

AssistConf: a Grid configuration tool for the ASSIST parallel programming environment

R. Baraglia¹, M. Danelutto², D. Laforenza¹, S. Orlando³, P. Palmerini^{1,3}, P. Pesciullesi²,
R. Perego¹, M. Vanneschi²

¹Istituto ISTI, Consiglio Nazionale delle Ricerche (CNR), Pisa, Italy

²Dipartimento di Informatica, Università di Pisa, Italy

³Dipartimento di Informatica, Università Ca' Foscari, Venezia, Italy

Abstract

This paper presents AssistConf, a graphical user interface designed to configure an ASSIST program and to run it on a Grid platform. ASSIST (A Software development System based upon Integrated Skeleton Technology) is a new programming environment for the development of parallel and distributed high-performance applications. The main goals of ASSIST are allowing high-level programmability and software productivity for complex multidisciplinary applications, and performance portability across different platforms, including homogenous parallel machines and cluster/Beowulf systems, heterogeneous clusters, and computational Grids. AssistConf is used to configure the ASSIST program and establish a mapping between the program modules and the most suitable machines in the Grid candidate to execute them. It simplifies the creation of the XML ASSIST configuration file, giving users a graphical view of the XML file produced by the ASSIST compilation phase, and permitting an easy identification of the machines to be used for execution. Finally, the configuration file produced by AssistConf is used as input to the assistrun command, which drives the execution of the ASSIST program over the Grid.

1 Introduction

In the past, parallel applications were developed as *monolithic* entities, usually coded in a *single executable*. Nowadays parallel/distributed applications tend to be more and more multi-modular, written by several development teams using different programming languages, using multi-source heterogeneous data, mobile, and interactive. [8, 11]. These applications have a *multidisciplinary* nature, that is, they are composed of several different *disciplinary modules*

coupled and coordinated in a single system.

A new application development style based on components is thus becoming popular. According to this style, programmers do not start from scratch, but build new applications by reusing existing *off the shelf* components and applications. These components may be distributed across a wide area network. A parallel/distributed application could be seen as the composition of simpler components executed concurrently under the control of a work-flow description.

In order to co-ordinate the execution of a parallel/distributed application on heterogeneous systems, several research efforts were conducted in the past. It is worth mentioning one of them, the *skeletal programming* model, which presents some interesting relations with the component-based technology. A seminal work in this area was conducted by Murray Cole [5]. A skeleton is basically a high-order, pre-defined function modelling a complex computational scheme, which makes all the details involved in the parallel computation structure transparent to the programmer. A skeleton-based language is used to co-ordinate the parallel activity of the processes defined using standard imperative languages. Programmer are provided with a set of skeletons that can be used to structure (compose) the parallel application. An important property of this approach is that each skeleton could have an associated *performance model* indicating its run-time characteristics.

ASSIST (A Software development System based upon Integrated Skeleton Technology) [16] is a new programming environment for the development of parallel and distributed high-performance applications. It can be considered a research framework to study, experiment and develop a set of programming issues for parallel and distributed high-performance applications. From the point of view of the programming model, parallel components are defined by a proper merging of the features of the component-based technology and of the skeletons technology. An application

is structured as a graph of sequential or parallel components connected by typed streams according to a data-flow and/or a nondeterministic style. Parallel components can utilize external shared objects, which represent abstractions of system resources, such as data sets, program codes, devices, memory hierarchies, and so on, possibly accessed interactively.

In order to optimize the mapping of a given parallel application on a heterogeneous system, the performance models can be useful for predicting the cost of a particular resource allocation strategy (mapping). A Grid [10] can be seen as an *extreme* heterogeneous system, and consequently, some research groups interested in skeletal programming approach have recently decided to investigate on the exploitation of this approach to design Grid-aware applications, introducing enhancements to overcome some limitations of the classical skeletal approach. [13, 16].

In order to map application components onto a Grid, it is necessary to verify the availability of enough resources able to satisfy the application requirements. This process is indicated in literature as *resource discovering and configuration* [2] or, also *resource searching and selection* [12]. In general, it is not an easy task. It requires several phases that could be executed manually, by the application developer (the Grid user), or automatically, by a specialized software entity (e.g. a *resource broker* or a *resource selection service*). Typically, the resource searching and selection process is accomplished accordingly to the following steps:

- The application developer describes the resource requirements of his/her application, using a *resource specification language*[6, 14].
- The description based upon the specific *resource specification language* is thus sent to a Grid Information Service (e.g., MDS [7]), which is responsible for maintaining updated information about the current resources available in the Grid.
- The Grid Information Service collects all available information, and send back them to the user/broker. In case of a manual resource selection, those information could be returned to the user through a GUI that helps the user to select the resources.

In this paper we present *AssistConf*, a graphical user interface designed for the manual searching and selection of suitable Grid resources candidate to execute an ASSIST application. It simplifies the creation of the ASSIST configuration file (see Section 4) giving users a graphical view of the XML file produced by the ASSIST compilation phase, and permitting an easy identification of the machines to be used for the application execution. Finally, the configuration file produced by *AssistConf* is used as input to the

assistrun command, which drives the execution of the ASSIST program on the Grid.

This paper is organized as follows. Section 2 outlines the main characteristics of the ASSIST environment. Section 3 introduces the ASSIST Run-time system and its configuration for Grid execution. Section 4 gives an overview of *AssistConf*, a graphical user interface designed to configure an ASSIST program to run on a Grid platform. Finally, Section 5 concludes the paper by discussing some future works.

2 ASSIST

ASSIST is a new programming environment for the development of parallel and distributed high-performance applications. The proposal originates from previous research conducted in the structured skeleton-based, parallel programming field [3, 4] and aims to combine in a unified approach the interesting features of *skeleton* programming models [5, 9] and software component technology. The main goals of ASSIST are allowing high-level programmability and software productivity for complex multidisciplinary applications, and performance portability across different platforms, including homogenous parallel machines and cluster/Beowulf systems, heterogeneous clusters and network computers, and computational Grids.

Readers interested in details regarding the programming model of ASSIST, and the constructs provided by ASSIST-CL, the coordination language used to define and glue ASSIST software components, can refer to [16]. For the purposes of this paper, we can consider an ASSIST program as a graph, whose nodes are software components and the arcs are abstract interfaces that support streams, i.e. ordered sequences, possibly of unlimited length, of typed values. Components can be parallel or sequential modules. A sequential module is the simplest component expressed in ASSIST. It can be coded with any sequential programming language hosted by the ASSIST-CL (currently C, C++, and Fortran). It has an internal state, and is activated by the input stream values according to a non-deterministic data-flow behavior. A parallel component may be an ASSIST subgraph, e.g. an independently designed ASSIST program, or a parallel module expressed with a *parmod* construct, a sort of generic skeleton which can be programmed in such a way to emulate the most common specific skeletons, but which is also able to easily express new forms of parallelism (e.g. optimized forms of task + data parallelism, nondeterminism, interactivity), as well as their variants and personalizations [16].

In order to give just a flavor of ASSIST-CL syntax and expressive power, let us consider a simple example of parallel application composed by a pipeline P of three stages C_1 , C_2 , and C_3 . Suppose that C_1 and C_3 are sequential mod-

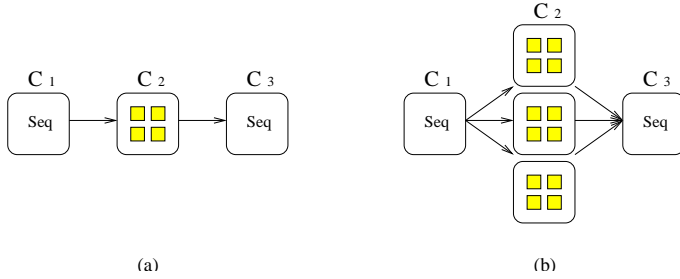


Figure 1. An example of ASSIST application structured as a three stage pipeline where the second stage is a *parmod* (a), or a *farm* of *parmods* (b).

ules, while C_2 performs a heavy data parallel task. C_2 is thus expressed by means of a *parmod* construct. The structure of this simple application is sketched in Figure 1.(a), while its ASSIST-CL code is:

```

pipe main (argc, argv)
{
  // global declaration of streams
  stream int s12;
  stream int [N1][N1] s23;

  // definition of the composition graph
  C1 (output_stream s12);
  C2 (input_stream s12, output_stream s23);
  C3 (input_stream s23);
}

C1 (output_stream s12 (int x))
  $C{
    < local declarations >
    x = fun (status);
    assist_out (s12, x);
  }C$

parmod C2 (input_stream s12 (int y);
          output_stream s23 (int [N1][N1] A))
  {
    // definition of the parallel module
  }

C3 (input_stream s23 (int [N1][N1] B))
  $C{
    < local declarations >
    consume(B);
  }C$

```

In the code above we can note the global declaration of the streams, and the definition of the composition graph. Each component is defined separately and its implementation can easily be changed without affecting the other modules. Returning to the previous example, we can increase the bandwidth of the second stage of the pipeline: 1) by increasing the parallelism degree of the *parmod*, or, 2) by replicating the stage through a *farm* construct. While the first solution only affects the configuration phase of the ASSIST program (see Section 4), the second one (sketched in

Figure 1.(b)) requires to lightly modify the high-level structure of the application in the following way:

```

pipe main (argc, argv)
{
  // global declaration of streams
  stream int s12;
  stream int [N1][N1] s23;

  // definition of the composition graph
  C1 (output_stream s12);
  MY_FARM (input_stream s12, output_stream s23);
  C3 (input_stream s23);
}

farm MY_FARM (input_stream s12 (int y);
              output_stream s23 (int [N1][N1] A))
  {
    C2 (input_stream s12; output_stream s23)
  }

```

The current implementation of the ASSIST environment is based on a flexible abstract machine and runtime support, which exploits the underlying mechanisms of ACE [15] and Distributed Shared Memory libraries. The compiler, realized according to the object-oriented technology, currently makes use of a set of pragmas for the sake of experimentation. The first version of the implementation currently run on homogenous parallel machines and clusters (Linux), and also contains basic interfaces for experimenting ASSIST in heterogeneous Grids. Work is in progress to define and to realize the next version of ASSIST, which will progressively remove some constraints, and will allow it to fully exploit heterogeneous large-scale platforms and Grids.

3 The ASSIST RTS and its configuration for Grid execution

ASSIST-CL is a coordination language aimed to increase software productivity for complex multidisciplinary applications. Among other innovative features, ASSIST-CL permits programmers to declare specific forms of parallelism (skeletons) that can be used to hierarchically compose sequential/parallel components. The adoption of a restricted number of well-known forms of parallelism allows us to define accurate performance models for them, and also to identify some component parameters that can be tuned to improve performance. This also simplifies the design of a (semi) automatic configuration tool, able to map the various components involved and tune their configuration parameters for each possible platform. Consider, in fact, that the target parallel architectures supported by the ASSIST programming environment range from homogeneous/heterogeneous clusters of sequential/SMP workstations to computational Grids. Hence, in order to ensure *code and performance portability*, programs need to be reconfigured on the basis of the specific features of each target architecture. For example, decisions like degree of parallelism of

data-parallel modules, number of replicated modules, mapping of components, etc. should be postponed till loading time, when the features of the target architecture - e.g. number and type of processors available - are known.

The reconfiguration of ASSIST-CL programs is possible because it is well supported by the CLAM (Coordination Language Abstract Machine), the original run-time support (RTS) of ASSIST-CL. To this end the CLAM is deployed through a set of processes, named CLAM-loaders, each of which usually runs on a distinct node of the chosen platform. The CLAM-loaders permit the compiled modules, which are generated by the ASSIST compiler as dynamic libraries (see Figure 2), to be loaded on the basis of an *XML configuration file*. Therefore we can consider the CLAM-loaders as a sort of containers for the ASSIST-CL modules. Besides allowing the execution of the various modules within internal threads, the CLAM-loaders implement additional services. For example, they are responsible for monitoring the program execution, and for dynamically reconfiguring the program in presence of load imbalance. CLAM-loaders and ASSIST modules are currently implemented on top of the portable ACE layer, which exploits in a very efficient way the thread and the TCP/IP socket libraries available on the OSs of the machines involved.

From this introductory description of the CLAM, it should be clear that, in order to execute an ASSIST-CL program, it is needed to launch first the CLAM-loaders. One of them will be elected master, thus becoming the coordinator of the activities of all the other loaders. The ASSIST XML configuration file will be passed to the master loader by the `assistrun` command provided in the ASSIST programming environment. This configuration file will contain loading information logically subdivided as follows:

- A static section that specifies the binary modules produced by the compiler for a given ASSIST program;
- A section that specifies the configuration of the program, i.e. degrees of parallelism and replication (parmods and farms).
- A section that contains mapping/loading information, i.e. mapping information about the CLAM-loaders, and loading information about the ASSIST module instances (as described and configured in the previous sections).

In order to build the two last two sections, the ASSIST programming environment provides a tool, called `AssistConf`. When we plan to exploit a single administrative domain for executing our applications, e.g. if we use a cluster located in our department, we could pre-launch the CLAM-loaders on the various nodes, and then dynamically choose how many loaders are involved in the execution of

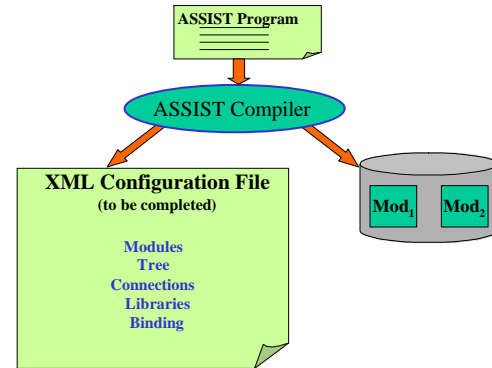


Figure 2. Overview of the ASSIST compilation process.

a given ASSIST-CL program through the XML configuration file. In this case, `AssistConf` will be simply responsible for configuring and mapping the modules on the chosen loaders. Conversely, in a Grid environment the computational nodes available may belong to distinct administrative domain and may change in the time. So `AssistConf` has the additional task of allowing users to select the Grid resources that best match their requirements. Once selected the machines and optimized the mapping of the modules, the `assistrun` command will be responsible for contacting the suitable Grid services in order to launch and co-allocated the CLAM-loader, before passing them the XML configuration file and compiled modules.

In the following we will discuss in more detail the configuration of an ASSIST-CL program, the related `AssistConf` tool, as well as the extensions to `AssistConf` to permit the brokering of the Grid computational resources needed for the program execution.

Configuration file. To introduce the syntax and the semantics of the ASSIST XML configuration file, consider Figure 2. The compiler produces the various modules, implemented as dynamic libraries handled by the ASSIST loaders, and a specific section of the ASSIST XML-based configuration file, i.e. the *Structure section*.

This section is static, and contains information about the software modules produced as dynamic libraries (*Modules element*), i.e. names, pathnames, etc., and information on the structure of the ASSIST program (*Tree element*), i.e. parallel constructs and their hierarchy. Moreover, in this section the stream interconnections between modules are specified (*Connections element*), the libraries used are identified (*Libraries element*), and, finally, the association between the ASSIST modules and the corresponding libraries

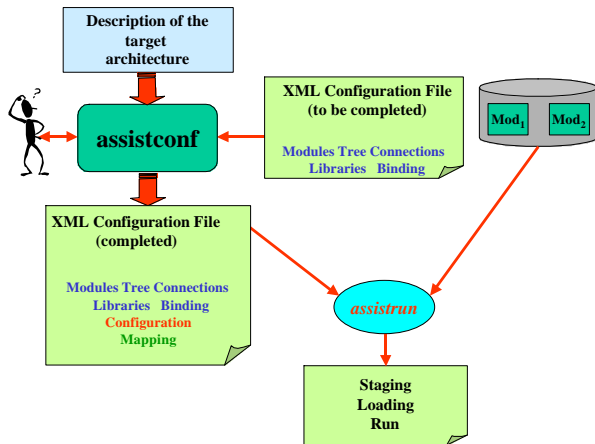


Figure 3. Structure and interactions among the configuration and mapping phases (AssistConf), and execution phases (assistrun).

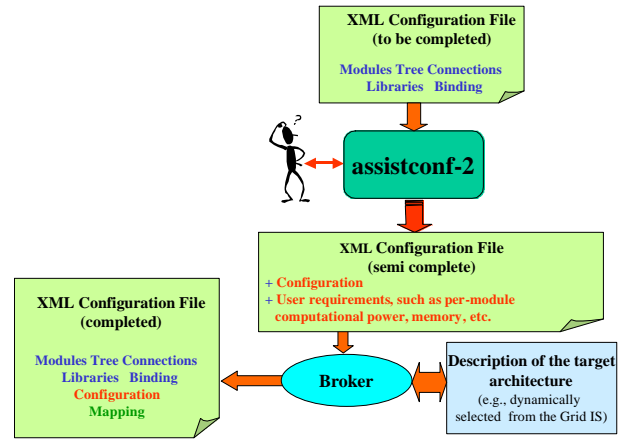


Figure 4. Structure and interactions among the configuration and mapping phases when the enhanced version of AssistConf will be adopted.

are established (*Bindings element*).

The *structure* section must be completed with further two sections, the *Configuration* section and *Loading* section. The former will contain information about the replication degree of some ASSIST modules (i.e. those included in a farm construct), and the parallelism degree of the *parmods*. The latter section will contain information about the nodes hosting the ASSIST loaders, and the mapping of the ASSIST module instances onto these loaders.

The whole structure of the XML ASSIST configuration file is thus the following:

```
<?xml version="1.0" ?>
<!DOCTYPE assist_config SYSTEM "ASSIST.DTD">
<assist_config>
  <structure >      .... </structure >
  <configuration > .... </configuration >
  <loading >       .... </loading >
</assist_config>
```

where the last two sections, configuration and loading ones, have to be produced by the AssistConf tool, as illustrated in Figure 3.

The complete XML configuration file will be then supplied to the *assistrun* command, which will perform all the needed steps to execute the program. On a Grid environment like Globus, these steps should require:

- To generate the RSL-ground commands, needed to launch the ASSIST loaders on the Globus GRAMs of the chosen nodes.
- To stage the libraries associated with the modules onto the chosen nodes of the Grid.

- To actually launch the ASSIST loaders (this step involves GRAM and DUROC services on Globus), and pass them the XML configuration file through the master loader.

Configuration tool. Figure 3 sketches a diagram that shows the interactive process through which users can select a set of Grid nodes (or pools of nodes), configure and map the ASSIST-CL modules, and, finally, run the program. AssistConf currently is a semi-automatic tool that facilitates the user in producing the final XML ASSIST configuration file through a user-friendly GUI. In the following section both functionalities and implementation of the interactive tool will be discussed in depth.

As future work, we plan to devise an enhanced version of AssistConf (see Figure 4), which will support users in configuring ASSIST-CL programs in a higher level way. Through this enhanced tool, users will only specify the mapping requirements of the various components of their program, by asking for machines with given characteristics in terms of computational power, disk space, memory, etc. A Grid *broker* will then determine the actual mapping and produce the final XML ASSIST configuration, by choosing the best Grid machine pools available on the basis on information supplied by the Grid Information Service.

4 AssistConf

AssistConf is a GUI written in Java. The main aim of the interface is to simplify the creation of the XML ASSIST configuration file. This entails giving the user a graphical

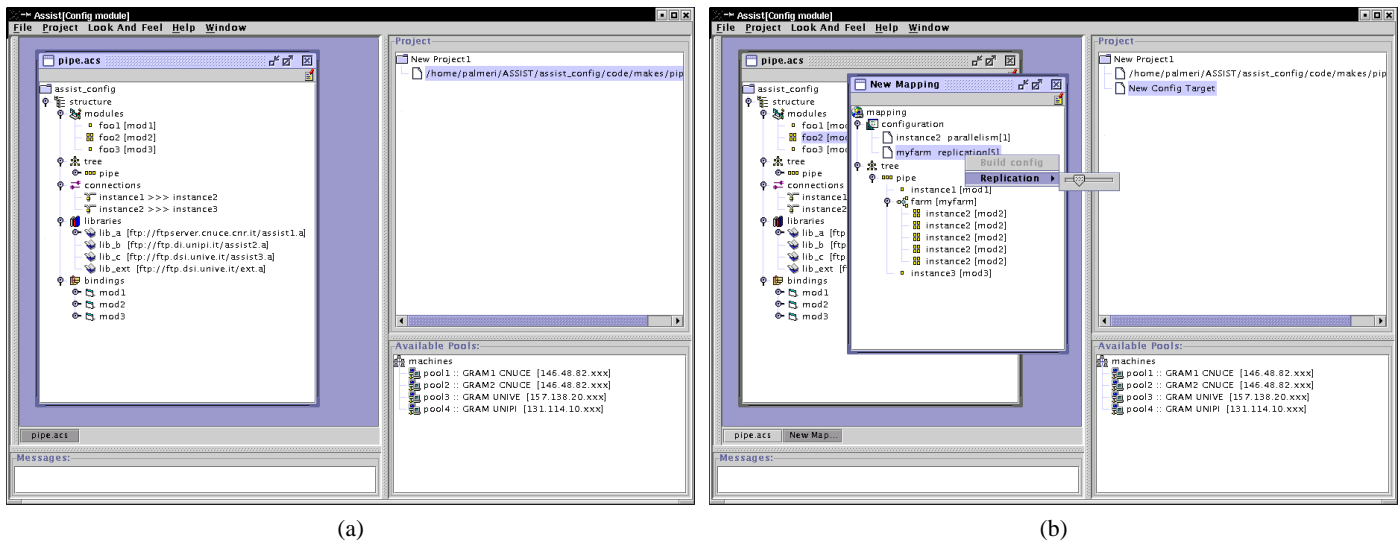


Figure 5. AssistConf windows. (a) Main window. A new project is created from the configuration file generated by the ASSIST compiler. This file contains only the structure part of the configuration, while the other two are built in a semi-automatic way using the AssistConf GUI. (b) Configuration window. The parallel modules of the application can be configured in terms of the parallelism degree of the parmods and the replication degree of the farm workers. Modifications are visualized accordingly.

view of the XML file produced by the ASSIST compilation phase, and an easy identification of the machines to be used for the application execution.

Figure 5.(a) depicts the AssistConf main window. A message area is shown at the bottom left, in which errors and information messages are displayed. The Project and Available Pools areas display, respectively, the files related to the configuration under development and the machine pools available to run the Grid program. In the current AssistConf version, the information describing the machine pools are gathered by accessing a *machine file*, where the IP addresses of the different resources are listed. Future implementations of AssistConf will provide interaction with the Grid Information Service (e.g. the Globus MDS) to obtain information about available resources and their characteristics.

In order to configure an ASSIST-CL program, a project has to be created, by opening the related XML ASSIST configuration file. As already explained in Section 3, the structure section of the configuration file is produced by the compiler. We refer to this first version of the XML file, as the *ASSISTConfiguration Source* (.acs extension).

In Figure 5.(a) the file corresponding to the ASSIST-CL program of Figure 1 is shown (Example.acs). The XML tree is displayed according to the specifications contained in the associated DTD file. Note that only the structure section is present at this level.

The ASSIST-CL program is composed of a pipe of three stages: mod1 (sequential), mod2 (parallel) and mod3 (sequential). All the elements like connections, libraries and bindings, which are contained in the structure section, are represented, but none of them can be modified since this section is static.

In order to add the other two sections of the configuration file, a "New mapping" has to be created. This is displayed in a new window only containing the configuration section and a replication of the *tree* section (i.e., a subsection of the structure section) showing the ASSIST modules and their nesting. As the parallel modules are configured (Figure 5.(b)), the tree section is changed accordingly to reflect the number of instances of a replicated module or the parallelism degree of a parmod.

The final step is to establish a mapping between the program modules and the machines in the Grid. This task is accomplished by activating the pool selection menu (Figure 6.(a)). If a pool is assigned to an intermediate node in the tree, all the leaves will inherit the same pool.

Once all the modules have been assigned to the machines, the configuration is complete and we can generate the final configuration file, whose graphical representation is given in Figure 6.(b). This file is called *ASSISTConfiguration Target* (.act extension), and together with the corresponding .acs file, form an *ASSISTConfiguration Project* (.acp extension). More than

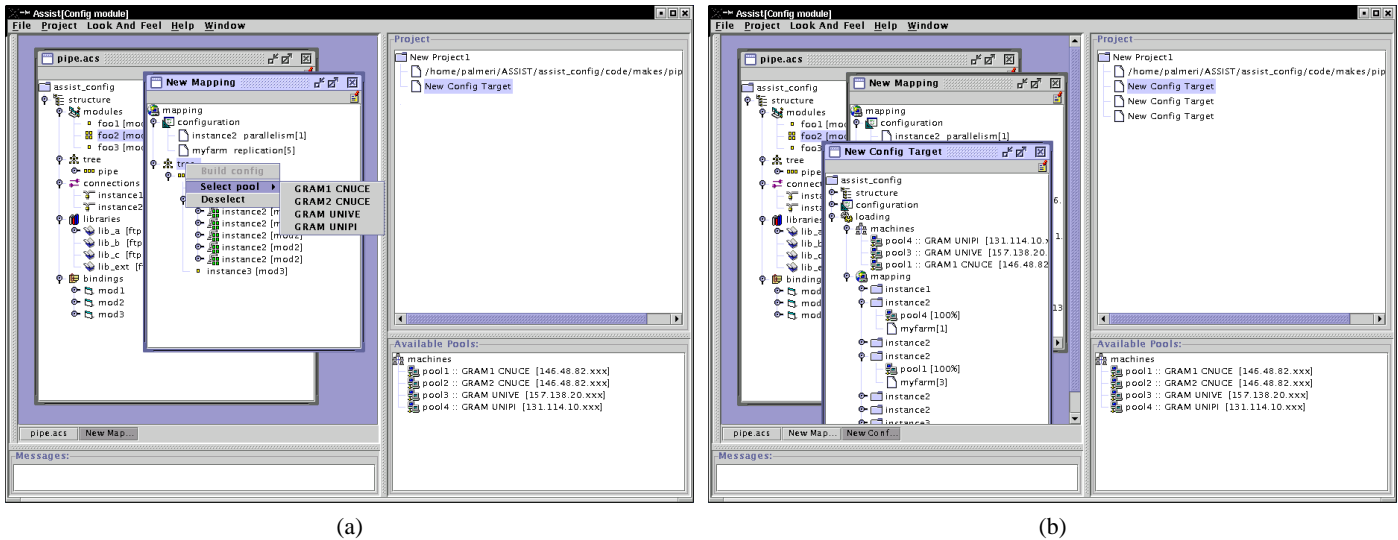


Figure 6. AssistConf windows. (a) Mapping window. Machines can be assigned to the modules by selecting among a set of available resources. (b) Once the parallelism degree and the mapping are completed, the final configuration can be built. There can exist more than one mapping for the same source, within the same project.

one `.act` file can be generated for the same `.acs` file, but still within the same project, i.e. the same `.acp` file. This corresponds to having more mappings for the same application, which might be useful for evaluating different choices for variables like the parallelism degree or the resources used.

The configuration file produced by AssistConf (`.act`) can be used as input to the `assistrun` command, which will drive the execution of the ASSIST program on the Grid.

5 Conclusions

In this paper we have presented AssistConf, a graphical user interface designed for the manual searching and selection of suitable Grid resources candidate to execute an ASSIST application. This tool mainly aims at simplifying the creation of the ASSIST configuration file, giving users a graphical view of the XML file produced by the ASSIST compilation phase, and permitting an easy identification of the machines to be used for the application execution. Finally, the configuration file produced by AssistConf is used as input to the `assistrun` command, which drives the execution of the ASSIST program on the Grid. AssistConf is designed to be used as an independent tool to assist the user to establish a mapping between the ASSIST program modules and the most suitable machines in the Grid candidate to execute them. In the future, it is our intention to fully integrate it into the ASSIST development environment.

In the current AssistConf version, the information de-

scribing the machine pools are gathered by accessing a machine file, where the IP addresses of the different resources are listed. The next version of AssistConf will interact with the Grid Information Service (e.g. the Globus MDS [7]) to obtain information about the available resources and their characteristics.

We are aware that the present version of AssistConf just represents a first step towards the design of an advanced resource selection and configuration tool. In our vision, the tool will have to support users in a higher level way: users will have only to specify the mapping requirements of the various components of their program, by asking for machines with given characteristics in terms of computational power, disk space, memory, etc. A Grid *broker* will then determine the actual mapping and produce the final XML ASSIST configuration file, by choosing the most suitable machines available in the Grid on the basis of information supplied by the Grid Information Service. In order to achieve this goal we are studying the feasibility of the integration of AssistConf with the Grid Resource Broker (GRB) [1], a tool for Grid resource discovery and selection developed at the Department of Innovation Engineering at the University of Lecce (Italy).

References

- [1] G. Aloisio, M. Cafaro, I. Epicoco, and S. Fiore. Grid Resource Broker (GRB). In <http://sara.unile.it/grb/grb.html>.

- [2] Angulo, D. and Foster, I. and Liu, C. and Yang, L. Design and Evaluation of a Resource Selection Framework for Grid Applications. In <http://www.globus.org/research/papers.html>, 2002.
- [3] B. Bacci, M. Danelutto, S. Pelagatti, S. Orlando, and M. Vanneschi. P3L: a Structured High-level Parallel Language and its Structured Support. *Concurrency: Practice and Experience*, 7(3), 1999.
- [4] B. Bacci, M. Danelutto, S. Pelagatti, and M. Vanneschi. SkIE : A heterogeneous environment for HPC applications. *Parallel Computing*, 25, 1999.
- [5] M. Cole. *Algorithmic skeletons: structured management of parallel computation*. MIT Press, 1989.
- [6] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.
- [7] Czajkowski, K. and Fitzgerald, S. and Foster, I. and Kesselman, C. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, IEEE Press, August 2001, 2001.
- [8] F. Darema. Next Generation Software Research Directions. In <http://www.cise.nsf.gov/eia/NGS-slides/sld001.htm>, 2001.
- [9] J. Darlington, Y. Guo, H. W. To, and Y. Jing. Skeletons for structured parallel composition. In *Proc. of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1995.
- [10] C. K. e. I. Foster. *The Grid: Blueprint for a future computing infrastructure*. Morgan Kaufmann, 1999.
- [11] D. Laforenza. Grid Programming: Some Indications Where We Are Headed. *To be published on Parallel Computing*, North-Holland Elsevier, 2002.
- [12] S. Melody, J. Schopf, and Z. X. Grid Searcher. In <http://people.cs.uchicago.edu/hai/GridSearcher/overview.html>, 2002.
- [13] N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field, J. Darlington. An Integrated Grid Environment for Component Applications. In *Proceedings Grid Computing - Grid 2001, Second International Workshop, Denver 2001, LNCS Vol. 2242*.
- [14] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL*.
- [15] D. C. Schmidt. The ADAPTIVE Communication Environment: Object-Oriented Network Programming Components for Developing Client/Server Applications. In *11th and 12th Sun Users Group Conference*, 1994.
- [16] M. Vanneschi. Programming Model of ASSIST, an Environment for Parallel and Distributed Portable Application. *To be published on Parallel Computing*, North-Holland Elsevier, 2002.