existing problems
- Support multiple optimization methods including multilevel schemes.
- Provide debugging support for multiple processes executing across computers on the network.
Problem Execution:

- Automate execution of processes and movement of data
- Execute multiple processes in parallel
- Support execution distributed across network of heterogeneous computers
- User interaction (steering) during design cycle
- Operate in batch mode

Information Access:

- Provide database management features
- Capability to visualize intermediate and final results of the analysis
- Monitoring capability for viewing the status of an execution and system status
- Mechanism for fault tolerance

6.3 Related work

6.3.1 Frameworks at Research Labs

There are various MDO frameworks developed at research labs. Some of the important frameworks are discussed in the following subsections.

6.3.1.1 FIDO

FIDO stands for Framework for Interdisciplinary Design Optimization was developed by NASA Langley Research center [Townsend, 1993]

Architectural Design: The FIDO architecture is modular. The framework is organized into distributed computational and service modules, which communicate through a communications library. There is a computational module for each discipline contributing to the application. The service modules, such as the GUI, Executive (control), Data Manager, Setup, and Spy, are intended to be application independent.
The communications library contains functions designed to facilitate communications among a general system of computer codes executed in a heterogeneous, distributed network of computers. This library allows FIDO to be programmed without directly accessing the underlying, message-passing primitives and minimizes the impact on FIDO due to any changes in them. Currently, the PVM (Parallel Virtual Machine) primitives from the Oak Ridge National Laboratory are used.

The GUI is limited to displaying the status of the execution. The Spy tool promotes collaboration among researchers by allowing access to Spy from multiple remote computers. Although object-oriented principles were not applied to FIDO, there was a strong emphasis on producing a modular system.

**Problem Formulation and Construction:** A major limitation of FIDO is that it lacks support for building and reconfiguring MDO problem formulations at a higher level of abstraction than coding in the currently available programming languages, such as FORTRAN and C. Some discipline codes used in FIDO are decades old and originally contained deeply embedded `print` and `stop` statements. These codes were modified to behave as library subroutines and are invoked from the appropriate discipline driver. As a result, the discipline driver and the associated discipline codes are linked into one executable program. Overall, this is not a desirable approach because it involves extra work, duplicates maintenance tasks, and does not promote code reuse.

Coordination of discipline analyses is provided by a problem-dependent Master module. The code for this module must be rewritten for each specific MDO application.

**Problem Execution:** The user designs the FIDO ‘master’ module so that the optimization and analysis processes are invoked and synchronized appropriately. The user provides the synchronization logic within the discipline drivers in the form of calls to the communications libraries send and receive routines.

The FIDO ‘setup’ module allows the user to choose the system configuration, initial conditions and constraints of the optimization process from a range of previously defined possibilities. These are contained in configuration files that define the data in standardized formats.
All major data elements (individual items or file pointers) that are shared between modules are passed to, stored in, and retrieved from the central Data Manager. Using file pointers, data files are passed directly on request from the generating computer to the requesting computer within a LAN. The discipline drivers and their corresponding analyses are assigned to execute in parallel on different computers defined to be part of the PVM network.

From the beginning of its development, FIDO was designed to allow some interactivity during the design cycle. This feature is accomplished using the Spy tool, which allows the user to steer the process while the application is executing. By means of the Spy tool, the user may change current values of design variables, constraints, and parameters. On the other hand, FIDO lacks a convenient way of setting up multiple problems that can execute one after the other. In particular, changing the initial conditions of a problem requires manually editing the input and configuration files.

**Information Access:** The FIDO Data Manager allows storage and retrieval of data during problem execution, so that no additional coding is required for new problems. The user must define the data to be handled prior to execution.

The FIDO Spy module allows the user to access and plot data from previous design cycles. The accessible data includes information on the cycle status and selected scalar and array data from each cycle. The data can be displayed as text or graphics. However, the database is not persistent, so FIDO must be running for data to be accessed.

The FIDO GUI displays the state of the problem execution at all times. The GUI displays the problem formulation and uses color to indicate those processes that are starting up, executing, inactive, or shutting down. Although FIDO allows restart from a completed optimization cycle, it provides no other fault-tolerance capability. However, a restart requires some data file preparation.

### 6.3.1.2 DAKOTA

DAKOTA which stands for Design Analysis Kit for OpTimizAtion, was developed by Sandia National Laboratories [DAKOTA]. The DAKOTA design is based on object-
oriented principles and is implemented with the C++ language. The definition of generic interfaces between optimization methods and analysis codes hides the specifics of each. Use of these interfaces and object oriented language features promotes the "plug and play" capability.

**Problem Formulation and Construction:** To define the MDO application, the user must create a file that specifies information about interfaces, variables, responses, strategies, and methods. In DAKOTA, "strategies" manage methods and "interfaces" provide access to the discipline code, which map the variables to the responds.

Several types of interfaces are defined in DAKOTA, the primary being the application interface. The application interface allows discipline codes to be accessed through either system calls or direct function calls. The direct function call interface requires converting main programs to function calls and linking the functions into the DAKOTA executable. The system call interface allows access to external programs; communication between the external program and DAKOTA is accomplished via files.

The interface section of the specification file must include the name of the analysis (or analysis driver), and if required, the names for the input and output filters (i.e., pre and post processors). Note that only one analysis driver may be specified; however, an entire MDO application, developed outside of DAKOTA, along with an input and an output filter. The input filter must use the design parameters list provided by DAKOTA to prepare the input for the analysis driver. Also, the output filter must retrieve data from the analysis driver and prepare the response and sensitivity data in the format required for use by DAKOTA.

A variety of optimization methods are provided, including Non-Linear Programming (NLP) and genetic algorithms. The DAKOTA strategies manage multiple methods, disciplines, and approximations. The strategies include single, multilevel hybrid, and sequential approximate optimization. The single strategy allows a single method to be used with a single discipline. The multilevel hybrid strategy allows multiple methods to be used in succession with a discipline. This strategy uses the best solution from one method as the starting point for the next method. The switching criteria used can either be based on an individual method's convergence criteria or an adaptive...
technique that employs method performance metrics. The sequential approximate optimization strategy uses both a discipline and an approximation of the discipline. The approximation model is optimized, and the discipline model is evaluated at the approximate optimal solution. These results are used to update the approximation.

**Problem Execution:** Both the execution of the analysis driver and input/output filters and the transfer of data between these and the optimization methods are automated by DAKOTA. Distributed computing is supported using Message Passing Interface (MPI) message passing on workstation clusters and on massively parallel supercomputers. The asynchronous function evaluation command option allows concurrent analysis calculations and is available with both system call and direct function interfaces. This feature can be used when calculating derivatives using finite differences or when using the parallel algorithms provided in DAKOTA.

**Information Access:** There is an option for the user to specify creation of a restart log. Also, several options are available for handling application failure recovery.

**6.3.1.3 CASDE**

CASDE is Center for Aerospace Design and Engineering department in IIT Bombay [Amitay, 2003]. They have developed a software integration system, which helps user tie in all the disciplinary analysis software, and automate the execution of those modules in the sequence provided by the user. The design of their framework is based on object oriented principles and was implemented in Python.

**Architectural Design:** The framework developed by CASDE contains the GUI, which provides features for constructing the analysis module sequence, design problem and for analyzing the results. The GUI is written using the wxPython language. The user can include any analysis code as part of an MDO application as long as the design input and output can be identified in the input/output files. Database engine used in this framework is MySQL.

The concept of Data Server is introduced which serves the purpose of exchanging data between different analysis modules in the distributed environment. Data can be specified as scalars, vectors, multidimensional arrays or complex
structures. CORBA (Common Object Request Broker Architecture) is used as the distributed computing feature in the framework.

**Problem Formulation and Construction:** The user employs the framework GUI to define the analysis sequence. Through the GUI, the user identifies the data flow and the execution sequence analyzes modules along with their corresponding input and output files. In addition, the files associated with the design data input and output are identified.

MDO framework uses multithreading approach to perform the parallel execution that is identified in the execution sequence. Corresponding to each analysis module, one thread of execution is started. Directed edges in the sequence diagram denote the events, which each thread waits on.

**Problem Execution:** The framework automates the execution of the various disciplinary modules included in the analysis, manages the input and output data, and adjusts the design variables. The processes defined in the analysis are executed sequentially. For distributed computing support, CORBA features are used which could provide a remote process invocation methods.

**Information Access:** The framework’s data server allows storage and retrieval of data during problem execution and is designed in such a way that no additional coding is required for new problems. The user must define the data to be handled prior to execution. The framework GUI is helpful in viewing the results of all the problems which were already executed and saved.

**6.3.2 Frameworks at Commercial Organizations**

There are some commercial organizations developed MDO frameworks. Some of such commercial frameworks are discussed in the following sub sections.

**6.3.2.1 iSIGHT**

The iSIGHT [iSIGHT, 1998] framework is a generic shell environment for supporting multidisciplinary optimization developed by Engineous software. A key feature of iSIGHT is the ability to combine numeric, exploratory, and heuristic methods during an optimization.
**Architectural Design:** The iSIGHT environment consists of several modules including an interpreter, toolkits, and GUIs. The Tcl language is the interpreter that provides the “glue” for integrating various processes. GUI services are provided for connecting processes, defining the optimization plan, and monitoring results. GUI services are provided for wrapping discipline codes. In addition, the user may integrate additional optimization techniques into iSIGHT. However, the iSIGHT Application Programming Interface (API) must be used to create the appropriate interface between the optimizer and the framework. In addition, a Tcl command must be created for the optimization technique.

**Problem Formulation and Construction:** The iSIGHT framework provides a GUI, in which icons represent modules, and the Multidisciplinary Optimization Language (MDOL) for constructing MDO problems. Use of the GUI to define the problem generates the appropriate MDOL file, referred to as a description file. MDOL has a block structure style and English like language constructs.

The GUI provides the user with building blocks representing discipline codes and calculation blocks that may be needed in addition to the discipline codes. The user may define the input, output, and execution invocation of the discipline code blocks, as well as the arithmetic expressions for the calculations blocks. However, within the GUI, the user is limited to define a sequential order for the disciplines and calculations.

The iSIGHT framework allows users to construct MDO applications using existing discipline codes without modifications by interactively generating a code wrapper. Parsing utilities create the appropriate input files and extract the appropriate data from discipline output files. Using the GUI and the input and output file templates for a discipline code, the user can generate the appropriate file parsing commands. As a result, the user is able to integrate legacy and proprietary codes into the MDO problem.

**Problem Execution:** The iSIGHT framework automates the execution of the various discipline codes and calculation blocks, the handling of the data, and the adjustment of design variables during optimization. The computational processes defined in the problem formulation are executed sequentially, because iSIGHT provides no support for parallelism. There is very limited support for distributed computation. For example, a
discipline code may initiate a remote process; however, all description codes for a problem must reside in the same directory on a single computer. Interactive features in iSIGHT provide the capability to pause the execution and continue it later. The user can stop the execution to modify the optimization methods, design variables, constraints, and objective function. Upon restart, the execution resumes from the best design point of the previous optimization. Discipline codes can be switched only if the appropriate logic is present in the description file.

**Information Access:** The iSIGHT framework lacks a database capability other than the data management toolkit that keeps a history of the design states. Therefore, data sharing among several discipline codes has to be accomplished by writing and parsing files. A monitoring capability is provided that can be applied at any time during execution. Input and output values can be monitored in tabular or graphical form. In addition, the user can review the data from a completed optimization and can restart the optimization process from a design point previously computed. The GUI also indicates which process in a task is running by changing the appearance of the module icon.

### 6.3.2.2 LMS Optimus

LMS Optimus [LMS, 1997] is a framework for multidisciplinary optimization, developed by LMS International Inc. LMS Optimus, allows a user to setup a problem, select a method to be used with the problem, and analyze the results. Features of LMS Optimus provide nonlinear programming (NLP), DOE and RSM techniques for optimization.

**Architectural Design:** The Optimus Kernel module contains the GUI, which provides features for constructing the analysis sequence and design problem and for analyzing the results. The GUI is written using the C++ language and Motif. The user can include any analysis code as part of an MDO application as long as the design input and output can be identified in the input/output files.

Two optimization modules are provided in LMS Optimus: the DOE/RSM module and the NLP module. A recently available feature allows the integration of an external optimizer. All that is required for integration of an optimizer is that it writes the adjusted design variables to a file and reads the analysis results from a file. Due to internal array
sizes, the maximum number of design variables allowed is 50; the maximum number of design outputs allowed is 200.

**Problem Formulation and Construction:** The user employs the LMS Optimus GUI to define the analysis sequence. Through the GUI, the user identifies the analyses and their corresponding input and output files. In addition, the files associated with the design data input and output are identified. Command file will be created based on the actions taken through the GUI. The command file contains sections for defining design inputs, design outputs, discipline input and output file parsing commands, analysis sequencing, and optimization method selection. The sequencing commands include if/then/else and for control statements. The GUI generates only a subset of commands that can be included in the command file; the user can edit the command file to include additional commands.

The LMS Optimus user can include legacy and proprietary codes in the analysis sequence without making any modifications. The GUI can be used to identify the design data in the input files; before each analysis is executed. The framework will include appropriate input automatically from the input files. Similarly, the user identifies, through the GUI, the output which needs to be extracted from the output file; after the analysis completes, the data is automatically extracted.

**Problem Execution:** The Optimus Kernel automates the execution of the various discipline codes included in the analysis, manages the input and output data, and adjusts the design variables. The processes defined in the analysis are executed sequentially. For distributed computing support, the command language includes a command for executing a remote process.

**Information Access:** The results from a completed NLP or DOE method can be loaded by the GUI and post processed. The results of an optimization can be displayed in a tabular format. Several options exist for visually analyzing as RSM.
6.4 UH MDO Framework Over Grid

6.4.1 Architecture

The architecture of the MDO Framework is shown in following figure 6.1 [Saran 2005, Nilakanta 2006].

Figure 6.1: UH MDO Framework Architecture [Nilakanta, 2006]

UH MDO framework has the following main components:

1. **Graphical User Interface (GUI)** - is the interface for interaction between the user and framework.

2. **MDO Controller** - is the interface between optimizer, execution manager and service directory. This provides the user what services are available, how to execute the activation graph i.e., either sequential or parallel execution of analysis services.
3. **Database** - is Relational Database Management System (RDBMS). PostgreSQL is used as a database engine. Database stores the data dictionary information about various analysis software and wrappers, analysis and optimization variables and their solutions.

4. **Optimization manager** - manages the execution of various optimizers. There can be a multiple optimizers available in the framework.

5. **Optimization Services** - are grid services in the grid based environment. Available optimizer services can be integrated with the framework.

6. **Service Directory** - contains the list of the services available on the grid and their corresponding FQDN (Fully Qualified Domain Name) to tell where the service is located. The nodes on the grid must register with this service directory to know what services that grid can contribute to the framework.

7. **Execution Manager** - does the job of distributed execution of analysis modules. Given multiple analysis modules located on the network and a sequence of execution of these analysis services, the execution manager contact corresponding analysis managers to execute required analysis. If possible execution manager executes some of the analysis services in parallel.

8. **Name Server** - Any distributed system requires a mechanism to locate resources on the network. Either it can be done statically through configuration files or dynamically through name resolution. UH MDO framework supports locating the services dynamically. Name server keeps track of all components running in the system.

9. **Analysis Service Manager** - analysis service manager is a process running on each computer where analysis services are located. The analysis manager controls execution of all analysis modules.
10. **Analysis Services** - Analysis service is wrapped disciplinary analysis software. This is the only component, which is created by service provider. Analysis services are the grid services that can be accessed across the network.

6.4.2 **Problem Formulation and Construction in the UH MDO Framework**

A Java based Graphical User Interface is developed to create activation graphs and to execute the activation graphs. The user employs the framework GUI to define the analysis sequence. Through the GUI, the user identifies the execution sequence of different services along with their corresponding resource properties.

MDO Framework uses multithreading approach to perform the parallel execution is identified in the execution sequence. Directed edges in the sequence diagram denote the events, for which each thread waits on.

GMIS viewer supports creation and execution of activation graphs. It also provides the status of job execution. So instead of UH MDO Framework GUI, GMIS viewer can be used to formulate the problem.

6.4.3 **Problem Execution in the UH MDO Framework**

The framework automates the execution of the various services drawn in activation graph. For distributed computing support, Globus Toolkit 4 (GT4) features were used which could provide a remote service invocation. The activation graph can be executed in parallel. The flow of the execution depends upon the notification from one service to other service.

GMIS supports the distributing computing support over WAN so GMIS services can be used for UH MDO Framework problem execution.

6.4.4 **Information access in the UH MDO framework**

The information regarding different domain names, services, their resource properties, and the input variables are stored in the database. The retrieval of the information is done while executing the framework. The transferring of files from one
system to other is transferred using *gridftp*, which is provided by Globus toolkit. These transferred files are displayed dynamically based upon the arrival of data.

GMIS services provide all resource information including applications information. In GMIS framework input/output files transfer happens through File Transfer Protocol (FTP). Using GMIS resource selector and monitor services required information can be accessed for UH MDO framework.

### 6.4.5 Mapping of UH MDO Framework Components to GMIS components

UH MDO framework has developed based on Globus, which can be migrated to GMIS as all the required components of UH MDO framework will be provided by GMIS framework. The following table gives the mapping of UH MDO Framework components to GMIS components.

<table>
<thead>
<tr>
<th>UH MDO Framework component</th>
<th>GMIS component</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI</td>
<td>Viewer</td>
</tr>
<tr>
<td>MDO Controller</td>
<td>Topology Manager</td>
</tr>
<tr>
<td>Database</td>
<td>Database</td>
</tr>
<tr>
<td>Optimization Manager</td>
<td>Resource Selector</td>
</tr>
<tr>
<td>Service Directory</td>
<td>Monitor Service</td>
</tr>
<tr>
<td>Execution Manager</td>
<td>Resource Selector</td>
</tr>
<tr>
<td>Name Server</td>
<td>Data Collection Manager</td>
</tr>
<tr>
<td>Analysis Service Manager</td>
<td>Resource Selector</td>
</tr>
</tbody>
</table>

*Table 6.1: Mapping of MDO Framework Components to GMIS Components*

### 6.5 UH MDO Application Design on GMIS viewer

UH MDO application design on GMIS viewer will be done in two levels, one is creation of activation graph and second one is execute activation graph. These two steps are discussed in detail in the following sub sections.
6.5.1 Creation of Activation Graph

The following steps are used to create activation graph on GMIS viewer.

1. Select a drawing Space for Activation graph, which can be done by selecting “Jobs->Activation Graph -> Create” option in the menu bar of viewer. The drawing space created is shown in the following figure 6.2.

2. First give activation graph name for the activation graph which is going to be created. Then click on “Add Task” button to add a task. This will be the first task for an activation graph. There will be list options to select the node using FQDN names present in GMIS. After selecting the system, next list shows the services available in that particular node. Select service which needs to be executed. After that need to mention the input file location. Finally need to mention the dependent task(s) which needs to be executed before executing this task. If it is ‘0’ means this task is either the first task or can be executed in parallel. This is shown in the following figure 6.3.
3. Next select “Save” button on the viewer. It shows a rectangular text box with task number as shown in the following figure 6.4.
4. Repeat steps 2 – 3 to add multiple tasks. Following figure 6.5 shows after adding three tasks to the activation graph.

![Image of Activation Graph with 3 Tasks]

**Figure 6.5: After Creation of Activation Graph with 3 Tasks**

### 6.5.2 Execution

Two options are provided to execute the Activation Graph. First option is after creation of activation graph click on “Execute” button to execute the tasks according to the order mentioned. Second option is select “Execute” option from Jobs-> Activation Graph -> Execute option from the file menu. After selecting this menu it populates all the activation graphs created so far in a list. Select an activation graph which needs to be executed. Selected activation graph will be shown and option is provided to execute the activation graph. The screen shot of this option is shown in the following figure 6.6.
6.6 Virtual Cluster

A virtual cluster is defined as a set of virtual machines configured to behave as a cluster and intended to be scheduled on a physical resource at the same time [Xuehai Zhang, 2005]. In other words, virtual cluster is a group of physical or virtual machines configured for a common purpose.

Virtual cluster is another application of GMIS framework. As all the resources information is stored in database and topology manager maintains the resources hierarchy, a user defined virtual cluster can be formed as shown the following figure 6.7.

In this figure 6.7 there are two physical clusters Cluster I and Cluster II. Cluster I contains three nodes and Cluster II contains two nodes. The requirement is to have a new cluster which contains node 1 of Cluster I and Node 5 of Cluster II. In reality forming such a cluster is expensive. However a virtual cluster can be constructed with these two nodes, since all the node information is available in the GMIS's database and nodes are maintaining by the topology manager. Virtual cluster formation can be done by using the user based discovery resource option of GMIS viewer. To create a virtual
cluster first create a cluster manually using the option provided on viewer and then add required nodes to this cluster manually using the interface provided on viewer. This cluster will be created under default grid. If any particular grid name required then first create a grid manually followed by adding clusters and nodes.

6.7 Summary

UH MDO framework has been developed as a general programming environment for automating the distribution of complex computing tasks on nodes connected over a grid network. The framework system facilitates communications between computational tasks distributed over a grid network and provides the automatic interactions in multidisciplinary problems. Globus was selected as a distributed computing environment for UH MDO framework. This chapter discussed how GMIS services can be used for UH MDO framework, as all services required for UH MDO framework are provided by the GMIS. Hence UH MDO framework is chosen as a case study for GMIS framework. Virtual cluster formation using GMIS is also discussed in this
chapter. These two applications demonstrate how GMIS services can be used as distributed computing environment over WAN.

Next chapter concludes the thesis and discusses the enhancements of GMIS framework.