85-72
2002

Documento n. WP5-005
Edizione 1
Data: 12 settembre 2002

PROGETTO:	ASI-PQE2000

TITOLO DOCUMENTO:	Assistconf: uno strumento interattivo per la configurazione e il caricamento di programmi ASSIST

ISTI
BIBLIOTECA
Collloc. ARCHIVIO

Totale Pagine: 30

WP 5	PAR-GRID: PPE INTEGRATO IN CONTESTO PARALLELO + GRID	DATA	Firma
Autori:	**ISTI-CNR, Pisa:** R. Baraglia, R. Ferrini, F. Furfari, D. Laforenza, S. Orlando, P. Palmerini, R. Perego **Dip. Informatica, Università di Pisa:** M. Danelutto, P. Pesciullesi, M. Vanneschi	9 settembre 2002	

		Data	Firma
Autorizzato Da	Marco Vanneschi Coordinatore Area 1	12 Settembre 2002	

Storia del Documento

ED.	DATA	AUTORI	REVISIONE	APPROVAZ.	AUTORIZZ.	MOTIVO DELLE MODIFICHE
1		R. Baraglia, M. Danelutto, R. Ferrini, F. Furfari, D. Laforenza, S. Orlando, P. Palmerini, P. Pesciullesi, R. Perego, M. Vanneschi				

Registrazione delle Modifiche

ED.	PAGINA	SEZIONE	MODIFICA (dell'edizione attuale rispetto alla precedente)

Lista di Distribuzione

ENTE	NOME	FUNZIONE	N. DI COPIE
Agenzia Spaziale Italiana	Rosa Loizzo	Membro Comitato di Gestione	1
Agenzia Spaziale Italiana	Fabrizio Micolitti	Membro Comitato di Gestione	1
Agenzia Spaziale Italiana	Giovanni Milillo	Coordinatore Comitato di Gestione e Coordinatore Area 2	1
Università di Perugia	Antonio Sgamellotti	Membro Comitato di Gestione	1
Università di Pisa	Marco Vanneschi	Membro Comitato di Gestione e Coordinatore Area 1	1

INDICE

1. PREMESSA

1.1. Scopo del documento

Il presente documento presenta l'interfaccia grafica e l'architettura software di *Assistconf*, uno strumento *interattivo* di configurazione e caricamento per programmi ASSIST. Il documento è così organizzato: la Sezione 3 richiama il processo di compilazione di un programma ASSIST, mentre la Sezione 4 riassume le caratteristiche salienti del file di configurazione XML per la CLAM. Infine, la Sezione 5 illustra lo strumento Assistconf, che permette di specificare in dettaglio il file XML finale per l'esecuzione di un programma ASSIST-CL. Viene fornito un semplice manuale d'uso dello strumento, e una descrizione ad alto livello del progetto software e della sua configurazione. Maggiore documentazione viene fornita in formato elettronico nel CD allegato.

1.2. Applicabilità

Il presente documento si applica al Progetto ASI-PQE2000 "Sviluppo di applicazioni di Osservazione della Terra mediante sistemi e strumenti di Calcolo ad Alte Prestazioni", Area 1 : Ambiente di Programmazione.

1.3. Definizioni, Acronimi e Abbreviazioni

ASI	Agenzia Spaziale Italiana
ASSIST	A Software development System based on Integrated Skeleton Technology
CLAM	Coordination Language Abstract Machine
COW	Cluster di Workstation
DTD	Document Type Definition
GIIS	Grid Index Information Service
GIS	Grid Information Service
GRAM	Globus Resource Administration Manager
GRIS	Globus Resource Informazion Service
LDAP	Lightway Directory Access Protocol
JAXP	Java API for XML Parsing
JDOM	Java Document Object Model
MDS	Metacomputer Directory Service
XML	Extensible Markup Language
ASSISTCONF	Strumento di configurazione e mapping sviluppato nel WP5
ASSISTRUN	Strumento per il run di programmi ASSIST su cluster o Grid

2. Riferimenti

2.1. Documenti Applicabili

	Document Code	Document Title
[A1]		Progetto ASI-PQE2000: Allegato Tecnico e di Gestione, 1 marzo 2001

2.2. Documenti di Riferimento

	Document Title
[1]	Foster I. and Kesselman C. (editors), ***The Grid: Blueprint for a Future Computing Infrastructure***, Morgan Kaufmann Publishers, USA, 1999.
[2]	*ASSIST Resource Management in Ambiente Grid*, Progetto ASI-PQE, Deliverable WP005_003, 30 ottobre 2001
[3]	*Configurazione e mapping di programmi ASSIST in ambiente Grid: File di configurazione e File di Assistconf*, Progetto ASI-PQE, Deliverable WP005_004, 7 gennaio 2002
[4]	http://www.qarbon.com/
[5]	http://java.sun.com/
[6]	"Java and XML" By Brett McLaughlin, June 2000 O'Reilly
[7]	http://java.sun.com/j2se/1.3/
[8]	http://java.sun.com/j2se/1.4/
[9]	http://xml.apache.org/
[10]	"JavaTM Technology and XML Part 2: API Benchmarks" By Thierry Violleau, March 2002 (http://developer.java.sun.com/developer/technicalArticles/xml/JavaTechandXML_part2/) Java Developer Connection
[11]	http://java.sun.com/xml/jaxp/
[12]	http://www.jdom.org/
[13]	"Design Patterns Elements of Reusable Object-Oriented Software" by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides October 1994 ,416 pages, Addison-Wesley ISBN 0-201-63361-2.
[14]	"Create a scrollable virtual desktop in Swing" Tom Tessier, (2,000 words; November 30, 2001) JavaWorld column

3. Introduzione

Come descritto in altri rapporti tecnici del Progetto ASI-PQE2000 [3, 4], durante il processo di compilazione ASSIST vengono generati:

- Moduli software, prodotti come librerie dinamiche, caricabili e eseguibili da parte dei LOADER CLAM;
- Informazioni sulla struttura del programma ASSIST;
- Informazioni sui moduli (nomi, *pathname*, ecc.).

Tale processo è illustrato in Figura 1:

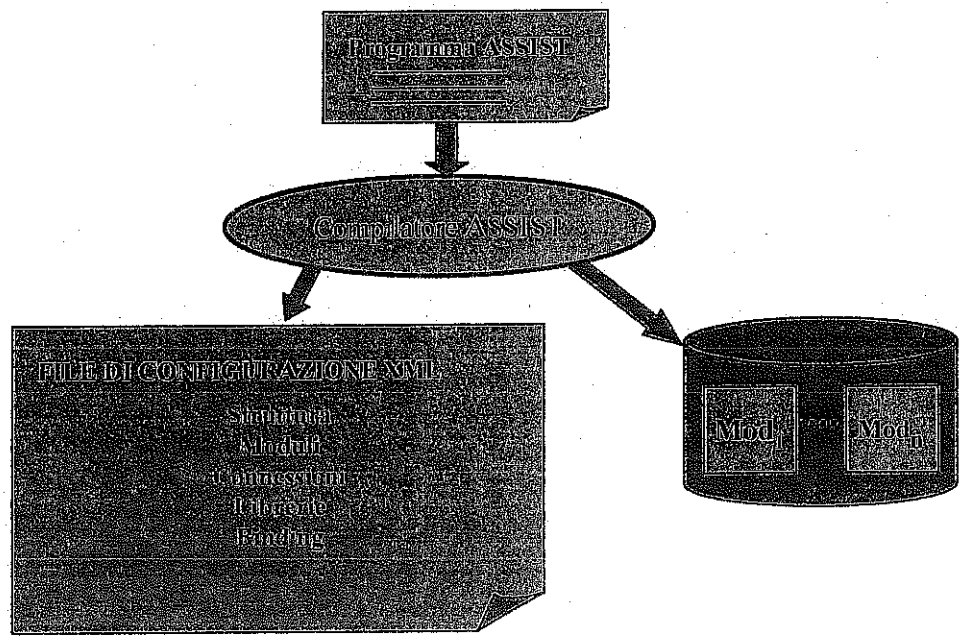


Figura 1: File prodotti durante il processo di compilazione di un programma ASSIST

Le informazioni sulla struttura del programma e sui moduli (nomi, *pathname*, ecc.) sono prodotti sotto forma di file XML [4]. Tale file deve essere `completato` con informazioni di *configurazione/mapping* per poter essere dato in input al comando **`Assistrun`** per l'esecuzione del programma.

Come illustrato in Figura 2, il processo di configurazione del programma è svolto interattivamente mediante uno strumento, appositamente progettato ed implementato, denominato **`Assistconf`**. `Assistconf` permette di estendere il file di configurazione XML mostrato in Figura 1, mediante l'aggiunta di ulteriori informazioni concernenti:

- la configurazione del programma;
- il mapping delle istanze dei moduli, eventualmente replicati/parallelizzati.

Il **file di configurazione CLAM**, ovvero il file XML completato con le estensioni prodotte tramite `Assistconf` è dato in input ad `Assistrun`, che si occupa dell'eventuale movimento dei moduli eseguibili (*staging*), del caricamento (*loading*) degli stessi, e dell'avvio della esecuzione del programma (*run*).

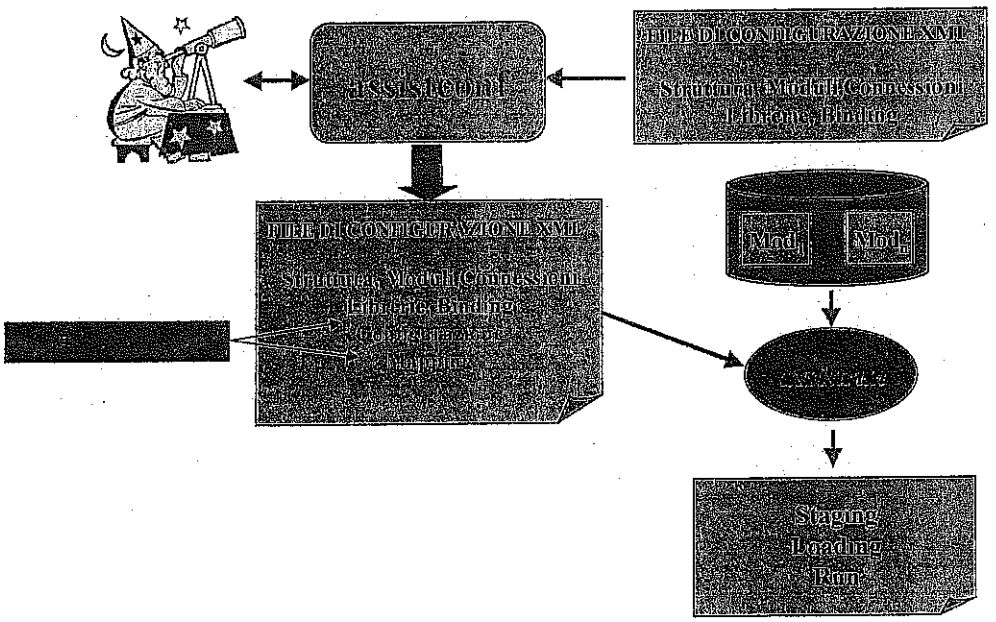


Figura 2: Processo di configurazione di un programma ASSIST mediante `Assistconf`

4. Specifiche del file di configurazione

Il file di configurazione XML di un generico programma ASSIST-CL è così strutturato:

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

Per una trattazione dettagliata di tale file di configurazione rimandiamo ad altri rapporti tecnici del Progetto ASI-PQE2000 [4]. Per gli scopi di questo rapporto è sufficiente riassumere a grandi linee il significato delle sezioni in cui il file di configurazione è strutturato. Il prologo identifica il formato XML del file di configurazione. Nella seconda riga viene definito l'URI (Uniform Resource Identifiers) del Document Type Definition (DTD) relativo, ovvero del file contenente la

"grammatica" del file XML. L'elemento `assist_config`, definito nel DTD, è scelto come *radice* del documento XML.

Nel DTD l'elemento **`assist_config`** viene definito come contenente al suo interno tre elementi, dove ciascun elemento corrisponde a una *sezione* distinta del file di configurazione:

```
<!ELEMENT assist_config (structure, configuration?, loading?) >
```

La prima sezione, denominata "**structure**", contiene informazioni statiche sulla struttura del programma ASSIST e sulle librerie dinamiche (DLL) che implementano i vari moduli. La sezione è generata dal compilatore e non deve essere modificata dall'utente.

La seconda sezione, chiamata "**configuration**", contiene informazioni su parametri di configurazione del programma ASSIST. In particolare, in questa sezione si specifica il grado di parallelismo dei costrutti *parmod*, e il grado di replicazione dei costrutti annidati nei *farm*. Tale sezione è chiaramente modificabile e permette all'utente di configurare il numero di risorse di calcolo (logiche) necessarie all'esecuzione dell'applicazione. Inoltre questa sezione è opzionale, in quanto può essere inserita in un secondo tempo, a partire della sezione precedente e deve essere completata dallo strumento di configurazione e loading (`Assistconf`).

La terza sezione, denominata "**loading**", contiene informazioni per il caricamento e il mapping dei vari moduli componenti l'applicazione ASSIST descritti nelle sezioni precedenti. Tale sezione è anch'essa modificabile, e contiene dati che consentono:

- la selezione del pool di macchine su cui fare eseguire il programma ASSIST
- il mapping delle istanze dei moduli ASSIST, così come sono stati individuati nelle sezioni *structure* e *configuration*, su macchine specifiche appartenenti al pool.

Questa sezione è anch'essa opzionale, in quanto dovrà essere inserita in un secondo tempo, sulla base di entrambe le sezioni precedenti, e dovrà essere completata mediante `Assistconf`.

5. AssistConf: un tool per la generazione semi-automatica del file di configurazione

Le specifica formale del DTD del file di configurazione (§4) ha delineato i *requirements* di uno strumento semi-automatico per il mapping dei moduli di un programma Assist.

Nella fase progettuale l'attenzione si è focalizzata principalmente sulla necessità di fornire una interfaccia grafica abbastanza semplice con la quale visualizzare il contenuto dei file di configurazione (provenienti dal *compilatore* ASSIST-CL) e consentire la loro manipolazione. Lo scopo è di variare un certo numero di parametri e aggiungere le informazioni per il la configurazione di alcuni costrutti ASSIST-CL, e il mapping delle istanze dei moduli.

In particolare i parametri presi in considerazione sono:

- il **grado di parallelismo** dei moduli paralleli;
- il **grado di replicazione** dei *worker* di un farm;
- il ***pool*** di macchine sui cui allocare le varie istanze di moduli;
- le **librerie Assist** da allocare sui moduli replicati.

L'esigenza di sperimentare più configurazioni per la medesima computazione ha indotto ad introdurre il *concetto di progetto* nelle specifiche di realizzazione del modulo. Il *progetto* (file **.acp**) raggruppa, perciò, tutte le risorse utilizzate durante la fase di *mapping* delle istanze. In particola re distinguiamo tre tipi di risorse:

- file *assist config source* (**.acs**)
- file *assist config mapping* (**.acm**)
- file *assist config target* (**.act**)

Un file *.acs* rappresenta il file XML generato dal *compilatore* ASSIST-CL. Tipicamente il file sarà privo dei sottoalberi *configuration* e *mapping*. Un file *.acm* è un file temporaneo generato dal modulo AssistConf per gestire il *mapping* effettuato dall'utente. Infine un file *.act* è il file di configurazione *xml* finale generato in corrispondeza del mapping effettuato e rappresenta *l'input* per il modulo *Assistrun*.

L'utilizzo tipico del modulo Assistconf sarà quello di associare uno o più file *.act* ad un unico file di configurazione iniziale (*.acs*), anche se attualmente è possibile inserire in un progetto più file di configurazione *.acs*. Ad ogni modo, il concetto di progetto coniugato con lo sviluppo di adeguati componenti grafici potrebbe consentire in futuro la composizione e manipolazione dei diversi file *.acs* per generare nuove computazioni in modo modulare.
In aggiunta alla modalità grafica il modulo AssitConfig può funzionare in *quiet mode* mappando la computazione su un unico *pool* di *default*. Tale modalità potrà esser utile per velocizzare le fasi di *test* e *debug* del sistema. La sintassi è la seguente:

```
java it.cnr.AssistConf [<source> [<machine> [<filename>]]]
```

`<source>` -- *source file (.acs)*
`<machine>` -- *nome della macchina su cui mappare l'intera computazione*
`<filename>` -- eventuale filename da utilizzare per l'output (.act)

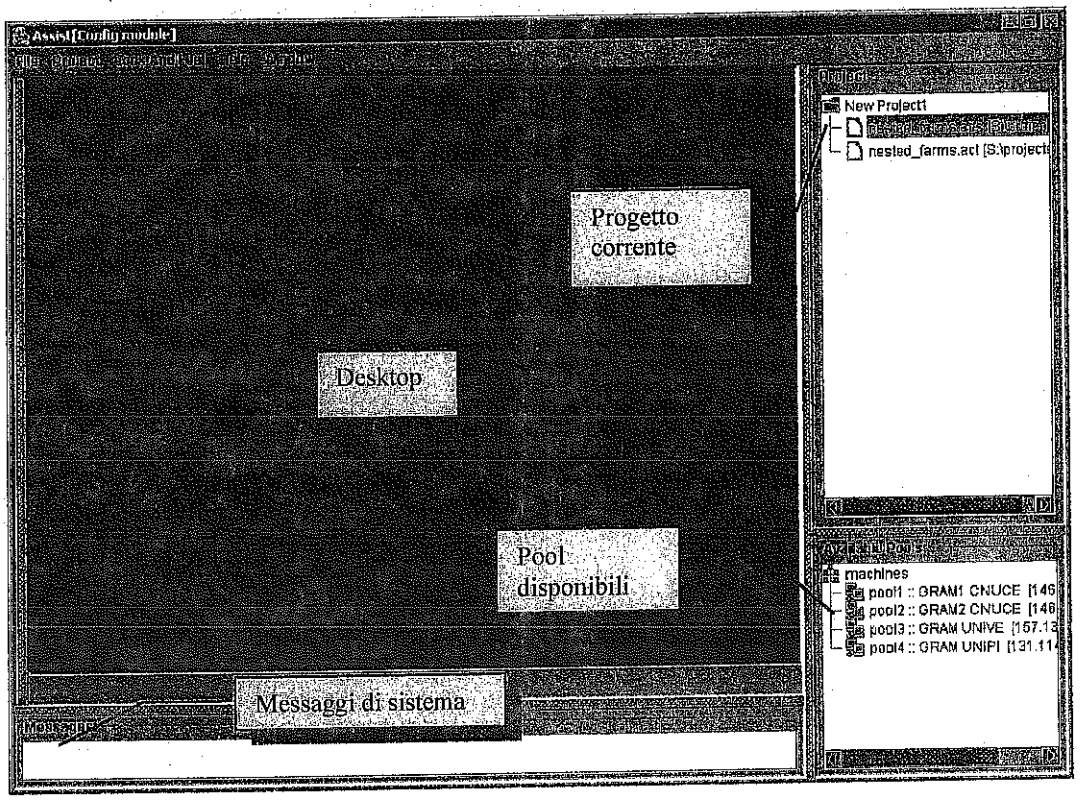
Nel prossimo paragrafo si descrive brevemente l'*interfaccia grafica* del modulo. Seguirà una descrizione dell'architettura del modulo, delle librerie utilizzate e delle procedure da seguire per l'installazione ed esecuzione del modulo.

User Interface

Il *layout* della *user interface* è suddiviso in 4 zone, come illustrato nello *screenshot* seguente.
L'area in alto a destra contiene un pannello che fornisce una vista del progetto corrente (***project viewer***). Il pannello mostra tutti i file di configurazione inseriti nel progetto (*.acs*) e i file di *mapping* generati dall'utente (*.act*). Il contenuto del pannello è manipolabile tramite il menu "`Project`" oppure con un menu contestuale richiamabile con il bottone destro del mouse. Le operazioni consentite sono :

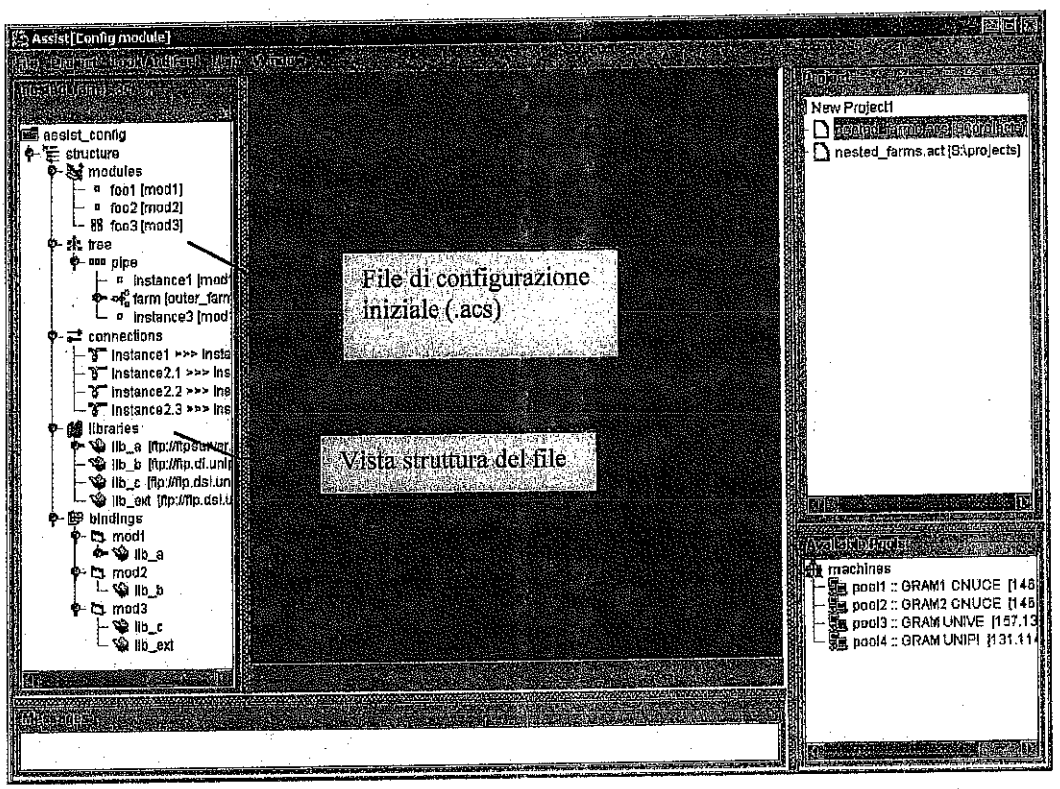
- `new`
- `open`
- `save`
- `close`
- `insert`
- `remove`
- `view structure`
- `view source`

Le prime operazioni servono a creare e gestire un progetto, `insert` e `remove` per aggiungere i file di configurazione proveniente dal *front-end* ed infine `view structure/source` consentono di vedere una rappresentazione strutturata o testuale dei file.
Al di sotto del *project viewer* è posto un pannello che mostra la lista dei pool attualmente disponobili (***pool viewer***). Attualmente la lista dei pool disponibili è ottenuta leggendo un file di configurazione: `DummyMachines.xml` . In futuro nuove funzionalità saranno aggiunte per ottenere una lista di pool aggiornata in tempo reale.

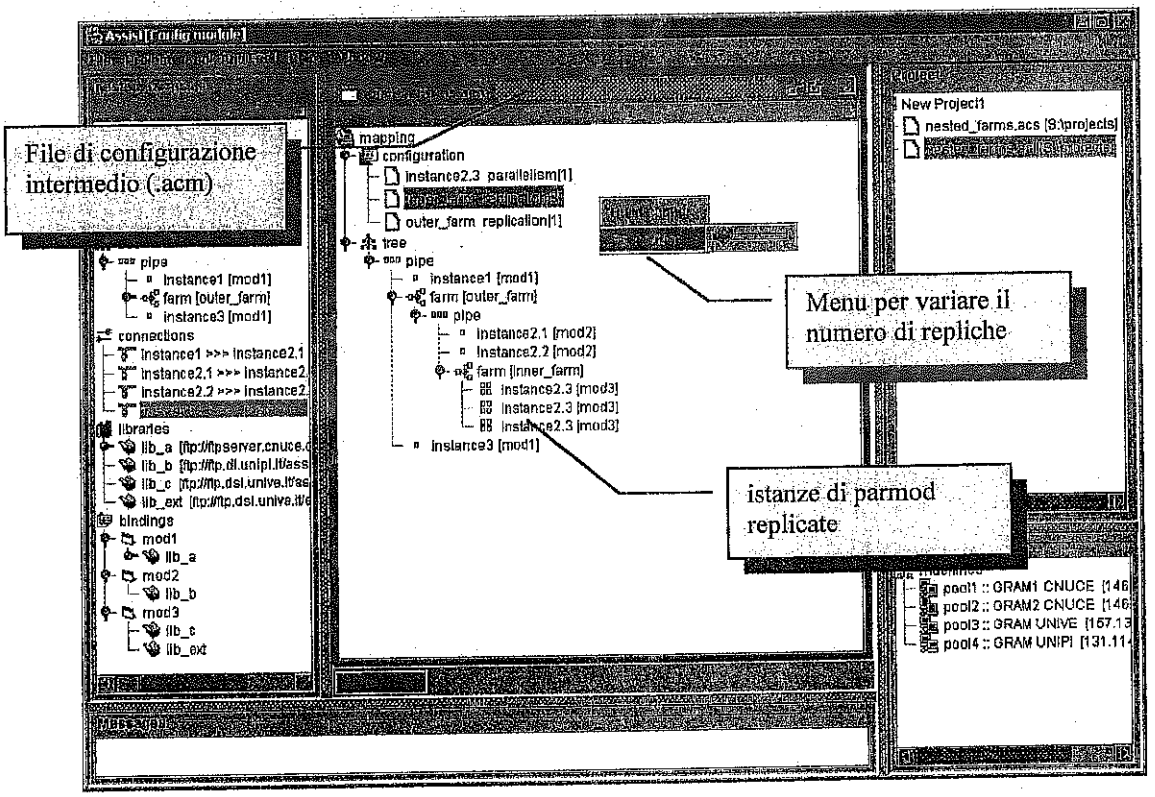


L'area in basso fornisce una visualizzazione dei messaggi di sistema (***message viewer***), ad esempio in seguito ad una validazione fallita, oppure alla impossibilità di raggiungere/connettersi con un pool di macchine.
L'area centrale è occupata dal ***desktop*** nel quale sono visualizzate le finestre releative ai diversi *file* di configurazione. Il posizionamento di tali finestre è regolabile dal menu "`Window`" o dalla barra di stato.

Il processo di *mapping* è iniziato dalla finestra del file di configurazione iniziale (*.acs*) la cui struttura è mostrata nello *screenshot* seguente:



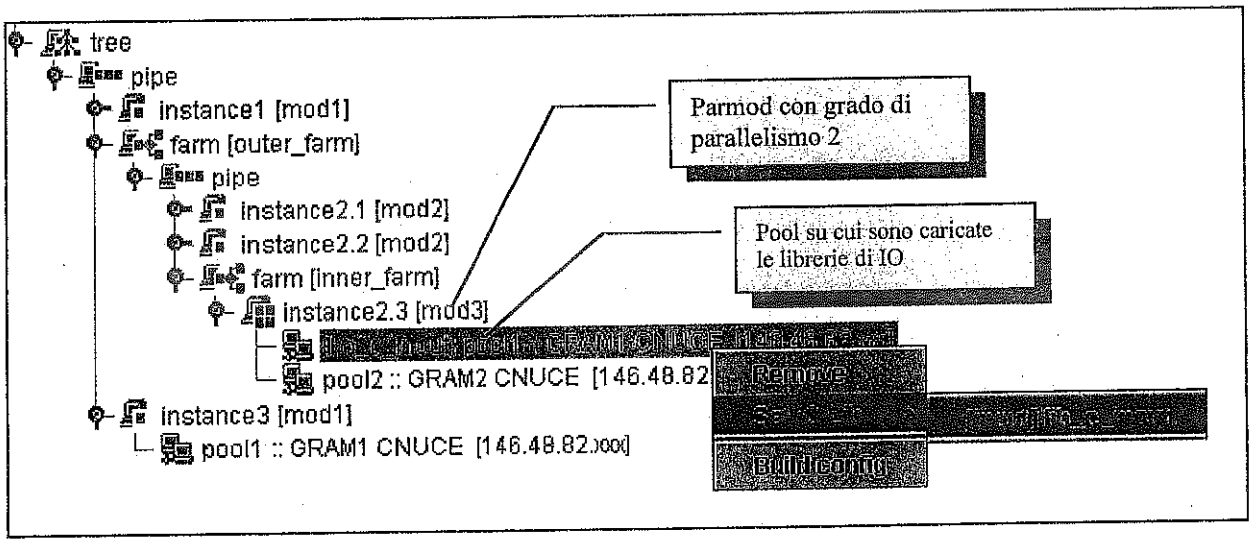
Tramite il menu contestuale "`New mapping`" si crea una nuova finestra (mostrata di seguito) per ogni *mapping* che si vuole effettuare, crendo un file intermedio di configurazione e mapping (*.acm*).



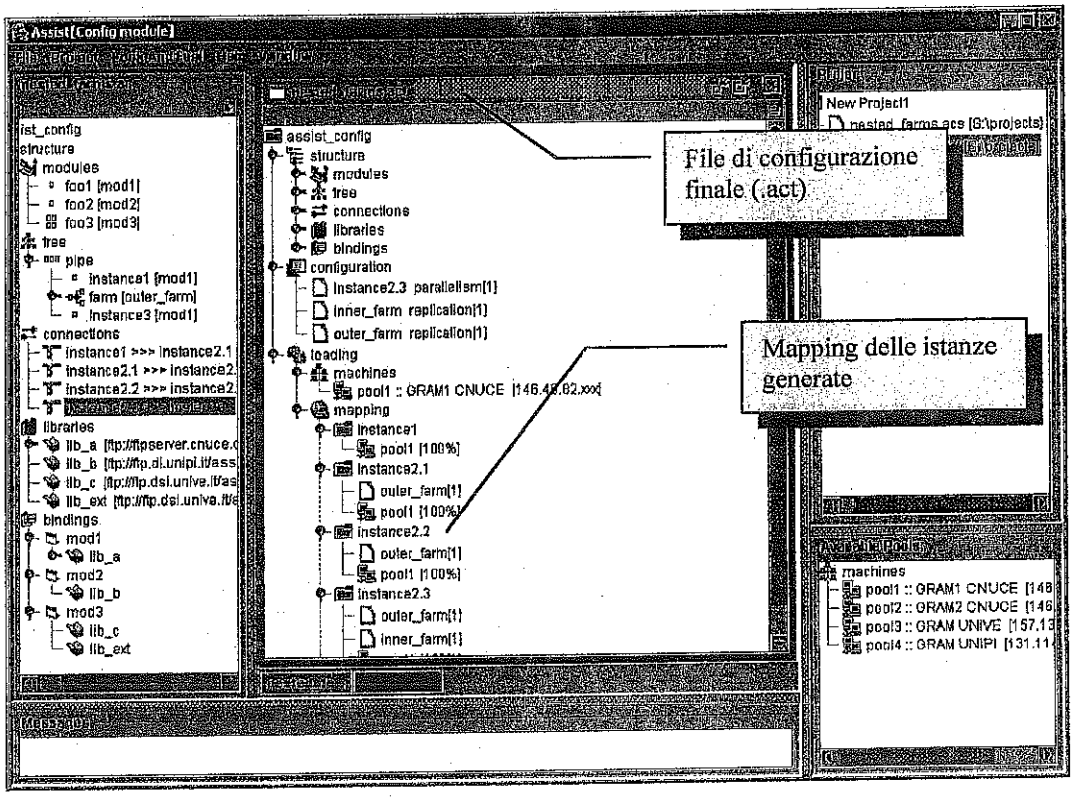
Le operazioni principali per completare la configurazione ed il mapping si effettuano agendo su quest'ultima finestra, che mostra due sottoalberi del file di configurazione: `configuration` e `tree`. Per mezzo di menu contestuali associati ai nodi del sottoalbero *configuration* si può variare il grado di parallelismo o di replicazione dei moduli. Nel sottoalbero *tree* il numero di istanze da mappare varierà in accordo alle modifiche apportate nel sottoalbero di configurazione. Ogni nodo del sottoalbero è mappabile su uno specifico *pool* sempre tramite un menu contestuale. Per semplificare le operazioni di *mapping*, la selezione di *pool* per uno specifico nodo è ereditata da tutti i nodi del suo sottoalbero, per questo motivo per effettuare il mapping conviene adottare un approccio *top-down*. Soltanto quando tutti i nodi dell'albero sono stati mappati può esser creato il file di configurazione finale. Graficamente l'icona che rappresenta la radice della computazione cambia stato e il menu contestuale "Build Config" viene abilitato. Di seguito è riportata la tabella dei nodi di una computazione e delle icone associate al loro stato (*Mappato* e *Non mappato*).

NODI	Non mappato	Mappato
Tree		
Pipe		
Farm		
Parmod		
Mod (sequenziale)		

In particolare per in nodi di tipo *Parmod* bisogna selezionare tante macchine quante ne richiede il grado di parallelismo specificato e inoltre, sempre tramite menu contestuale, indicare su quale macchina caricare le librerie di IO.
La figura seguente mostra lo stato di un nodo di tipo Parmod con grado di parallelismo 2.



Infine, lo *screenshot* seguente mostra il file di configurazione competo (*.act*) che contiene il mapping completo. Si possono notare le nuove sezioni XML *configuration*, *loading/machines* e *loading/mapping* aggiunte al file iniziale (*.acs*).

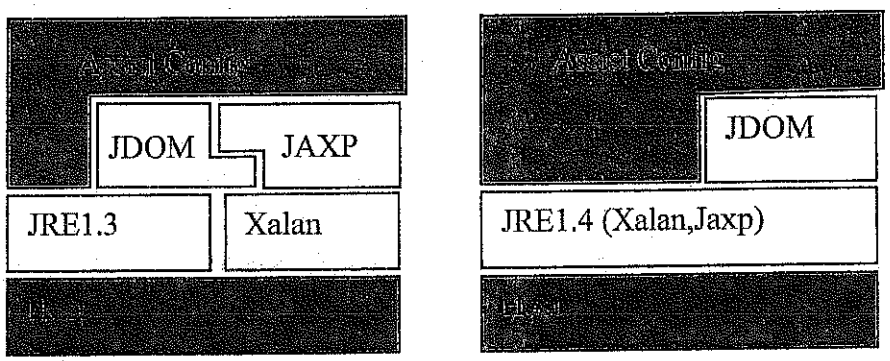


Configurazione ed architettura del software

Le esigenze di portabilità del codice hanno spinto verso la scelta di Java [5] come piattaforma di riferimento per lo sviluppo del codice. Il binomio Java e XML [6] ampiamente riconosciuto come sinonimo di portabilità di codice "+" dati ha ulteriormente giustificato l'utilizzo di Java per la vasta disponibilità di librerie per la manipolazione di XML e standard correlati.

Il software è stato inizialmente sviluppato basandosi sulla versione 1.3 del Java Development Kit (j2se1.3) [7], e in fase di sviluppo si è verificato la compatibilità con la nuova *release* della piattaforma Java [8] Per quanto riguarda le librerie (*parser*) per XML le più diffuse e testate in ambiente Java sono la Xerces, Crimson e Xalan tutte originate da progetti della fondazione Apache [9]. Pur non risultando la più performante [10], si è optato per la libreria Xalan perché indicata come la probabile candidata ad essere inclusa nella successiva release di Java (JRE1.4). Altre librerie utilizzate sono JAXP [11] e JDOM [12]. JAXP inserisce un livello di indirizzamento tra applicativi e *parser* XML garantendo una maggiore indipendenza degli applicativi dagli specifici *parser*. La libreria progettata da Sun consiste in una serie di interfacce per accedere alle *factory class* [13] che istanziano i *parser*. Lo svantaggio di tali interfacce è che per l'accesso e navigazione del *Dom* di un documento XML bisogna usare classi di iterazione (*iterator*) non standard. La

libreria JDOM evita questi problemi basandasi sul framework di rifermento della piattaforma Java (`Collection` framework).



Il codice java dell'applicativo è contenuto nel package ***it.cnr.assit.config***. La classe preposta al lancio dell'applicativo è la class *it.cnr.AssistConf*. Oltre alle librerie sopra citate il codice dipende dal file di configurazione:

- `assist.settings.txt`
  (file::/<AssistRoot>/**classes**/it/cnr/assist/config/settings/assist.settings.txt)
- `program.settings.txt`
  (file::/<AssistRoot>/**classes**/it/cnr/assist/config/settings/program.settings.txt)
- `DummyMachines.xml`
  (file::/<AssistRoot>/**classes**/it/cnr/assist/config/machines/DummyMachines.xml

e dalle risorse contenute nella *directory* : <AssistRoot>/**classes**/it/cnr/assist/config/images.

Il file di *setting* consente un minimo adeguamento del logica dell'applicativo ad eventuali cambiamenti del DTD. Operazioni come il cambio dei nomi degli elementi definiti nel DTD o anche la modifica di sottoalberi non strettamente coinvolti nella logica dell'applicativo possono essere apportate senza la necessità di rigenerare il codice.

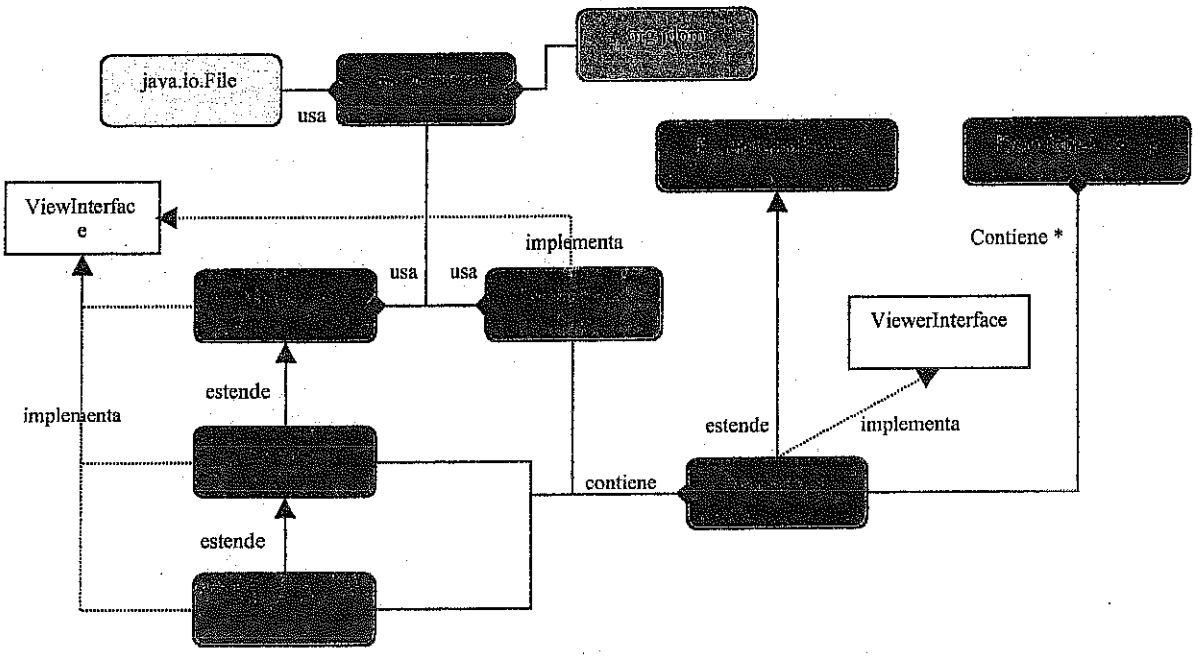
L'interazione tra i *package* che compongono l'applicativo è garantita dalla classe `Sys` che svolge il ruolo di *mediator*. Tramite i metodi di `Sys` è possibile accedere ai vari moduli dell'applicativo (implementati come *singleton*).

I principali moduli sono:

- `ProgramManager`
- `WindowManager`
- `ProjectManager`
- `PoolManager;`

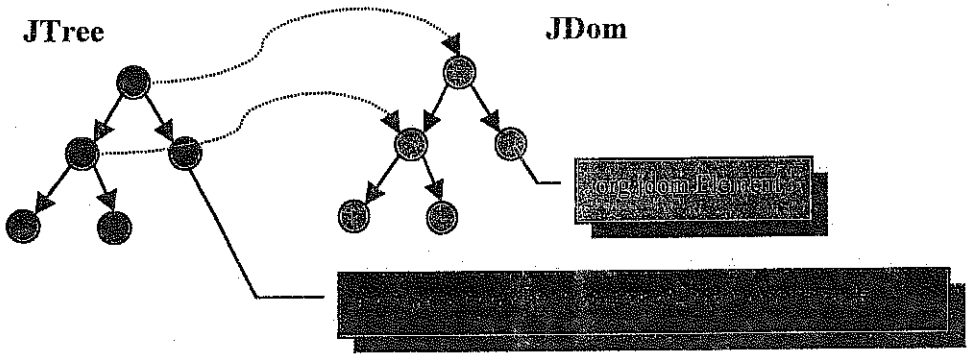
Il `ProgramManager`, invocato all'avvio dalla classe di lancio, è il modulo preposto alla costruzione del *layout* grafico.
Il `WindowManager` gestisce il desktop dell'applicativo. Essendo le classi `swing` per la gestione del *desktop* (`JDesktopPane`, `JInternalFrame`) piuttosto primitive si è utilizzato una

una libreria di dominio pubblico [12] che estende le funzionalità di base: `com.tomtessier.scrollabledesktop`.
Il `ProjectManager` è il modulo che realizza gran parte della logica dell'applicativo. Dalla creazione di nuovi progetti alla manipolazione dei file di configurazione fino alla generazione del *mapping*.
Le classi impiegate si trovano nel package `it.cnr.assist.config.document` e sono illustrate di seguito.



`XmlDocument` è una classe che incapsula il concetto di file e ne arrichisce le funzionalità per manipolare i file di tipo XML. Le operazioni di validazione e costruzione del *DOM* sono effettuate in questa classe base. Ad ogni documento XML possono essere associate più viste (*view*) e ciascuna vista è visualizzata tramite un *viewer* (tipicamente una finestra del *desktop*). Nel *package* sono presenti 4 tipologie di viste. `XmlView` è la vista di *default* per generici documenti *xml*. Tale vista è estesa dalla classe `ACView` (*AssitConfigView*) per visualizzare in modo appropriato i sottoalberi definiti nel DTD di Assist. A sua volta `ACView` è estesa da `MapView` e permette la concreta manipolazione del file di configurazione iniziale.

Ognuna di queste viste utilizza la classe swing `JTree` per la rappresentazione del *DOM* di Assist. `ACView` costruisce una struttura ad albero isomorfa al modello del documento attraverso la classe `AssistNode`. Pertanto il documento si può attraversare sia tramite le funzionalità di visita ereditate da `JTree` sia con i metodi definiti da `JDOM`.



Infine la vista `SourceView` fornisce una rappresentazione testuale del file di configurazione Assist. In tal modo variazioni non previste del DTD si potranno apportare direttamente sui file xml.

Ultimo modulo accessibile tramite la classe `Sys` è il `PoolManager`. Attualmente la configurazione dei *pool* disponibili è definita nel file di configurazione `DummyMachines.xml` posto nella directory: `<AssistRoot>/classes`.

6. Conclusioni e lavoro futuro

In questo documento, dopo aver introdotto alcune delle problematiche relative alla configurazione e al mapping di programmi ASSIST, è stata presentata la *user interface* e la relativa implementazione di uno strumento integrato nell'ambiente ASSIST, definito `Assistconf`. Lo scopo del strumento è quello di estendere il file XML, usato la configurazione della CLAM, per permettere l'esecuzione di un generico programma ASSIST-CL sia su un cluster di workstation che su una griglia computazionale.
Molto lavoro rimane ancora da fare per rendere completamente Grid-enabled l'ambiente ASSIST. In particolare, sarà necessario modificare l'attuale RTS e gli strumenti di caricamento (`assistrun`) per permettere una piena integrazione con la Grid, mentre nelle versioni successive dello strumento Assistconf verranno affrontate le problematiche per rendere più automatiche le scelte delle configurazioni dei programmi e delle risorse computazionali della Grid.

7. APPENDICE

7.1. **AssistConf: A Grid Configuration tool for the ASSIST parallel programming environment** (paper accepted for publication in the proceedings of the 11-th Euromicro Conference on Parallel Distributed and Network based Processing, PDP2003, Genoa, Italy, February, 5-7, 2003)

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

R. Baraglia[2], M. Danelutto[3], D. Laforenza[2], S. Orlando[1], P. Palmerini[1,2], P. Pesciullesi[3], R. Perego[2], M. Vanneschi[3]

[1] Dipartimento di Informatica, Università Ca' Foscari, Venezia, Italy
[2] Istituto CNUCE, Consiglio Nazionale delle Ricerche (CNR), Pisa, Italy
[3] Dipartimento di Informatica, Università di Pisa, 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 to 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 on the Grid.

1 Introduction

In the past, parallel applications were developed as *monolithic* entities, usually coded in a *single executable*. Nowdays 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 system, 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 trasparent 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 optimise 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 leterature 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 *Specification Language*(SL) [6,14].
- The SL-description is sent to a Grid Information Service (e.g., MDS [7]), which is responsible for mantaining 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 presenting 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 of the 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 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].

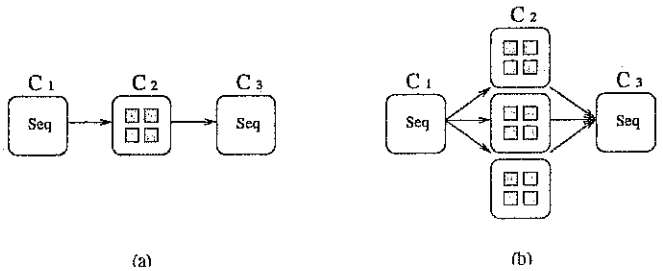


Figure1. An example of ASSIST application structured as a three stage pipeline where the second stage is a *parmod* (a), or a *farm* of *parmods* (b).

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 modules, 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 (ouput_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 be easily changed without affecting the other modules. Returning to the previuos 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 (ouput_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 run-time support, which exploit 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 will run on homogenous parallel machines and clusters (Linux), and it will contain also 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, to be loaded on the basis of an *XML configuration file* (see Figure 2). Therefore we can consider the CLAM-loaders as a sort of containers for the ASSIST-CL modules, where the contained components and the communication channels among them are specified through the configuration file. 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 introductive 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 which has to be elected as master. The master will become 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 section that specifies the binary modules produced by the compiler for a given ASSIST program, as well as the structure of the program: in particular, it determines the modules and the libraries implementing the various constructs composing the program, and the coordination streams connecting the various modules.
- A section that specifies the configuration of the program, i.e. degrees of parallelism and replication.
- A section that contains loading information, i.e. the specification of the mapping of module instances as described and configured in the previous sections.

In order to build the two last sections described above, 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 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, i.e. names, pathnames, etc., and information on the structure of the ASSIST program, i.e. parallel constructs and their hierarchy, their interconnections through streams, external libraries to be loaded, association of the ASSIST constructs with the files containing the implementation code.

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

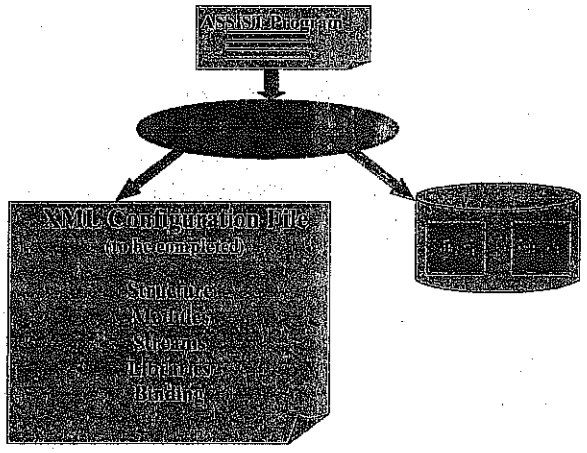


Figure2. Overview of the ASSIST compilation process.

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 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 generates 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 Grid chosen nodes.
- 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.

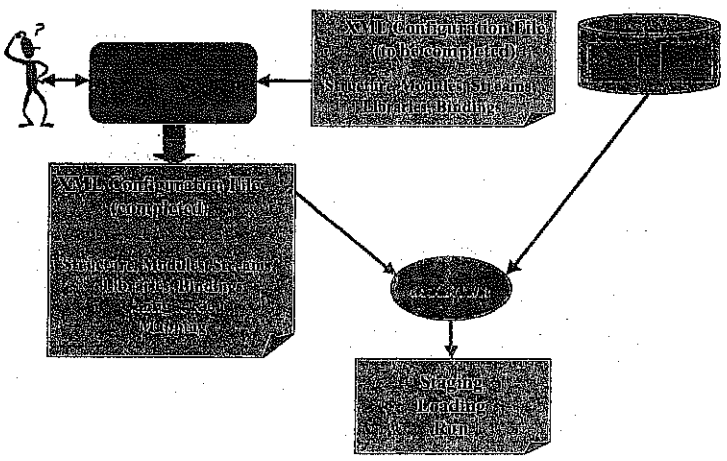


Figure3. Structure and interactions among the configuration, mapping, and execution phases.

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 a 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 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 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

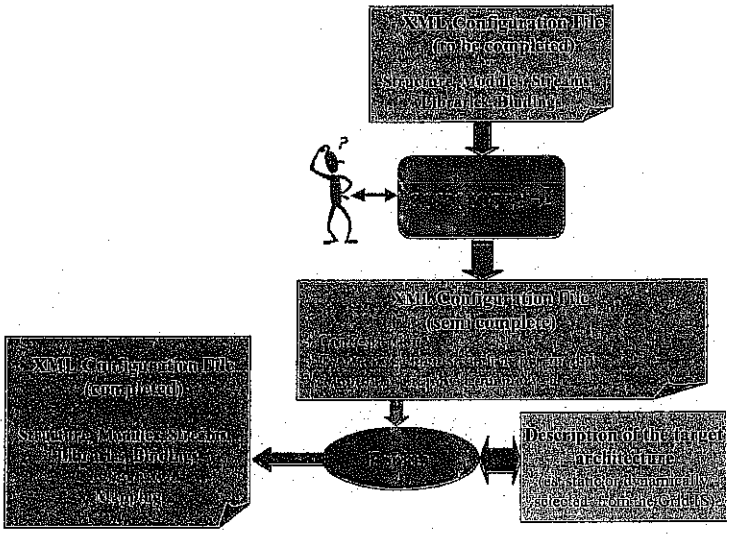


Figure4. Structure and interactions among the configuration, mapping, and execution phases with the enhanced version of AssistConf.

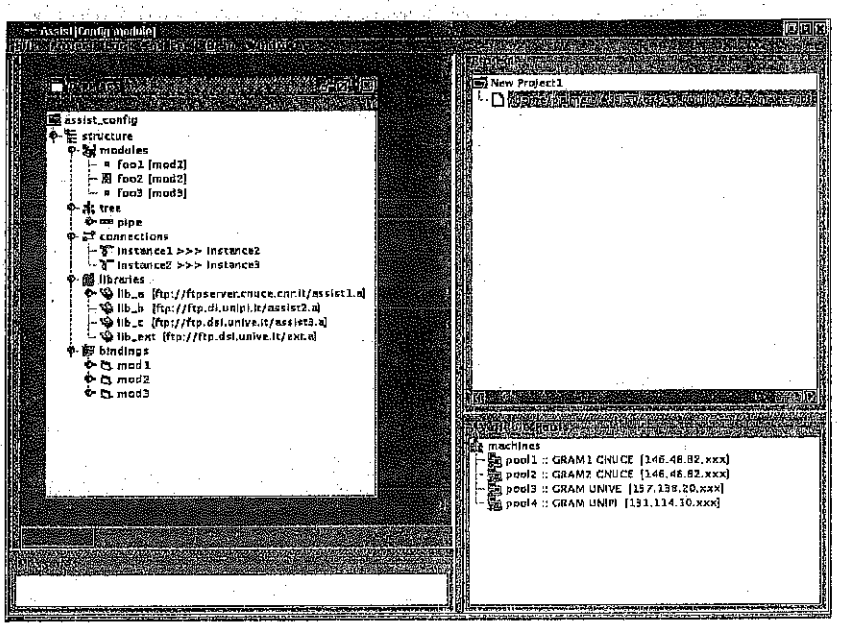


Figure5. AssistConf main window. A new project is created from the configuration file generated by the ASSIST compiler. This file contains only the `structure` part of the configuration, while the other two are built in a semi-automatic way using the AssistConf GUI.

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 ASSIST *Configuration Source* (`.acs` extension).

In Figure 5 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 three stages pipe: `mod1` (sequential), `mod2` (parallel) and `mod3` (sequential). All the elements like connections, libraries and bindings, contained in the structure section are represented, but none of them can be here modified.

In order to add the other two sections of the configuration file, a 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 6), the tree section is changed accordingly to reflect the number of instances of a replicated module or the parallelism degree of a parmod.

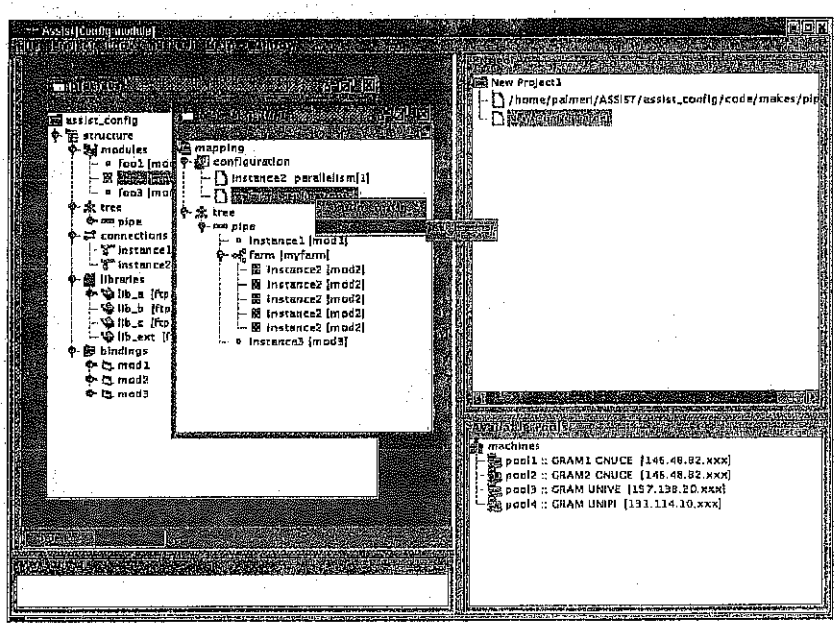


Figure6. The parallel modules of the application can be configured in terms of the parallelism degree of the parmods and the replication degree of the farm workers. Modifications are visualized accordingly.

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 7). 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 one resource, the configuration is complete and we can generate the final configuration file, whose graphical representation is given in Figure 8. This file is called ASSIST *Configuration Target* (`.act` extension), and together with the corresponding `.acs` file, form an ASSIST *Configuration Project* (`.acp` extension). More than 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.

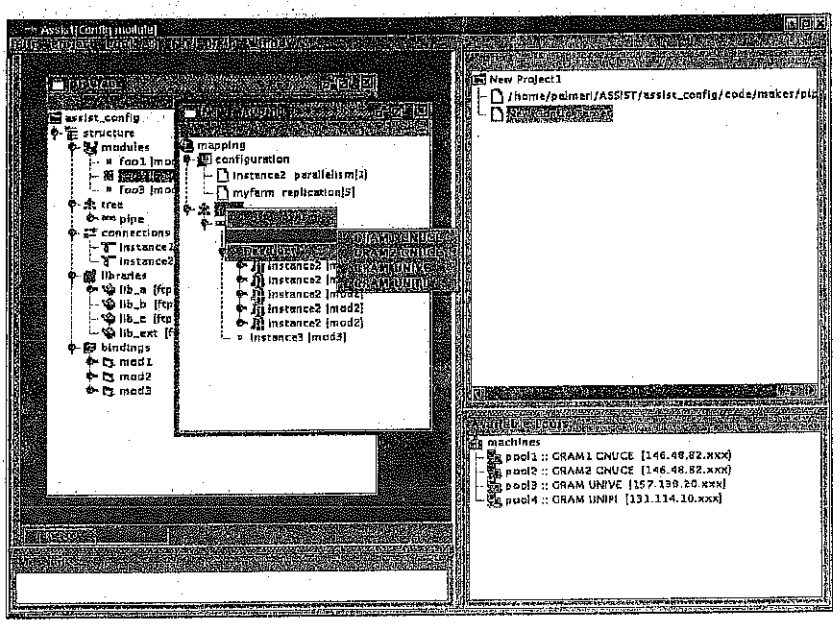


Figure7. Machines can be assigned to the modules by selecting among a set of available resources.

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 simplifiying 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

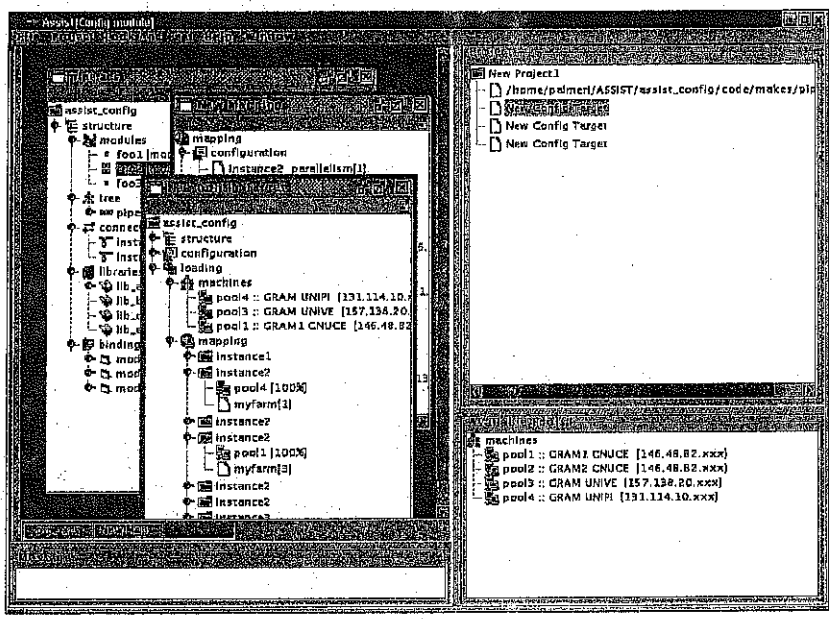


Figure8. Once the parallelism degree and the mapping are completed, the final configuration can be built. There can exhist more than one mapping for the same source, within the same project.

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 describing 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, po. 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. Kesselman (eds) 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 Zhang 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. Douglas 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.