XML-Based Visual Specification of Multidisciplinary Applications

Ahmed Al-Theneyan[a] Amol Jakatdar[a] Piyush Mehrotra[b] Mohammad Zubair[a]
[a]*Computer Science Department, Old Dominion University, Norfolk, VA 23529 USA*
and
ICASE, MS 132C, NASA Langley Research Center, Hampton, VA 23681 USA
{theneyan, ajakatda, zubair}@cs.odu.edu
[b]*NAS Division, M/S T27A-1, NASA Ames Research Center, Moffett Field, CA 94035 USA*
pmehrotra@arc.nasa.gov

Abstract

The advancements in the Internet and Web technologies have fueled a growing interest in developing a web-based distributed computing environment. We have designed and developed Arcade, a web-based environment for designing, executing, monitoring, and controlling distributed heterogeneous applications, which is easy to use and access, portable, and provides support through all phases of the application development and execution. A major focus of the environment is the specification of heterogeneous, multidisciplinary applications. In this paper we focus on the visual and script-based specification interface of Arcade. The web/browser-based visual interface is designed to be intuitive to use and can also be used for visual monitoring during execution. The script specification is based on XML to (a) make it portable across different frameworks, and (b) make the development of our tools easier by using the existing freely available XML parsers and editors. There is a one-to-one correspondence between the visual and script-based interfaces allowing users to go back and forth between the two. To support this we have developed translators that translate a script-based specification to a visual-based specification, and vice-versa. These translators are integrated with our tools and are transparent to users.

1. Introduction

The advancements in the Internet and Web technologies have fueled a growing interest in developing a web-based distributed computing environment. We have designed and developed Arcade [3], a web-based environment for designing, executing, monitoring, and controlling distributed heterogeneous applications. The focus of Arcade is to support simulations and computations that consist of multiple heterogeneous modules interacting with each other to solve an overall design problem. For example, Multidisciplinary Design Optimization (MDO) methods are being explored at NASA Langley Research Center (LaRC) for the design optimization of aerospace vehicles [12]. Typically the modules in such applications are developed in different disciplines and are optimized independently. The traditional path for integrating these modules, through the use of scripts, makes the process of specifying and optimizing the overall design of such applications a long and tedious process often taking several weeks. The slowness of this process is mainly due to the absence of a collaborative environment where (i) different modules and their interactions can be specified, and (ii) testing, monitoring, and steering of the overall design can be done by multiple users from different disciplines concurrently.

The objective of Arcade is to design an environment, which is easy to use, easily accessible, portable, and provides support through all phases of the application development and execution. We implement various parts of the environment by leveraging off commodity technologies, such as the Web, Java, and Jini along with service layers such as Globus [5] being implemented by various research groups. These technologies are capable of seamlessly interconnecting disparate hardware platforms running different operating systems across diverse locations providing an ideal environment for distributed simulation of complex systems. The problem is that these technologies are relatively new and there is not enough experience with them in building such a framework for large-scale distributed applications. The major issues that need to be considered for building such an environment are:

- Design of a protocol over HTTP to support communication between different components of the environment that addresses the MDO-like application requirement.
- A visual tool to enable users to specify a MDO-like application. Also, related to this issue is the design of a visual language specification that has one-to-one correspondence with the visual representation.
- Application Programming Interfaces (APIs) specification to allow different visualization and

resource management tools to easily work with the proposed environment.

- Collaboration support in the design, execution, and monitoring phases of a distributed heterogeneous application.
- Multi-domain support to allow users from different domains to work together.

The focuses of this paper are the visual- and script-based interfaces for the Arcade framework. To better understand the requirements on these interfaces, consider a use case scenario for developing and executing a distributed application. A team of designers collaboratively develops the application consisting of a hierarchical set of modules. That is, individual members are made responsible for specifying the submodules while the project leader is responsible for the overall integration of the application, i.e., connecting the outputs of one module to the inputs of another. For this purpose, members can use visual- or script-based specification. Note that the modules can range from simple sequential programs to data parallel programs capable of execution on a multiprocessor or a network of workstations, to more complex subsystems, which are defined hierarchically through the use of submodules. Once developed, the application is executed in a distributed environment using a heterogeneous network of workstations and multiprocessor machines. During the execution, team members sitting at their individual workstations simultaneously monitor the flow of progress of the application. That is, the team members can see the currently executing modules at any level of the hierarchy. They can also monitor the intermediate data between different modules using visualization tools to view large data sets. A team member responsible for a particular subsystem can change data values under the control of the subsystem in order to steer the computation in the right direction. The team member can also dynamically alter the control flow if necessary. For example, in a design cycle, the responsible team member may decide that a particular module is not affecting the optimization and may bypass the module by using old values in each cycle. Similarly, the team could replace a module with a plug-compatible module, for example, to use another algorithm. Once the execution is complete, team members again can examine the final results using the visualization tools.

In this paper we describe the visual- and script-based specification interfaces of the Arcade system. The web/browser-based visual interface is designed to be intuitive to use. Once specified the same visual representation of the application can also be used for visual monitoring during execution. The script specification is based on XML. Using XML allows us to leverage off existing freely available XML parsers and editors to develop our tools. Also, such an XML-based script presents the potential of inter-framework portability. Thus, if a piece of the overall application needs to be executed by another framework, we could translate that portion of the XML specification into the framework specific representation. There is a one-to-one correspondence between the visual- and script- based interfaces allowing users to go back and forth between the two. Thus, some users will specify the application visually and then use the script representation to make changes. On the other hand, other users may be more comfortable writing the XML script using an offline editor and then using the visual representation for execution. To support this we have developed translators that translate a script-based specification to a visual-based specification, and vice-versa. These translators are integrated with our tools and are transparent to users.

The rest of this paper is organized as follows. In Section 2, we present some of the related work. Section 3 describes the overall architecture of the Arcade system. Section 4 focuses on the application specification prototype in Arcade and, finally, in Section 5 we conclude by summarizing the work that we have described in this paper.

2. Related Work

Several software systems have been developed that make distributed computing available to an application programmer. These can be distinguished into different categories. The first category of environments includes systems such as MPI [11], PVM [16], pPVM [10], and JAVADC [4]. All these environments support distributed computing in varying degrees of generality; however, either they are not web based or they lack collaborative features. Also, they are mostly suitable for running SPMD programs.

The second category consists of environments that focus on large distributed heterogeneous codes, but are generally specific to a single application domain. Examples of such environments include FIDO [18], MIDAS [14], NetSolve [2], and Ninf [13]. However, both systems are either hardwired to a specific problem area or are too restrictive. The other major limitation is that they lack a collaborative environment, which would permit different members in a group to interact with the application at various stages of its design and execution.

The third category of environments that includes IceT [6], Programmer's Playground [7], PRE [15], VDCE [17], HeNCE [8], and WebFlow [1], supports heterogeneous distributed applications in some form or other. For example, the Grid Enabled Console COmponent (GECCO) [9] is a graphical tool that has been built on top of Globus [5] to allow users to interactively specify and monitor the execution of a set of tasks with dependencies between them. GECCO enables the user to formulate job execution as a task graph.

The front-end for most of these systems is generally some variation of large-grained data flow graphs with computational modules being triggered when their inputs are available. In our experience, we have found that to easily express heterogeneous applications requires more control structure than provided by such data flow-based systems. For example, in a multidisciplinary optimization code, the optimization cycle would have to be embedded within a module in a system that only supports data flow rather than being explicit at the outer level. Also, these systems mainly concentrate on different aspects of the infrastructure required for managing the execution and interaction of the modules making up the application whereas the goal of the project described here is to build an integrated framework for all phases of the design and execution of distributed heterogeneous applications. However, such collaborative systems do not provide any support for the management and steering of the execution of distributed applications.

3. Overview of Arcade

Arcade is a web-based integrated metacomputing environment that is being built to provide support for a team of discipline experts to collaboratively design, execute, and monitor multidisciplinary applications on a distributed heterogeneous network of workstations and parallel machines. This framework is suitable for applications, which in general consist of multiple heterogeneous modules interacting with each other to solve the overall design problem, such as the multidisciplinary design optimization of an aircraft. As shown in Figure 1, Arcade is based on a three-tier architecture. The first tier is a web-based, lightweight client, which provides the user interface to the whole system. It consists of applets, which allow users to design an application, monitor and allocate resources, and execute, monitor and steer the application in a collaborative manner. It also has interfaces, which allow the system administrator to manage the system including resource registration and user management and authentication. Most of the logic of the system is contained in the Java-based middle tier. Among other modules, the middle tier consists of the *User Interface Manager*, which provides logic to process the user input and coordinate between the other components; the *Execution Manager*, which manages the overall execution of the application; the *Data Manager*, which manages the shared data; the *Resource Manager*, which manages the distributed heterogeneous resources of the system; and the *Security Manager*, which controls the access of the system. The third tier consists of the distributed resources that are used to actually execute the user modules and application codes. A lightweight controller executes on each resource providing a gateway to the resource.

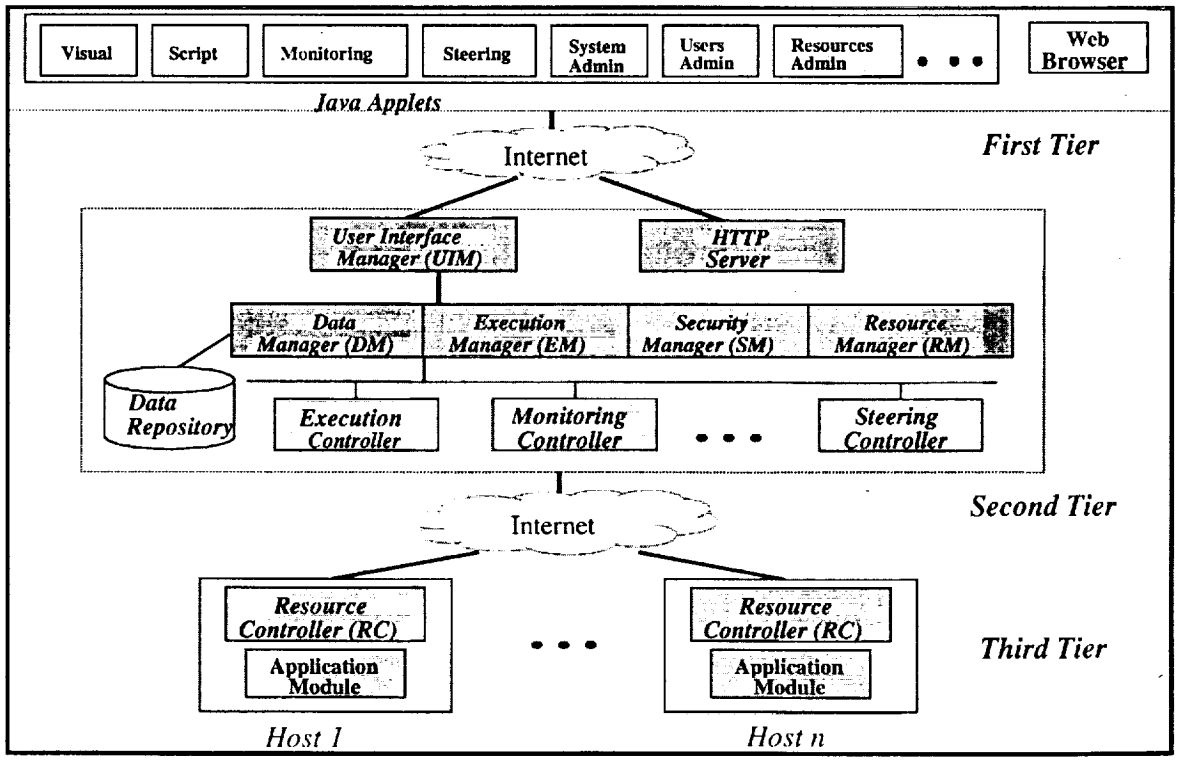


Figure 1. The ARCADE System Architecture

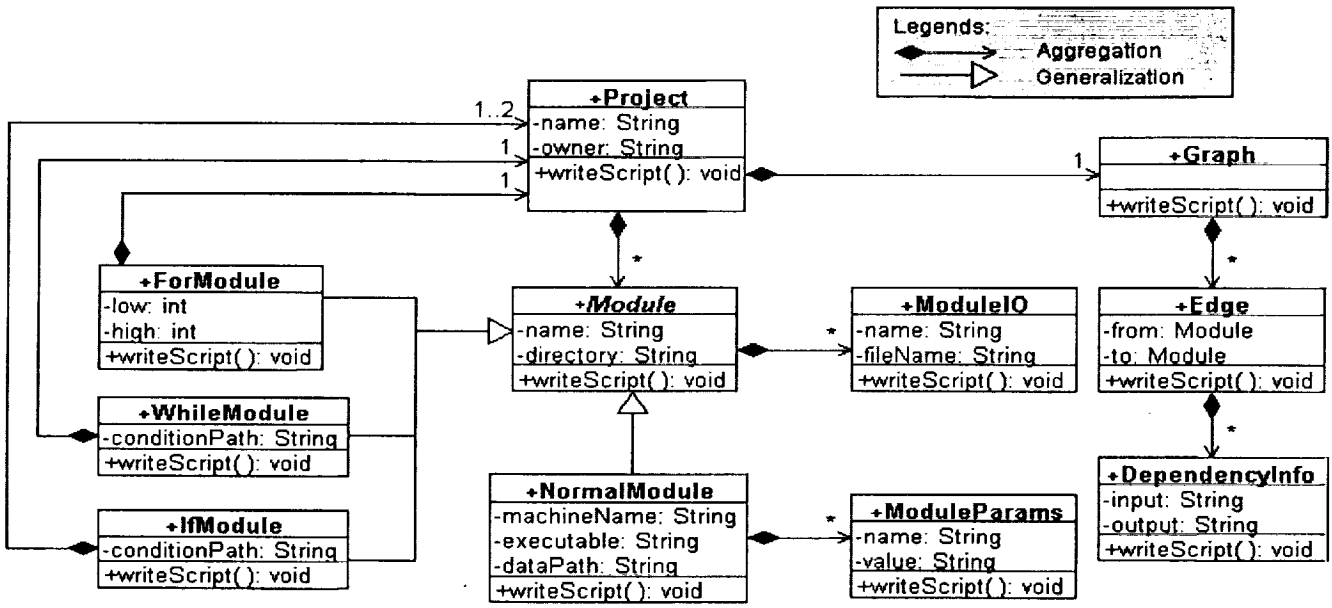


Figure 2: Representation of *Project* and associated objects using UML

4. Application Specification

In our framework, a distributed application consists of a collection of heterogeneous modules (application codes from different disciplines). We are targeting applications where these modules are very coarse grained. A typical distributed application requires these modules to be executed in some order and possibly on different machines. For certain problems, a set of modules may need to be executed iteratively, for example, until a desired optimization criterion is reached.

In Arcade, each application is internally represented as a Java *Project* object. Figure 2 provides the description of the *Project* and associated objects in UML. Here *aggregation* means the *contains* relationship. *Generalization* means the *is-a* relationship. The number on the arrow specifies the number of instances contained. For example, * on the *aggregation* between *Project* and *Module* means a *Project* can contain 0 to arbitrarily many *Modules*. The *generalization* between *Module* and *NormalModule* implies that *NormalModule* is a kind of *Module*.

The *Project* object consists of a vector of *Module* objects. This is the central object in our framework. All the information related to the application, both static and dynamic, is stored within this object. The *Project* object is a complex object that is shared by all the processes of the middle tier, (cf. Figure 1) and supports methods that are used by these processes.

To be able to support a wide variety of distributed applications, we support different types of modules. All these modules have a common set of properties and, hence, are derived from a general *Module* object. Some common attributes of the *Module* object are:

- *Module Name*
- *Module Directory*
- *Input/Output Names.*

The following types of modules derive from the general *Module*:

- ***Normal Module*:** This is the basic module in our framework and is used to represent the executable parts in the applications. A *Normal Module* is identified by its executable code, command line, arguments, resource requirements, and input/output file requirements.
- ***Loop Modules*:** These modules allow a set of "*internal*" modules to be iteratively executed. There are two kinds of looping modules: *For Module* for a predetermined number of iterations and the *While Module*, where the iteration condition is tested at the beginning of the loop. These modules have a *Project* object, which represents the set of internal modules.
- ***If Module*:** This module provides a mechanism for testing the value of a condition. The truth-value of the condition determines whether the modules in the *then-block* or the *else-block* (if present) will be executed.

- ***Hierarchical Module***: An abstract *Module* representing a subgraph, i.e., a recursively defined collection of modules.

In the current prototype there are two ways to specify distributed applications: visual based and XML-script. We describe these two approaches in the next two subsections. The first subsection talks about the visual specification of the project and how a user can visually specify an application. The second subsection talks about how an application can be represented using the script-based XML format.

4.1. Visual Specification

The visual specification applet allows a user to graphically specify a heterogeneous application. The objective is to support a visual specification, which is intuitive to build, can be used for visual monitoring, and works with the Web.

We have implemented a Java applet, as shown in Figure 2, that provides a visual specification interface and addresses some of these issues. In the background of the figure, we can see the *Application Editor* window where the user can graphically specify the modules that comprise the application and their intercommunications. When a module is specified, a separate pop-up window appears allowing the user to specify the properties of the module. In the bottom left of the figure, we have the module information window for the *Normal Module* M1.

The application can be seen as a graph where a node represents a module and the arcs represent data flow. It is easy to see how a data flow-based application can be modeled using such a system. When the output port of a module is connected to the input port of another module, another pop-up window appears allowing the user to specify the interconnection. In the upper right of the figure, we have a dependency information window where the user is specifying a data flow between M1 and M2.

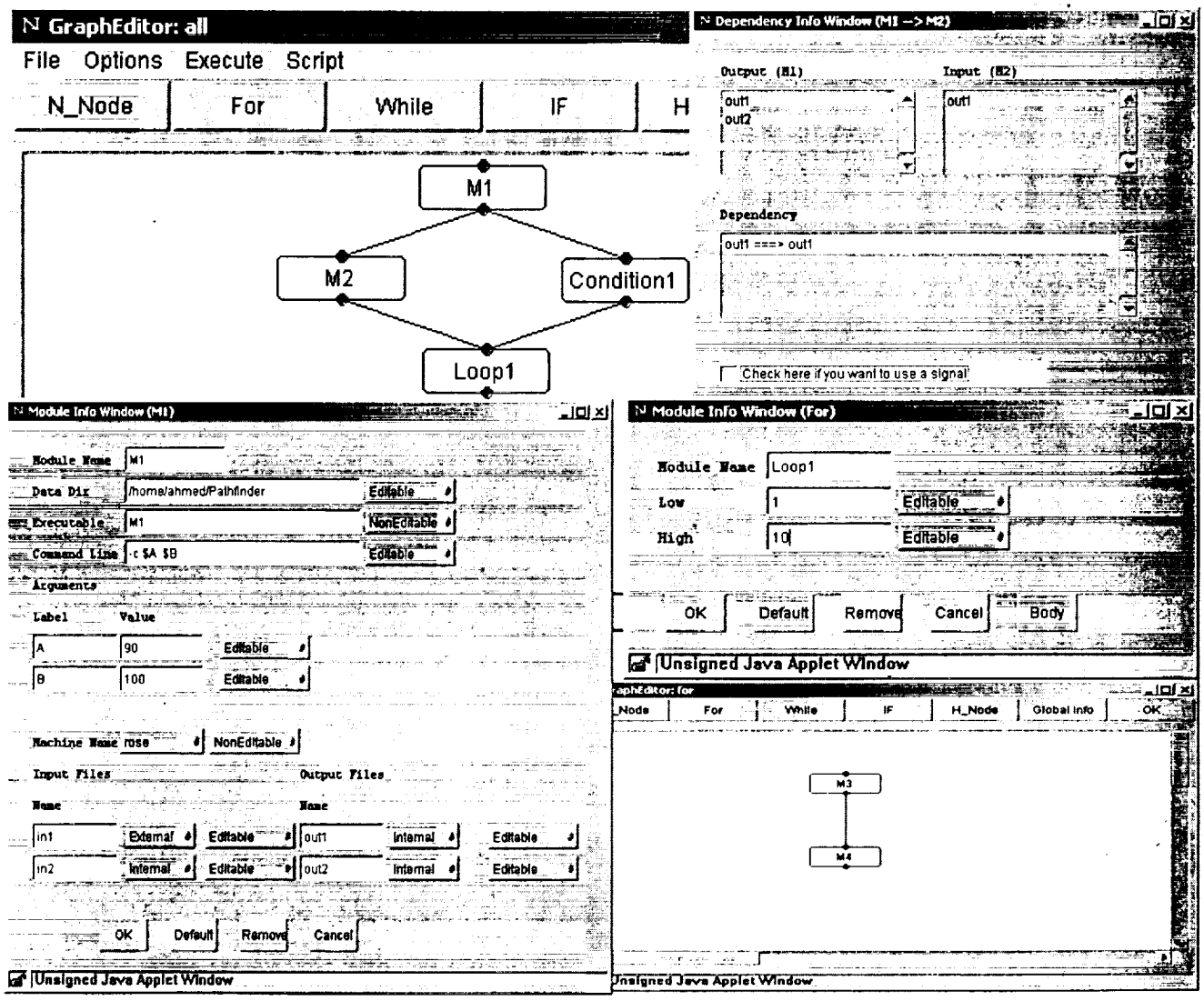


Figure 3. Snapshots of the visual specification in Arcade

It becomes a little trickier to accommodate control structures such as conditionals and iterations, in particular when we want to use the visual specification for monitoring too. We accommodate *If Modules* and *Loop Modules* by restricting their bodies to be *Hierarchical Modules*, which are specified through a separate window. Thus, the modules labeled *Then-block* and *Else-block* represent *Hierarchical Modules* abstracting the then and else part of the *If Module* construct, respectively. Similarly, the module *Body*, represents the loop body of the for loop. Restricting the bodies of control structures to *Hierarchical Modules* eases the task of specification and allows the application to be visually represented. However, it does not provide an integrated view of the whole application in a single window, i.e., the body of a control structure is always shown in a separate window. In the bottom right of Figure 3, we have the module information window for the *Loop Module* Loop1 along with its body.

The visual representation of an application described above can also be used to monitor an application during execution. A monitoring applet uses a pre-determined color-scheme to indicate modules, which are pending, are currently executing, and have finished execution.

4.2. XML-script Specification

Arcade supports XML based specification of a heterogeneous application. The syntax of the script is simple and the DTD for XML specifications can be found at http://www.icase.edu/arcade/project.dtd. The XML-based project specification consists of two parts: specification of different types of modules used in the *Project* object, and the dependency graph between these modules. We now give sample XML specifications for the *Project* followed by different types of modules and the *dependency graph*. For simplicity, only important attributes are shown here. For detailed, functionally correct, and specifications please refer to the DTD.

As explained earlier in this section, all the modules are bundled in a *Project*. A sample *Project* object looks like this in XML:

```
<Project name="PathFinder" owner="ajakatda">
    <XModule ...>
    .                          } n number of times.
    .
    </XModule ...>
    <Graph>
    .
    .
    </Graph>
</Project>
```

Here, *XModule* can be one of *Normal*, *For*, *While*, or *If* modules described below. The *Graph* describes the dependency information about the modules that comprise the *Project*. The *Project* also has a *name* and an *owner*.

As we described earlier, all the modules are inherited from a general *Module* object. A sample *Module* looks like this:

```
<Module name="M1" directory="/home/ahmed/PathFinder "
 startx="260" starty="52" endx="260" endy="28" >
      <ModuleIO type="Input" name="IOFile0"
      filename="in1" external="i" editable="Y"/>
      <ModuleIO type="Output" name="IOFile1"
      filename="out1" external="i" editable="Y"/>
      .
      .
</Module>
```

Along with the basic information, *Module* also stores information about its input and output, and some graphical information required by the visual specifications.

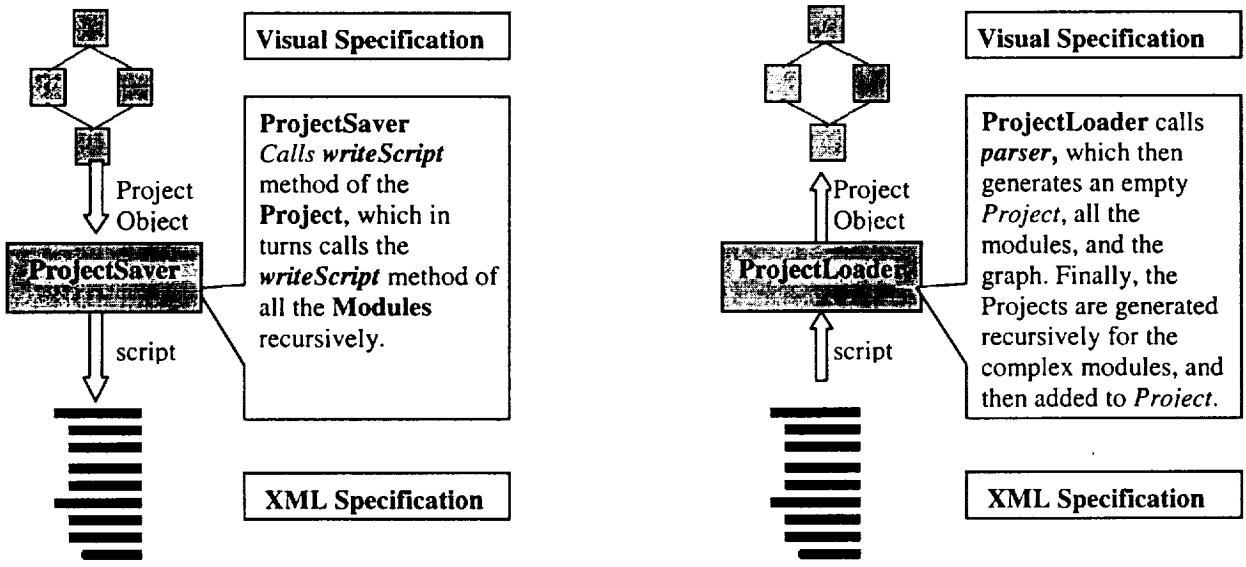
A *Normal Module* is the basic module in our framework. A sample XML specification of the *Normal Module* looks like this:

```
<NormalModule dataPath="/home/ahmed/PathFinder "
   commandLine="-c $A $B" machineName="rose"
   machineNameEditable="false" execName="M1">
    <Module name="M1" ...>
    .
    .
    </Module>
    <ModuleParam name="A" value="90"/>
    <ModuleParam name="B" value="100" />
</NormalModule>
```

As one can notice along with the basic information like executable code, command line, argument, and resource requirements, it also stores some administrative information, like can a user edit the parameters, and some functional information such as machine name to execute the *Normal Module*.

A *For Module* is used for representing the loops of fixed repetitions. A sample *For Module* looks like this in XML:

```
<ForModule low ="1" high ="10">
    <Module name="Loop1" ...>
    </Module>
    <Project name="Loop1" owner = "ajakatda">
    <NormalModule ...>
    .
    .
    </NormalModule>
    .
    .
    </Project>
</ ForModule>
```



(a) (b)

Figure 4: Storing and retrieving of the *Project* object in XML

An *If Module* is used for representing the decision scenario. A sample *If Module* looks like this in XML:

```
<IfModule conditionPath="/home/PathFinder/If/cond" >
    <Module name="IF" ...>
    .
    .
    </Module>
    <Project name="thenProject" owner = "zubair">
    .
    .
    </Project>
    <Project name="elseProject" owner = "zubair">
    .
    .
    </Project>
</IfModule>
```

The *If Module* also has a child called *Module*, as it is also a type of general *Module*. Notice that *If Module* may have two *Projects* in it. These are separated from each other by their order. The first one is always treated as a *Then Project* and the second one, if exists, is treated as an *Else Project*.

The dependency is implemented using files or signals. Consider sample modules M1 and M2, where an input of M2 depends on an output of M1, (cf. Figure 3). The Graph specification will look like this:

```
<Graph>
  <Edge From="M1" To="M2">
  <DepInfo Output="out1" Input="out1"/>
  </Edge>
</Graph>
```

4.3 Storing and Retrieving of the *Project* object

The user can design the project online using the visual tools provided, or he/she can write an XML script specification offline. We have to provide support for conversion between these two formats.

Storing the *Project* Object: The ProjectSaver Class starts the process of saving the *Project*. Each Class, extending from *Module* class, implements a procedure to write itself to the file in XML format. So all the components write themselves recursively to the specified file (see Figure 4(a)).

Retrieving the *Project* Object: The *Project parser* class implements the XML parser, which reads the XML file and generates a *Project* along with its visual information. It does so by generating an empty *Project* first and then adding individual *Modules* to it. As shown in Figure 4(b), the *Project* loader class calls this parser.

5. Conclusion

In this paper, we have described the application specification interfaces for Arcade, a web-based environment for distributed heterogeneous applications. We describe two interfaces: visual and script based. The visual interface has been designed to allow users to drag and drop modules providing the information required for each module. The dependencies between modules can

also be specified graphically. The system also supports control dependencies using hierarchical modules to specify the bodies of loops and the then and else blocks of conditionals. Such an approach shows just the data dependencies at each level, hiding the control structure in the hierarchy. The visual representation is, thus, clean with no cluttering of control and data dependencies. However, this approach does not seem to provide an overall view of the application in a single window forcing users to look through multiple windows. We are currently experimenting with other views.

The second interface is script based. We use an XML-based script for offline specification of the application. We can, thus, leverage off XML tools to build the interfaces. The use of XML also provides a more standard approach to specifying distributed applications that presents the opportunity of inter-framework portability of such specifications. We are currently exploring the translations required for such inter-operability.

Acknowledgment

This work was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the authors were in residence at ICASE, NASA Langley Research Center, Hampton, VA 23681.

References

[1] D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski, and G. Premchandran, "WebFlow: a visual programming paradigm for Web/Java based coarse grain distributed computing," Concurrency: Practice and Experience, Java Special Issue, vol 9(6), March 1997, pp. 555-578.

[2] H. Casanova and J. Dongarra, "NetSolve: A Network Enabled Server, Examples and Users," Proceedings of Heterogeneous Computing Workshop, Orlando, Florida, 1998.

[3] Z. Chen, K. Maly, P. Mehrotra, and M. Zubair, "ARCADE: A Web-Java Based Framework for Distributed Computing", WebNet 99, October 1999.

[4] Z. Chen, K. Maly, P. Mehrotra, P. Vangala and M. Zubair. "Web-based Framework for Distributed Computing," Concurrency: Practice and Experience, Java Special Issue, vol 9(11), November 1997, pp. 1175-1180.

[5] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," International Journal of Supercomputer Applications, Volume 11(2), 1997, pp. 115-128.

[6] P. Gray and V. Sunderam. "IceT: Distributed Computing and Java," Concurrency: Practice and Experience, Java Special Issue, vol 9(11), November 1997, pp. 1161-1168.

[7] K. J. Goldman, B. Swaminathan, T. Paul McCartney, M. D. Anderson, and R. Sethu-raman. "The Programmers' Playground: I/O Abstraction for User-Configurable Distributed Applications," IEEE Transactions on Software Engineering, 21(9), September 1995, pp. 735-746.

[8] HeNCE (Heterogeneous Network Computing Environment), http://www.netlib.org/hence/.

[9] G. Von Laszewski, I. Foster, "Grid Infrastructure to Support Science Portals for Large Scale Instruments," Proceedings of the Workshop Distributed Computing on the Web (DCW), 1999.

[10] K. Maly, S. Kelkar and M. Zubair. "Scientific Computing using pPVM," International Conference on Parallel Processing, vol 2, August 1994, pp. 201 - 205.

[11] Message Passing Interface Forum. "MPI: A Message-Passing Interface Standard Version 2.0 Technical Report," Computer Science Department, University of Tennessee, Knoxville, Tennessee, 1997.

[12] Multidisciplinary Optimization Branch (MDOB) at NASA Langley Research Center, http://fmad-www.larc.nasa.gov/mdob/MDOB/index.html.

[13] H. Nakada, M. Sato and S. Sekiguchi, "Design and Implementations of Ninf: Towards a Global Computing Infrastructure," Future Generation Computing Systems, Metacomputing Issue, 1999.

[14] J. C. Peterson, "Multidisciplinary Integrated Design Assistant For Spacecraft (MIDAS)," http://mishkin.jpl.nasa.gov/Midas Page.

[15] Product Realization Environment, http://wwwcollab.ca.sandia.gov/pre.

[16] V. Sunderam. "PVM: A Framework for Parallel Distributed Computing," Concurrency: Practice and Experience, vol 2 No 4, December 1990.

[17] H. Topcuoglu, S. Hariri, D. Kim, Y. Kim, X. Bing, B. Ye, I. Ra, and J. Valente, "The Design and Evaluation of a Virtual Distributed Computing Environment," The Journal of Networks, Software Tools and Applications, Cluster Computing, 1998.

[18] R. P. Weston, J. C. Townsend, T. M. Eidson, and R. L. Gates. "A Distributed Computing Environment for Multidisciplinary Design," 5th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Panama City, FL, AIAA 94-4372, September 7-9, 1994.