Descrizione Loader GEA

S. Magini, P. Pesciullesi, L. Potiti, G. Virdis, A. Pascucci

{pascucci, virdis}@di.unipi.it

28/11/2006

Versione 1: 25 / 07 / 2005

Versione 1.1: 28 / 11 / 2006

Indice generale

Indice delle illustrazioni

NOTA INTRODUTTIVA:

Nel documento ci riferiamo al tool GEA chiamandolo Loader ASSIST.
La versione di riferimento del codice e' quella specificata nella versione 1.1 riferita nel CVS nella directory astCC1_3/Loader

1 Loader

Il Loader Assist è un'applicazione che serve per lanciare applicazioni ASSIST o componenti ASSIST, CCM o Web Service.
Il Loader attualmente è in grado di lanciare solo le componenti di tipo ASSIST (paragrafo 1.1).

Si possono installare due tipi differenti di Loader:

- L'ASAP SSH e' un Loader che gestisce un insieme di macchine raggiungibile tramite SSH (senza password) dalla macchina su cui viene installato il Loader. Tale insieme di macchine viene specificato dall'amministratore del sistema in un file di configurazione in formato XML (paragrafo 2.2.1).
  In questo caso il Loader può anche gestire un insieme di macchine di una rete privata, per far questo basta installare un ASAP SSH sul nodo front-end della rete privata.
  Per installare questa versione del Loader e' necessario eseguire la compilazione dello strumento usando il makefile *build.xml*. Questa versione necessita soltanto delle librerie contenute nella directory Loader/lib sia per la fase di compilazione che per quella di esecuzione ed e' la versione installata di Default durante la compilazione e installazione di ASSIST

- *L'ASAP GLOBUS e' un Loader che gestisce anche le macchine raggiungibili tramite i meccanismi proprietari di Globus, ossia gestisce le macchine che vengono trovate dal servizio GIS (Grid Information Service) installato su una macchina specificata in un file di configurazione del Loader (paragrafo 2.1). Attualmente e' supportata la versione 4.0.3 di GLOBUS.*
  Per installare questa versione del Loader e' necessario compilare lo strumento utilizzando il makefile buildGlobus.xml. In questo caso sia per la fase di compilazione che per quella di esecuzione vengono utilizzate le librerie di Globus e del tool CogKit. Entrambi questi pacchetti devono essere installati e si deve avere delle variabili d'ambiente settate opportunamente, rispettivamente deve essere settata la variabile GLOBUS_LOCATION e la variabile COG_LOCATION per i due strumenti nel caso si voglia utilizzare questo tipo di ASAP.

Il Loader fornisce dei servizi (paragrafo 1.3.1) raggiungibili tramite comandi da shell, tramite socket oppure può esportare tali servizi tramite Web Service (*ChannelAdaptor_WebService.pdf*).

1.1 La componente ASSIST e l'archivio AAR

Una componente ASSIST è un'applicazione ASSIST che può connettersi ad altre componenti attraverso la libreria di memoria condivisa *hoc* (Appendice 1).
I processi hoc hanno il compito di memorizzare al loro interno i dati che viaggiano sugli stream che la componente ASSIST vuole ricevere od inviare alle altre componenti. Le altre componenti possono cosi' agganciarsi a questi stream recuperando o memorizzando tali dati all'interno della memoria condivisa gestita dai processi hoc.

Il Loader per lanciare una componente ASSIST deve lanciare quindi due gruppi di processi:

- I processi hocStream, ossia i processi hoc che servono per connettere la componente ASSIST con altre componenti.
- I processi dell'applicazione ASSIST che possono essere sia processi ASSIST che processi hoc. I processi hoc sono lanciati nel caso in cui ASSIST sia stato compilato per utilizzare hoc come libreria di memoria condivisa (compilazione senza flag -L).

La componente ASSIST viene compattata in un archivio con estensione “aar” tramite l'utilizzo delle utilility JAR o ZIP. Il file AAR deve contenere al suo interno la struttura illustrata in figura 1.

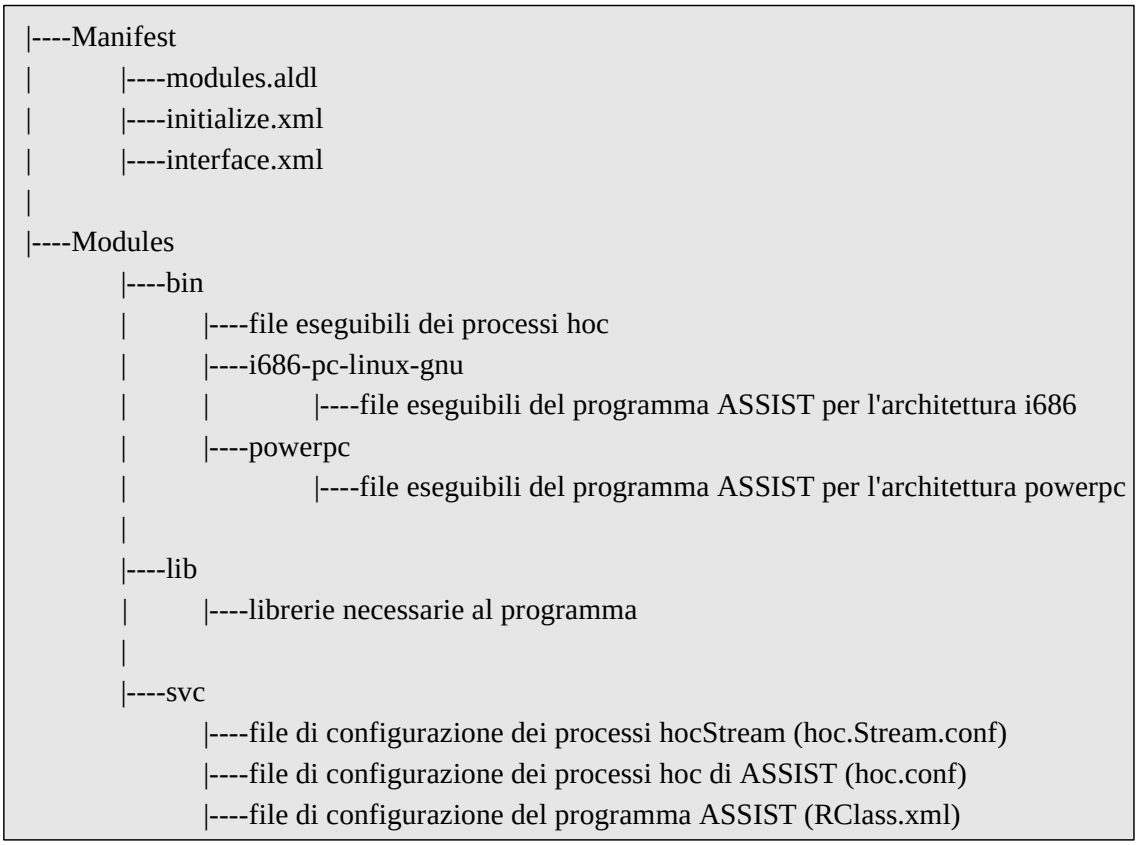

Figura 1. Struttura dei file contenuti nel pacchetto AAR.

Nell'archivio AAR troviamo quattro directory:

1. La directory *Manifest* contiene i file che descrivono la componente, in particolare contiene tre file:

   - Il file *modules.aldl* (ossia il file *ast.out.component.xml* prodotto dal compilatore astCC) descrive il grafo di processi della componente secondo la sintassi ALDL prodotta dal compilatore ASSIST, i path dei file riportati in questo file sono tutti path relativi all'archivio.

   - Il file *initialize.xml* descrive le informazioni di inizializzazione della componente al momento del lancio, come ad esempio quanti processi hocStream lanciare e quali stream e strutture di controllo creare.

   - Il file *interfaces.xml* descrive l'interfaccia della componente (NON UTILIZZATO).

2. La directory *bin* contiene i file eseguibili dell'applicazione ASSIST e dei processi hoc da lanciare.

3. La directory *lib* contiene tutte le librerie necessarie ai file eseguibili dell'applicazione ASSIST.

4. La directory *svc* contiene i file di configurazione dei processi hoc ed i file di configurazione del programma ASSIST, come il file RClass.xml necessario per la riconfigurazione dinamica.

1.2 APPLICAZIONI ASSIST

1. Nel caso di esecuzione di applicazioni ASSIST semplici e' necessario passare al Loader il file contenente il grafo dell'applicazione ASSIST in formato ALDL (ast.out.xml).
   Questo file puo' essere utilizzato nel caso in cui il Loader si trovi in esecuzione sulla stessa macchina su cui e' stata compilata l'applicazione ASSIST in quanto questo file contiene i riferimenti ai file eseguibili ed alle librerie necessarie all'esecuzione dell'applicazione usando path assoluti. Sotto queste condizioni se si cerca di avviare una applicazione ASSIST in una macchina avente una configurazione diversa da quella in cui e' stata compilata e' probabile riscontrare fallimenti in fase di esecuzione.

2. Per ovviare al problema sopra descritto, data la generalita' dell'archivio AAR creato tramite lo script aarer.sh, e' possibile anche utilizzare una componente ASSIST come se fosse una applicazione. Per fare questa operazione e' necessario effettuare il caricamento dell'applicazione come se fosse una componente, tramite il comando LOADAC e richiamare l'esecuzione dell'applicazione con il comando EXECAA.
   Questa operazione ha come vantaggio il fatto che le librerie e gli eseguibili necessari sono ricavati direttamente dal file AAR e non dalla macchina locale come avviene per le normali applicazioni ASSIST.

1.3 I servizi offerti dal Loader

I servizi offerti dal Loader si possono dividere in due gruppi: i servizi pubblici ed i servizi privati.
I servizi pubblici servono ad un generico cliente del Loader per richiedere l'esecuzione di un'applicazione o di una componente ASSIST. I servizi privati servono ai processi lanciati dal Loader per comunicare con il Loader stesso.

I servizi pubblici sono i seguenti:
- Il caricamento in memoria dell'architettura delle macchine gestite dal Loader.
- La visualizzazione dell'architettura gestita dal Loader.
- Per un'applicazione ASSIST:
  - Il caricamento in memoria dell'applicazione ASSIST

    - L'esecuzione di un'applicazione ASSIST caricata in memoria.
- Per una componente ASSIST:
    - Il caricamento in memoria della componente ASSIST.
    - L'esecuzione di una componente ASSIST caricata in memoria.
- La richiesta dei risultati dell'applicazione o della componente ASSIST eseguita.
- L'uccisione dei processi relativi ad un'esecuzione dell'applicazione o della componente ASSIST.
- Lo scaricamento dalla memoria dell'applicazione o della componente ASSIST memorizzata.

I servizi privati sono i seguenti:
- L'aggiunta di una risorsa (un processo VPM) all'esecuzione di un'applicazione ASSIST in corso.
- L'invio di informazioni da ogni processo hocStream lanciato.
- L'invio di informazioni dal processo hoc master lanciato.
- L'invio di informazioni da ogni processo hoc lanciato.
- L'invio di informazioni dal processo strategy lanciato.
- L'invio di informazioni dal processo master lanciato.
- L'uccisione dei processi hocStream lanciati insieme alla componente ASSIST.

Per richiedere i servizi pubblici offerti dal Loader è possibile instanziare un cliente, oppure è possibile richiedere tali servizi anche tramite script di shell.

La classe *ClientAdp* definisce i metodi che può utilizzare un cliente del Loader.
Attualmente esistono due implementazioni in JAVA della classe ClientAdp (la classe TcpClient e la classe WSClient) ed un'implementazione in C++ (la classe TcpClient).

Ogni script di shell prende in ingresso le informazioni per contattare il Loader, ossia l'host e la porta che identifica il socket su cui il Loader accetta delle connessioni ed esegue un programma JAVA che ha il compito di collegarsi col Loader tramite socket e di inviare al Loader il comando associato allo script. Per realizzare questa operazione il programma JAVA utilizza un oggetto della classe TcpClient su cui viene invocato il metodo associato al comando da effettuare nello script.

1.3.1 I servizi pubblici: interazione fra client e Loader e comandi da shell

- **CaricaCluster** (ASAP SSH)

boolean CaricaCluster (StringBuffer result)

CaricaCluster.sh hostLoader portLoader

Il metodo CaricaCluster carica in memoria il file contenente le informazioni sulle macchine gestite dal Loader e restituisce una stringa contenente il risultato del comando. Il path del file

contenente le informazioni sull'architettura (*.../cluster.xml*) si trova all'interno del file di configurazione (*.../configuration.xml*) passato come parametro all'avvio del Loader (paragrafo 2.1).
Lo script CaricaCluster.sh invoca il metodo CaricaCluster su un oggetto della classe TcpClient connesso con il Loader.

Questo metodo deve essere invocato se il Loader è un ASAP SSH, di conseguenza quando si vuole lanciare un programma ASSIST o una componente ASSIST il comando CaricaCluster è il primo comando che deve essere lanciato, senza un'architettura caricata infatti il Loader non può eseguire alcun altro comando.

Nel caso in cui il Loader sia un ASAP Globus, il comando CaricaCluster non deve essere chiamato, nel caso in cui venga chiamato le informazioni caricate vengono eliminate nel momento il cui viene eseguito il controllo delle macchine collegate al GIS di Globus.

- **GetInfo**

boolean GetInfo (StringBuffer stringaXML)

GetInfo.sh hostLoader portLoader

Questo metodo restituisce una stringa contenente le informazioni sull'architettura gestita dal Loader in formato XML, ossia restituisce una stringa contenente il file *cluster.xml* (paragrafo 2.2.1).
Nel caso in cui il Loader sia installato su un nodo front-end di una rete privata è possibile che le informazioni ricevute siano parziali, ad esempio è possibile che l'amministratore del sistema privato non voglia far conoscere all'esterno gli IP delle macchine.
Nel caso in cui il Loader sia un ASAP Globus questo metodo restituisce una stringa vuota.
Lo script GetInfo.sh invoca il metodo GetInfo su un oggetto della classe TcpClient connesso con il Loader e stampa a video i risultati del metodo invocato.

- **LoadAA** (*applicazione ASSIST*)

boolean LoadAA (String applXml, StringBuffer result, StringBuffer libHandle)

LoadAA.sh hostLoader portLoader applXml

Il metodo LoadAA carica in memoria l'applicazione ASSIST descritta nel file *applXml* (ossia nel file *ast.out.xml* generato dal compilatore astCC) e restituisce una stringa contenente il risultato del comando ed il libHandle associato a questa applicazione.

Il file *ast.out.xml* viene descritto in maniera approfondita nel paragrafo 2.5 .
Lo script LoadAA.sh invoca il metodo LoadAA su un oggetto della classe TcpClient connesso con il Loader.

- **ExecAA** (*applicazione ASSIST*)

ExecAA (int libHandle, StringBuffer result, StringBuffer runHandle)

ExecAA.sh hostLoader portLoader libHandle

Il metodo ExecAA esegue il mapping e lo stage&run dell'intera applicazione ASSIST associata al libHandle.
Questo metodo ritorna subito, non attende che l'applicazione termini, questa asincronia è necessaria per permettere l'esecuzione del comando AddProcess.
Restituisce il runHandle della sessione ed il risultato del comando.
Lo script ExecAA.sh invoca il metodo ExecAA su un oggetto della classe TcpClient connesso con il Loader.

NOTA
Questo comando puo' essere richiamato anche in seguito al caricamento in memoria di una applicazione passata in forma di archivio AAR (LoadAC, come descritto nel seguito).

- **loadex.sh** *(solo script)* (*applicazione ASSIST*)

loadex.sh hostLoader portLoader ast.out.xml

Questo script serve per caricare in memoria l'architettura, il file che descrive l'applicazione e per lanciare il programma Assist.
Infatti lo script invoca tre metodi consecutivi (CaricaCluster, LoadAA ed ExecAA) su un oggetto della classe TcpClient connesso con il Loader.
Restituisce a video il risultato dei tre comandi, il libHandle associato all'applicazione ed il runHandle associato all'esecuzione.

- **LoadAC** (*componente ASSIST*)

boolean LoadAC (String aar, StringBuffer result, StringBuffer libHandle)

LoadAC.sh hostLoader portLoader aar

Il metodo LoadAC carica in memoria le informazioni sulla componente descritta nel file AAR

passato come parametro (*aar*).
L'archivio AAR viene scompattato in una directory temporanea. Il file *modules.aldl* contenuto nell'archivio corrisponde al file *ast.out.xml* che descrive l'applicazione ASSIST, l'unica differenza fra i due file XML si trova nella descrizione dei file da trasferire: infatti nel file *modules.aldl* i path sono relativi alla directory di scompattamento, mentre nel file *ast.out.xml* i path sono assoluti.
Il metodo LoadAC esegue le stesse operazioni del metodo LoadAA considerando il file XML *modules.aldl* come il file che descrive l'applicazione ASSIST da lanciare.
Questo metodo restituisce il libHandle della sessione ed il risultato del comando.
Lo script LoadAC.sh invoca il metodo LoadAC su un oggetto della classe TcpClient connesso con il Loader.

NOTA
Questo comando puo' essere utilizzato anche nel caso interessi passare al Loader una applicazione ASSIST in forma di archivio AAR.

- **ExecAC** (*componente ASSIST*)

boolean ExecAC (int libHandle, String aar, StringBuffer result, StringBuffer runHandle, StringBuffer stringXml)

ExecAC.sh hostLoader portLoader libHandle [aar]

Il metodo ExecAC ha il compito di inizializzare la componente e di lanciare la componente associata al libHandle passato come parametro.
Questo metodo prende in ingresso un file AAR che contiene i dati (file di ingresso) necessari al corretto funzionamento della componente. Il nome di questo file deve essere uguale a stringa vuota nel caso in cui la componente non necessiti di file di input.
Nel caso in cui il pacchetto dati sia presente viene scompattato in una directory temporanea e vengono opportunamente settati i path dei file di input da trasferire.
Successivamente la ExecAC lancia i processi hocStream descritti nel file *initialize.xml* (paragrafo 2.6) contenuto nel file AAR passato come parametro alla LoadAC associata al libHandle e li collega alla libreria *streamlib3*. I processi hocStream sono processi hoc che servono per connettere la componente ASSIST con altre componenti.
Infine vengono eseguite le stesse operazioni descritte nel metodo ExecAA, ossia mapping, stage&running dell'applicazione ASSIST associata al libHandle.
Questo metodo restituisce il runHandle della sessione, una stringa in formalismo XML contenente alcune informazioni sui processi hocStream lanciati (*stringXml*) ed il risultato del comando.
Lo script ExecAC.sh invoca il metodo ExecAC su un oggetto della classe TcpClient connesso

con il Loader e stampa a video i risultati del metodo invocato.

- **GetResult**

GetResult (int runHandle, String nomefile_ass, StringBuffer result)

GetResult.sh hostLoader portLoader runHandle fileNamePackAAR

Il metodo GetResult impacchetta e trasferisce i risultati (file di output) dell'esecuzione associata al runHandle nell'archivio AAR (*nomefile_ass*).
Il metodo non ritorna finché l'esecuzione identificata dal runHandle non è terminata.
Il metodo restituisce anche una stringa contenente il risultato del comando.
Lo script GetResult.sh invoca il metodo GetResult su un oggetto della classe TcpClient connesso con il Loader.

- **KillProcess**

KillProcess (int runHandle, StringBuffer result)

KillProcess.sh hostLoader portLoader runHandle

Il metodo KillProcess serve per uccidere tutti i processi lanciati durante l'esecuzione associata al runHandle passato. Restituisce il risultato del comando. Questo metodo ha anche un utilizzo privato (paragrafo seguente).
Lo script KillProcess.sh invoca il metodo KillProcess su un oggetto della classe TcpClient connesso con il Loader.

- **Close**

boolean Close (int libHandle, StringBuffer result)

Close.sh hostLoader portLoader libHandle

Questo metodo scarica dalla memoria tutte le informazioni sull'applicazione ASSIST o sulla componente ASSIST associata al libHandle. Di conseguenza scarica dalla memoria anche tutte le informazioni legate alle esecuzioni associate ad un runHandle collegato al libHandle. Inoltre rimuove anche tutte le directory ed i file temporanei creati dal Loader durante tali esecuzioni. Nel caso in cui ci sia almeno un'esecuzione non terminata, allora la Close invia un messaggio di errore. Lo script Close.sh invoca il metodo Close su un oggetto della classe TcpClient connesso con il Loader.

1.3.2 I servizi privati: interazione tra Loader ed Application Manager e processi lanciati dal Loader

I servizi privati sono implementati tramite tre comandi: il comando AddProcess, il comando KillProcess ed il comando GenericCommand.
Questi servizi vengono utilizzati dall'Application Manager o dai processi lanciati dal Loader (processi ASSIST o processi hoc) e servono per inviare informazioni o richieste al Loader.

Il comando AddProcess

AddProcess (int numProcess, int runHandle, String nomeEseguibile, StringBuffer result)

AddProcess.sh hostLoader portLoader runHandle nomeEseguibile [numProcess=1]

Il metodo AddProcess e lo script di shell eseguono il mapping e lo stage&run di *numProcess* risorse (*nomeEseguibile*) da aggiungere all'applicazione ASSIST associata al runHandle.
Questo servizio può essere invocato solo dal processo strategy dell'applicazione, poiché è l'Application Manager attraverso le sue analisi a sapere se e quando il contratto di Performance sia stato violato e a sapere dunque quando bisogna aumentare il numero di risorse.
Attualmente lo strategy non è ancora in grado di richiedere automaticamente l'aumento delle risorse, di conseguenza il servizio è disponibile anche per i client generici.

Il comando KillProcess

KillProcess (int runHandle, StringBuffer result)

KillProcess.sh hostLoader portLoader runHandle

Il metodo KillProcess può essere chiamato dall'Application Manager quando l'esecuzione di una componente è finita, in questo caso la funzione KillProcess serve per uccidere i processi hocStream lanciati dal Loader per quella componente.
L'Application Manager, infatti, conosce quando la componente ASSIST è finita perché il Loader invia un messaggio di terminazione al processo hocIndice *(Appendice A*) che è in contatto con l'Application Manager.
Lo script KillProcess.sh invoca il metodo KillProcess su un oggetto della classe TcpClient connesso con il Loader.

Il comando GenericCommand per inviare informazioni al Loader

boolean GenericCommand (int runHandle, String generic_command, StringBuffer result)

GenericCommand.sh hostLoader portLoader runHandle generic_command

Il metodo GenericCommand e lo script di shell prendono in ingresso un runHandle ed una stringa contenente le informazioni che il processo vuole comunicare al Loader.

Il Loader attualmente può ricevere le seguenti informazioni (*generic_command*):

- PORT_S:*port*
  dove *port* è la porta aperta dal processo *strategy* dell'applicazione ASSIST.
- PORT_M:*port*
  dove *port* è la porta aperta dal processo *master* dell'applicazione ASSIST.
- HOCM:*host*:*port*
  dove *host* e *port* sono rispettivamente l'host su cui è stato lanciato il processo hoc master e la porta aperta dal processo hoc master per interagire con gli altri processi hoc.
- HOCS:*host*:*port*
  dove *host* e *port* sono rispettivamente l'host su cui è stato lanciato un processo hoc e la porta aperta dal processo hoc per interagire con i processi ASSIST.
- HOCSTR:host:port
  dove *host* e *port* sono rispettivamente l'host su cui è stato lanciato un processo hocStream e la porta aperta da quel processo.

UTILIZZO DEL COMANDO GenericCommand DURANTE IL LANCIO DI UNA COMPONENTE O DI UN'APPLICAZIONE ASSIST

Supponiamo che il Loader debba lanciare una *componente ASSIST* allora il Loader deve lanciare prima tutti i processi **hocStream** e successivamente l'*applicazione ASSIST*.

I processi hocStream da lanciare sono descritti nel file *initialize.xml* (paragrafo 2.6), tali processi inviano al Loader tramite il comando GenericCommand (HOCSTR) l'host su cui sono stati lanciati e la porta aperta per l'interazione con gli altri processi hoc. Dopo che il Loader ha ricevuto questa comunicazione da parte di tutti i processi hoc lanciati, stampa a video la parte *Communicator* del file XML *initialize.xml* completa delle informazioni sugli host e sulle porte aperte da ogni processo hoc lanciato.

Dopo aver lanciato i processi hocStream il Loader può lanciare l'applicazione ASSIST senza più considerare che l'applicazione fa parte di una componente.

Se l'applicazione ASSIST viene compilata per funzionare con i reference implementati con la libreria di memoria condivisa hoc (compilazione senza flag -L), il Loader deve lanciare i processi hoc descritti nel file XML che descrive l'intera applicazione prima di lanciare i processi ASSIST (*ast.out.xml* o *modules.aldl*).

Come prima cosa il Loader deve capire quanti processi hoc deve lanciare. Il Loader assume che su ogni macchina su cui lancerà almeno un processo ASSIST potrà essere lanciato al più un processo hoc. Data questa ipotesi il Loader deve lanciare gli **hoc necessari** descritti nel file XML che descrive l'applicazione ASSIST, nel caso in cui le macchine utilizzate dal Loader siano in numero minore rispetto al numero degli hoc necessari, allora il Loader comunica che non è possibile lanciare tale numero di processi hoc e non fa partire l'esecuzione dell'applicazione.
Nel caso in cui invece tale vincolo sia rispettato, il Loader segna le macchine su cui verranno lanciati gli hoc necessari e recupera dal file XML che descrive l'applicazione ASSIST quali processi hanno bisogno di un processo **hoc di tipo bridge**, ossia quali processi necessitano di un processo hoc sulla macchina dove saranno lanciati. Per tutti questi processi, qualora la macchina di destinazione non coincida con una delle macchine precedentemente segnate per il lancio di un processo hoc, sarà lanciato un ulteriore processo hoc (un hoc di tipo bridge).

Deciso il numero dei processi hoc da lanciare e le macchine su cui devono essere eseguiti, la procedura di lancio è la seguente:

1. Il Loader sceglie casualmente un processo hoc tra quelli da lanciare e lo lancia come master. La riga di comando per lanciare il processo hoc master è la seguente:
   *hoc -L hostLoader:portLoader:runHandle:**HOCM** -S hostHocM:portHocM:alias:numHoc*
   dove:

- *hostLoader* e *portLoader* indicano l'host e la porta su cui è in attesa il Loader
- *runHandle* indica l'handle della sessione di esecuzione associato al programma Assist
- HOCM indica che il processo hoc lanciato deve fare le veci del master
- *hostHocM* indica l'host dove il processo hoc master è stato lanciato, *portHocM* in questo caso non è significativo poiché è l'hoc master che sceglie la porta libera su cui accettare le richieste dagli altri processi hoc
- *alias* è l'alias associato al processo hoc e descritto nel file che descrive l'applicazione ASSIST (*ast.out.xml* o *modules.aldl*).
- *numHoc* è il numero degli hoc che saranno lanciati per l'intera applicazione Assist

2. L'hoc lanciato invia al Loader tramite il comando GenericCommand (HOCM) l'host su cui è stato lanciato e la porta aperta per l'interazione con gli altri hoc.

3. Il Loader riceve l'host e la porta aperta dal processo hoc master e lancia tutti gli altri hoc passandogli a riga di comando anche le informazioni sull'host e sulla porta dell'hoc master. La riga di comando per lanciare un processo hoc slave è la seguente:

   *hoc -L hostLoader:portLoader:runHandle:**HOCS** -S hostHocM:portHocM:alias*

   dove:
   - *hostLoader* e *portLoader* indicano l'host e la porta su cui è in attesa il Loader
   - *runHandle* indica l'handle della sessione di esecuzione associato al programma Assist
   - HOCS indica che il processo hoc lanciato è un hoc slave
   - *hostHocM* e *portHocM* indicano rispettivamente l'host e la porta dove il processo hoc master attende le connessioni dagli altri processi hoc
   - *alias* è l'alias associato al processo hoc

4. Ognuno dei processi hoc lanciati, compreso l'hoc master, invia al Loader tramite il comando GenericCommand (HOCS) l'host su cui è stato lanciato e la porta aperta per comunicare con i processi ASSIST.

5. Il Loader riceve queste informazioni e per ogni processo ASSIST setta la variabile di ambiente HOC_SERVERS contenente l'host e la porta dell'hoc associato al processo ASSIST.

A questo punto il Loader può lanciare l'applicazione ASSIST.

La procedura di lancio di un'applicazione ASSIST è la seguente:

1. Nel caso in cui sia presente un processo di tipo ***strategy***, il Loader lancia questo processo che apre una porta e la invia al Loader utilizzando il comando GenericCommand (PORTS). Il Loader riceve la porta dal processo strategy e passa l'informazione al processo master dell'applicazione tramite il flag *-S host:port* da riga di comando.
2. Il Loader lancia il ***master*** dell'applicazione (con flag -S se lo strategy è presente) che apre una

porta e la invia al Loader tramite il comando GenericCommand (PORTM). Il Loader riceve la porta dal master e passa l'informazione a tutti gli altri processi dell'applicazione tramite il flag *-p port* da riga di comando.

3. Dopo aver ricevuto la porta del processo master dell'applicazione ASSIST il Loader lancia ***tutti gli altri processi*** dell'applicazione ASSIST.

2 Come lanciare il Loader (script assistrun.sh)

Tramite lo script *assistrun.sh* che si trova nella directory $LOADER_ROOT (ossia nella directory $ASTCC_ROOT/Loader) è possibile instanziare i Loader opportuni per lanciare una data applicazione ASSIST.

Lo script *assistrun.sh* prende in ingresso due parametri:

1. *configuration.xml* è il file XML di configurazione degli Information Provider (paragrafo 2.1).
   Questo file descrive quale tipo di Loader instanziare e quale è il suo punto di contatto per ricevere informazioni sulle macchine che può utilizzare.

2. *userconfig.xml* è il file XML di configurazione dell'utente (paragrafo 2.4).
   Questo file descrive la macchina su cui viene lanciato il Loader.
   Nel caso in cui il Loader utilizzi i meccanismi dell'SSH, allora le macchine gestite dal Loader sono descritte nel file cluster.xml (paragrafo 2.2.1) ed e' possibile confrontare le informazioni contenute nel file cluster.xml con quelle contenute nel file userconfig.xml per decidere se siamo in presenza di un ambiente con Network File System.
   Il parametro userconfig.xml è opzionale e se non viene specificato viene preso di default il file userconfig.xml che si trova nella directory $LOADER_ROOT/xml.

Nella figura 2 troviamo alcuni esempio di lancio del Loader.

Esempio 1 (lancio dalla directory LOADER_ROOT con path relativi):
-->cd $LOADER_ROOT
-->./assistrun.sh xml/configuration.xml xml/userconfig.xml

Esempio 2 (lancio dalla directory LOADER_ROOT con userconfig.xml di default):
-->cd $LOADER_ROOT
-->./assistrun.sh xml/configuration.xml

Esempio 3 (lancio dalla directory LOADER_ROOT con path assoluti e con userconfig.xml di default):
-->cd $LOADER_ROOT
-->./assistrun.sh $HOME/configuration.xml

Esempio 4 (lancio con path assoluti e con userconfig di default):
-->$LOADER_ROOT/assistrun.sh $LOADER_ROOT/xml/configuration.xml

Figura 2. Esempi di lancio del Loader.

NOTA

E’ possibile passare come parametro allo script assistrun.sh la stringa –version o la stringa -v per avere informazioni sul numero di versione del Loader avviato dallo script:

Esempio:

```
[pascucci@orione Loader]$ ./assistrun.sh --version
Loader Version: 1.1
[pascucci@orione Loader]$
```

2.1 Il file di configurazione degli Information Provider (configuration.xml)

Il file di configurazione degli IP descrive il tipo degli Information Provider ammessi genericamente ed il caricatore (*loader*) disponibile per eseguire l'applicazione ASSIST.
In linea di principio e’ possibile utilizzare piu’ tipi di Information Provider contemporaneamente utilizzando piu’ elementi del tipo loader. (Caratteristica non testata nella presente versione 1.1)

2.1.1 Gli Information Provider

I tipi degli Information Provider attualmente implementati sono tre:
ASSIST, GT2 (per l’uso del Loader con Globus utilizzando i servizi Pre Web Services) e GT4 (per l’uso del Loader con Globus versione 4). Attualmente e’ pienamente supportato l’ IP GT4 mentre non e’ garantito il funzionamento del Loader con IP GT2.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://www.isti.cnr.it/schemas/configuration">
 <type      name="ASSIST"          classToLoad="it.unipi.di.XMLFileResolver"/>
 <type      name="GT2"             classToLoad="it.unipi.di.grm.ext.GT2Resolver"/>
 <type      name="GT4"             classToLoad="it.unipi.di.grm.ext.GT4Resolver"/>
 ... ... ...
 <loader    type=... ... ...   contact=... ... ...   gateway=... .... ...>
 </loader>
 ... ... ...
</configuration>
```

Figura 3. Descrizione dei tipi degli Information Provider.

Affianco al nome dell'Information Provider troviamo il nome della classe da caricare (*classToLoad*)

per eseguire le operazioni di ricerca delle macchine su cui lanciare l'applicazione ASSIST. Tale classe implementa la classe it.unipi.di.grm.ext.ResourceResolver.

Per gestire i componenti di tipo Legacy è possibile espandere i tipi degli Information Provider ammessi scrivendo una classe che implementa la classe ResourceResolver in modo che ricerchi macchine su cui è possibile eseguire il componente Legacy.

2.1.2 I Loader

Nel file di configurazione degli IP oltre alla descrizione dei tipi degli IP troviamo anche l'elenco dei caricatori (*loader*) che si possono utilizzare.
Ogni caricatore viene collegato ad un unico tipo di IP (*type*) e contiene un punto di contatto (*contact*) a partire dal quale è possibile ricavare le informazioni sulle macchine su cui è possibile eseguire l'applicazione ASSIST. Inoltre ad ogni caricatore viene associato un file (*gateway*) che permette al Loader di conoscere il processo hocIndice precedentemente instanziato e le macchine pubbliche che il Loader ha a disposizione per lanciare i processi hocStream (*gateway.xml*, paragrafo 2.2.2).

2.1.2.1 I punti di contatto (attributo contact)

Il punto di contatto del caricatore per l'**IP ASSIST** (figura 4) è il path di un file XML (.../*cluster.xml*, paragrafo 2.2.1) in cui troviamo la descrizione di una o più macchine raggiungibili tramite SSH dalla macchina di partenza.

Il punto di contatto del caricatore per l'**IP GT2** o **GT4** (figura 5, 6) è il nome della macchina e, nel caso di IP GT2, la porta su cui è installato il servizio GIS (Grid Information Service). È, infatti, possibile richiedere al servizio le informazioni sulla macchina su cui è installato il servizio e sulle macchine raggiungibili da questo.

2.2 Esempi di file di configurazione degli Information Provider

Nella figura 4 troviamo un esempio di caricatore per l'IP ASSIST. Il punto di contatto è il file *xml/cluster.xml* descritto nel paragrafo 2.2.1.
Questo file di configurazione lancia un ASAP SSH che gestisce le macchine descritte nel file cluster.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration  xmlns="http://www.isti.cnr.it/schemas/configuration">
 <type          name="ASSIST"        classToLoad="it.unipi.di.resource.ASAPResolver"/>
<loader         type="ASSIST"        contact="xml/cluster.xml"  gateway="xml/gateway.xml">
   <stage       classToLoad="it.unipi.di.resource.ASAPResolver"/>
   <run         classToLoad="it.unipi.di.resource.ASAPResolver"/>
 </loader>
</configuration>
```

Figura 4. Esempio di file configuration.xml per lanciare un ASAP SSH (*confLocalAsap.xml*).

Nella figura 5 troviamo un esempio di caricatore per l'IP GT2. Il punto di contatto di questo tipo di caricatore è il nome della macchina e la porta su cui è installato il GIS.
Questo file di configurazione lancia un ASAP Globus che contatta le macchine collegate al GIS installato sulla macchina capraia.di.unipi.it.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration  xmlns="http://www.isti.cnr.it/schemas/configuration">
 <type          name="GT2"    classToLoad="it.unipi.di.grm.ext.GT2Resolver"/>
 <loader        type="GT2"    contact="capraia.di.unipi.it:2135"
                base="Mds-Vo-name=site, o=Grid"      gateway="xml/gateway.xml">
   <stage       classToLoad="it.unipi.di.grm.ext.GT2Resolver"/>
   <run         classToLoad="it.unipi.di.grm.ext.GT2Resolver"/>
 </loader>
</configuration>
```

Figura 5. Esempio di file configuration.xml per lanciare un ASAP Globus con IP GT2 (*confGlobus.xml*).

Nella figura 6 possiamo vedere un esempio di caricatore per l' IP GT4.
Per questo tipo di caricatore e' necessario avere soltanto il nome della macchina in cui e' in esecuzione il servizio MDS di Globus. Il contatto (la URI del servizio) viene ricavato da questa informazione aggiungendo delle informazioni secondo lo standard GT4 (dall'esempio la url completa del servizio e' la seguente:

https://cristal.di.unipi.it:8443/wsrf/services/DefaultIndexService

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration  xmlns="http://www.isti.cnr.it/schemas/configuration">
  <type          name="GT4"    classToLoad="it.unipi.di.grm.ext.GT4Resolver"/>
  <loader        type="GT4"    contact="cristal.di.unipi.it"
                 base="Mds-Vo-name=site, o=Grid"      gateway="xml/gateway.xml">
    <stage       classToLoad="it.unipi.di.grm.ext.GT4Resolver"/>
    <run         classToLoad="it.unipi.di.grm.ext.GT4Resolver"/>
 </loader>
</configuration>
```

Figura 6. Esempio di file configuration.xml per lanciare un ASAP Globus con IP GT4 (*confGlobus4.xml*).

2.2.1 Il file cluster.xml (solo per ASAP SSH)

Il file *cluster.xml* contiene le informazioni sulle macchine raggiungibili tramite SSH dalla macchina di partenza del Loader descritta nel file di configurazione *userconfig.xml* (paragrafo 2.4).
Questo file può contenere informazioni su macchine singole (figura 7) e/o su cluster (figura 8).

```
<archconfig   xmlns="http://www.di.unipi.it/schemas/cluster">
  <arch   id = "andormeda"   type = "i686">
    <nfs   id = "M0"   pathMount = "/home"   hide = "no"/>
    <tmpFS   type = "private"   path = "/tmp"   hide = "no"/>
    <tmpFS   type = "shared"   path = "/home/potiti/tmpesec"   hide = "no"/>
    <cpu   vendor = "Intel" platform = "i686"   speed = "800"   hide = "no"/>
    <ram   size = "1024"   hide = "no"/>
    <net   address = "131.114.2.104"   mask = "24"   hide = "no"/>
    <machines>
          <machine   name = "andromeda"  ip = "131.114.2.104"  ram = "512" load = "0.98"
                     disk = "1024" hide = "no" hide_ip = "no"/>
    </machines>
  </arch>
  ... ... ...
</archconfig>
```

Figura 7. Descrizione della macchina andromeda nel file *cluster.xml*.

```
<archconfig   xmlns="http://www.di.unipi.it/schemas/cluster">
 <arch   id = "pianosa"   type = "i686">
    <nfs   id = "M1"   pathMount = "/home"   hide = "no"/>
    <tmpFS   type = "private"   path = "/tmp"   hide = "no"/>
    <tmpFS   type = "shared"   path = "/home/potiti/tmpesec"   hide = "no"/>
    <cpu   vendor = "Intel" platform = "i686"   speed = "800"   hide = "no"/>
    <ram   size = "1024"   hide = "no"/>
    <net   address = "10.0.10.0"   mask = "24"   hide = "no"/>
    <machines>
            <machine   name = "u3"  ip = "10.0.10.3"  ram = "512" load = "0.98"
                       disk = "1024" hide = "no" hide_ip = "no"/>
            <machine   name = "u4"  ip = "10.0.10.4"  ram = "512" load = "0.98"
                       disk = "1024" hide = "no" hide_ip = "no"/>
    </machines>
 </arch>
 ... ... ...
</archconfig>
```

Figura 8. Descrizione del cluster pianosa (u3, u4, u5) nel file *cluster.xml*.

Il file *cluster.xml* contiene all'interno dell'elemento *arch* tutte le informazioni su una macchina singola o su un cluster.

L'elemento *arch* a sua volta contiene gli attributi *id* e *type*, che indicano rispettivamente l'identificatore ed il tipo (*i686* se l'architettura è di tipo intel o *powerpc* se l'architettura è di tipo macintosh) dell'architettura descritta in seguito.

L'attributo *hide* all'interno di un elemento significa che l'informazione contenuta nell'elemento è privata se vale “yes”, pubblica se vale “no”. Nel caso in cui l'informazione sia privata, quando un cliente richiede al Loader l'esecuzione del servizio GetInfo, questo elemento del file cluster.xml viene nascosto.

All'interno dell'elemento *arch* troviamo i seguenti elementi:

- L'elemento *nfs* contiene le informazioni sul Network File System.
  Questo elemento contiene gli attributi *hide*, *id* e *pathMount*.
  L'attributo *id* è l'identificatore dell'NFS, l'attributo *pathMount* indica il path mount dell'NFS.
  Questi due attributi devono essere confrontati con l'identificatore dell'NFS ed il path mount dell'NFS della macchina dove viene lanciato il Loader per capire se i file binari ed i file di input

dell'applicazione ASSIST devono essere copiati o meno nella macchina destinazione.

- Se gli identificatori ed i path mount coincidono, significa che tra le due macchine c'è un Network File System e che i file sulla macchina di partenza sono esattamente gli stessi file sulla macchina di arrivo. Quindi se il programma ASSIST richiede l'utilizzo dell'NFS (shared=”yes”), i file non devono essere copiati.

- Se gli identificatori coincidono, ma i due path mount sono diversi significa che tra le due macchine c'è un Network File System e che i file sulla macchina di partenza e su quella destinazione sono fisicamente gli stessi ma hanno path diversi, questo implica che i file sono accessibili dalla macchina sorgente tramite un path e dalla macchina destinazione tramite un altro path. Anche in questo caso se il programma ASSIST richiede l'utilizzo dell'NFS (shared=”yes”), i file non devono essere copiati, ma si devono modificare i path dei file che saranno eseguiti sulla macchina di destinazione.

- Se gli identificatori sono diversi, allora non c'è un Network File System tra le due macchine. I file devono essere copiati sempre nella macchina di destinazione.
  Se la macchina di destinazione è un cluster ed il programma ASSIST richiede l'utilizzo dell'NFS (shared=”yes”), allora i file saranno copiati una sola volta nel cluster.

Nella tabella 9 troviamo una descrizione dei casi che si possono presentare.

ID NFS	***Path Mount NFS***	***File shared***	***Copia***
=	X	no	no
!=	X	no	N copie nel path privato
=	=	si	no
=	!=	si	no --> cambio path
!=	X	si	si --> 1 copia per cluster nel path shared

Tabella 9. Illustrazione dei casi in cui è necessario copiare i file binari di ASSIST su una macchina destinazione.

- L’emento *tmpFS* contiene le informazioni sui path su cui è possibile copiare i file.
  Questo elemento contiene l'attributo *hide* e gli attributi *type* e *path*.
  L'attributo *type* può valere “private” o “shared”.
  Se *type* vale “private”, allora l'attributo *path* indica il path dove il Loader deve copiare i file non shared. Se invece *type* vale “shared”, allora l'attributo *path* indica il path dove il Loader deve copiare i file shared.

- L'elemento *cpu* contiene le informazioni sul processore (attualmente è possibile definire solo cluster omogenei).
  Questo elemento contiene gli attributi *hide, vendor, platform* e *speed*.
  L'attributo *vendor* indica il tipo di macchina (Intel).
  L'attributo *platform* indica il tipo di architettura della macchina (i686 o powerpc).
  L'attributo *speed* indica i MHz della cpu.

- L'elemento *ram* contiene le informazioni sulla memoria.
  Questo elemento contiene gli attributi *hide* e *size*.
  L'attributo *size* indica la dimensione della memoria della macchina.

- L'elemento *net* contiene le informazioni sulla rete.
  Questo elemento contiene gli attributi *hide, address* e *mask*.
  L'attributo *address* indica l'indirizzo IP della macchina o del gruppo di macchine.
  L'attributo *mask* indica un insieme di indirizzi IP.

- L'elemento *machines* contiene le informazioni sulle macchine di questa architettura.
  All'interno dell'elemento *machines* troviamo uno o più elementi *machine*. Questo elemento descrive una singola macchina e contiene gli attributi *name, ip, ram, load, disk, hide* e *hide_ip*.
  L'attributo *name* contiene il nome della macchina.
  L'attributo *ip* contiene l'indirizzo IP della macchina. Questo attributo può essere mascherato ad un eventuale cliente del Loader che chiede informazioni sull'architettura dando il valore “yes” all'attributo *hide_ip*.
  L'attributo *ram* contiene la dimensione della memoria ram della macchina.
  L'attributo *load* contiene il carico della macchina.
  L'attributo *disk* contiene la dimensione della memoria fissa della macchina.

2.2.2 Il file gateway.xml

Il file XML *gateway.xml* (figura 10) descrive alcune informazioni necessarie alle componenti per comunicare tra di loro.
In questo file troviamo le informazioni (host e porta) del processo hocIndice. Un processo hocIndice è un processo hoc che è stato lanciato su una macchina pubblica, questo processo viene lanciato prima di lanciare il Loader.

```
<?xml version="1.0" encoding="UTF-8"?>
<gateway>
  <type          protocol="HOC"        hide="YES" />
  <service       ip="... ... ..."      port="6234" />
  <fileLocation  exec="/tmp/hoc"       conf="/tmp/conf" />
  <machine       ip="... ... ..."      portRange="12501-13500" />
  <machine       ip="... ... ..."      portRange="1239-3454" />
  ... ... ...
</gateway>
```

Figura 10. File *gateway.xml*.

L'elemento *type* indica il protocollo di comunicazione da utilizzare, attualmente è implementato solo il protocollo HOC che utilizza come mezzo di comunicazione la libreria di memoria condivisa hoc ed indica se le informazioni sulle macchine pubbliche (elemento *machine*) sono informazioni nascoste o meno (*hide*).

L'elemento *service* indica la macchina (*ip*) e la porta (*port*) del processo hocIndice.
Per lanciare un processo hocIndice e' sufficiente lanciare un processo hoc con i seguenti parametri:

hoc --accept-all -f hocIndice.conf

dove il file hocIndice.conf e' un file di configurazione opportunamente generato in cui viene riportata la porta 6234 scritta nel file gateway.xml.

L'elemento *fileLocation* contiene i path del file eseguibile e di configurazione di default dei processi hocStream, definiti nel file XML *initialize.xml* (paragrafo 2.6).

Gli elementi *machine* contengono ciascuno un indirizzo IP (*ip*) di una macchina pubblica ed un intervallo di porte aperte (*portRange*) (Non utilizzato attualmente TODO) che devono essere utilizzate per lanciare gli hocStream.

2.3 Scelta del Loader da instanziare

A partire dal file di configurazione degli Information IP vengono scelti i Loader da instanziare.

- Per ogni caricatore di tipo **IP ASAP** trovato viene instanziata un **Loader di tipo *ASAP SSH***.
- Per tutti i caricatori di tipo **IP GT2** trovati viene instanziata **un solo *ASAP Globus*** a cui vengono passati tutti i punti di contatto (GIS) trovati nei caricatori di tipo IP GT2.
- (La stessa cosa potrebbe essere applicata all'**IP GT4**.

Successivamente è possibile richiedere ai Loader instanziati l'esecuzione dei servizi visti nel paragrafo .

2.4 Il file di configurazione della macchina su cui viene lanciato il Loader

Il file XML *userconfig.xml* (figura 11) descrive la macchina su cui viene lanciato il Loader ed alcune informazioni sull'utente.

```
<?xml version="1.0" ?>
<userconfig>
  <machineSource      name = "orione"       port = "12700">
     <nfs             id = "M0"             pathMount = "/home/pacifico"/>
     <gridftp         urlprefix = "gsiftp://orione.di.unipi.it:2811/"/>
  </machineSource>
  <sshSeC             username = "pascucci"
   privatekey = "/.automount/pacifico/l/disc1/home/pascucci/.ssh/id_dsa"      passphrase = "" />
</userconfig>
```

Figura 11. File *userconfig.xml*.

All'interno dell'elemento *machinesource* che descrive la macchina su cui viene lanciato il Loader troviamo le seguenti informazioni:

- Il nome della macchina su cui il Loader viene lanciato.
- La porta che il Loader deve aprire e su cui deve attendere le richieste dei clienti.
- Le informazioni sul Network File System, ossia l'identificatore dell'NFS ed il path mount. Questa informazione viene riportata anche per le architetture descritte nel file XML *cluster.xml* (paragrafo 2.2.1), in modo che il Loader sappia se è presente un NFS tra la macchina di partenza e la macchina scelta per l'esecuzione di uno o più processi ASSIST.
  Infatti se i due identificatori e i path mount sono uguali non c'è bisogno di fare alcuna copia dei

file considerati *shared* (paragrafo 2.5.1) poiché la macchina di partenza e la macchina di destinazione hanno lo stesso File System; se gli identificatori sono uguali ed i path mount sono differenti oppure se gli identificatori sono diversi allora si deve fare una sola copia all'interno del cluster scelto dei file considerati shared. Ovviamente tutti i file che non sono considerati shared vengono copiati.

- Il prefisso della URL da utilizzare per eseguire gli stage dei file e l'esecuzione dei processi. Questa informazione viene utilizzata solo nel caso di uso dell'ASAP Globus e indica l'indirizzo del server GRIDFTP locale. Sotto questa ipotesi, dato che in questa modalita' tutti i trasferimenti vengono eseguiti in modalita' "third party", e' necessario avere un server GridFTP installato ed attivo sulla macchina in cui e' in esecuzione il Loader.

L'elemento *sshSeC* descrive le informazioni sulla chiave SSH. Attualmente (Versione 1.1) questa informazione non viene utilizzata, ma viene caricata durante l'esecuzione tramite il metodo System.getProperties di JAVA. La chiave utilizzata per l'SSH è la id_rsa.
I file di configurazione di una componente/applicazione ASSIST

Nel caso di un'applicazione ASSIST è necessario un unico file di configurazione per descrive l'applicazione ASSIST, questo file è il file *ast.out.xml* (paragrafo 2.5) prodotto dal compilatore astCC e preso in ingresso dal comando LoadAA.

Nel caso di una componente ASSIST invece sono necessari tre file di configurazione, questi tre file si trovano nella directory *Manifest* all'interno dell'archivio AAR preso in ingresso dal comando LoadAC.:

- *modules.aldl* descrive il grafo dei processi della componente ASSIST. Questo file è il file ast.out.xml (paragrafo 2.5) prodotto dal compilatore astCC, dove al posto dei path assoluti troviamo i path relativi alla directory di scompattamento.

- *initialize.xml* descrive le informazioni di inizializzazione della componente ASSIST. All'interno di questo file troviamo il numero di processi hocStream che il Loader deve lanciare per la gestione della componente e l'associazione tra gli hocStream e gli stream della componente ASSIST (paragrafo 2.6).

- *interfaces.xml* che descrive l'interfaccia della componente. (non viene ancora utilizzato)

2.5 ALDL: il descrittore del grafo dei processi (ast.out.xml e modules.aldl)

Il compilatore astCC genera due file che descrivono il grafo di processi da lanciare:

- Il file ast.out.xml contiene le informazioni necessarie per lanciare l'applicazione ASSIST.
- Il file ast.out.component.xml contiene le informazioni necessarie per lanciare la componente ASSIST; questo file viene rinominato in modules.aldl quando viene creato l'archivio AAR.

Entrambi questi file sono in formato ALDL, un formato creato appositamente per descrivere una applicazione ASSIST. Questi due file sono creati dal compilatore ASSIST e vengono interpretati dal LOADER per poter costruire il flusso di esecuzione dell'applicazione.

I due file differiscono soprattutto perché nel file che descrive la componente (modules.aldl) troviamo le URL con protocollo aar e dataAar e con path relativi alle directory contenute nell'archivio.

```
<aldl:application xmlns="urn:aldl-assist" ... ... ... targetNamespace="urn:aldl-assist"
                  xmlns:tns="urn:aldl-assist" xsi:schemaLocation="... ... ...">
 <aldl:requirement   name="... ... ...">... ... ... </aldl:requirement>
 ... ... ...
 <aldl:requirement   name="... ... ...">... ... ... </aldl:requirement>
 <aldl:group         name="... ... ...">... ... ... </aldl:group>
 ... ... ...
 <aldl:group         name="... ... ...">... ... ... </aldl:group>
 <aldl:module        name="... ... ...">... ... ...</aldl:module>
 ... ... ...
 <aldl:module        name="... ... ...">... ... ...</aldl:module>
 <aldl:instance      name="... ... ...">... ... ...</aldl:instance>
 ... ... ...
 <aldl:instance      name="... ... ...">... ... ...</aldl:instance>
</aldl:application>
```

Figura 12. Codice del file che descrive il grafo dei processi da lanciare.

Il file di configurazione dell'applicazione ASSIST, come si vede dalla figura 12, contiene quattro sezioni:

1. Nella sezione *requirement* sono descritte tutte le informazioni sui processi che fanno parte dell'applicazione (*aldl:requirement*).
2. Nella sezione *group* vengono definiti i parmod dell'applicazione associando tra loro i processi ism, vpm ed osm che fanno parte di un parmod e i tag di coallocazione(*aldl:group*).

3. Nella sezione *module* troviamo l'eventuale associazione tra i processi descritti nella sezione *requirement* che fanno parte di un parmod ed il gruppo che definisce il parmod (*aldl:module*),
4. Nella sezione *instance* viene descritta l'associazione tra i moduli ed i gruppo di coallocazione.(*aldl:instances*).

La sezione dei requirement è l'unica sezione che l'utente può modificare, le altre sezioni vengono completamente definite dal compilatore ASSIST.

2.5.1 Sezione Requirement

In una sezione di tipo requirement è possibile descrivere un intero processo ASSIST da lanciare in termini di:

- file eseguibile del processo ASSIST,
- file di ingresso necessari al processo ASSIST prima dell'esecuzione,
- file di uscita prodotti dal processo ASSIST al termine dell'esecuzione,
- librerie necessarie al processo ASSIST durante l'esecuzione come la libreria XML2 e ACE,
- processi hoc da lanciare prima del processo ASSIST,
- grado di parallelismo del processo ASSIST nel caso si tratti di un processo di tipo VPM,
- file contenente lo standard output,
- file contenente lo standard error,
- caratteristiche fisiche della macchina su cui si vuole lanciare l'eseguibile.

Se le informazioni da specificare per i processi ASSIST sono ridondanti, come ad esempio quelle per le librerie XML2 ed ACE, e' possibile descrivere un requirement di librerie da utilizzare nella sezione module associando il requirement sul processo ASSIST al requirement sulle librerie (paragrafo 2.5.2.3).

Quindi e' possibile specificare requirement per qualsiasi sottoinsieme delle caratteristiche sopraelencate.

Uno dei requirement più interessanti da specificare e' quello che descrive una macchina o una sottorete (paragrafi 2.5.2.1, 2.5.2.2). Associando questo requirement al requirement di un processo ASSIST si impone che il processo ASSIST venga lanciato su una data macchina o su un dato insieme di macchine.

Gli elementi che può contenere la sezione requirement sono i seguenti:

1. l'elemento *executable* (file eseguibile del processo ASSIST)
2. l'elemento *file* (file di I/O del processo ASSIST)
3. l'elemento *lib* (librerie necessarie al processo ASSIST)
4. l'elemento *hoc* (processi hoc necessari al processo ASSIST)

5. l'elemento *parallelism* (grado di parallelismo del processo ASSIST)
6. l'elemento *stdout* (file contenente lo standard output del processo ASSIST)
7. l'elemento *stderr* (file contenente lo standard error del processo ASSIST)
8. l'elemento *cpu* (potenza di calcolo della macchina su cui si vuole lanciare il processo ASSIST)
9. l'elemento *physicalMemory* (capacità di memoria della macchina su cui si vuole lanciare il processo ASSIST)
10. l'elemento *netAddress* (indirizzo IP della macchina o della sottorete su cui si vuole lanciare il processo ASSIST)
11. l'elemento *netMask* (maschera della macchina su cui si vuole lanciare il processo ASSIST)

Nei paragrafi seguenti descriveremo tutti questi elementi.

2.5.1.1 File eseguibile del processo ASSIST (elemento executable)

Il file ALDL contiene un elemento executable per ogni tipo di architettura consentita.
In questo elemento troviamo alcune informazioni sull'eseguibile stesso, come il path dell'eseguibile ed il tipo di architettura dell'eseguibile (*arch*).
Il path è assoluto nel caso in cui il file *ALDL* descrive un'applicazione ASSIST (figura 13), mentre è relativo nel caso in cui il file *ALDL* descrive una componente ASSIST (figura 14).
I tipi di architettura consentita sono presi dal file ast_rc utilizzato durante la compilazione del programma ASSIST; attualmente sono possibili due soli tipi di architettura: linux (i686) oppure macintosh (powerpc). All'interno dell'elemento executable l'attributo arch è obbligatorio.

```
<aldl:requirement name="ND000__generaConfiguration">
 <ns1:executable arch="i686">/home/po/com/bin/i686-pc-linux-gnu/ND000__genera</ns1:executable>
 <ns1:executable arch="powerpc">/home/po/com/bin/power-pc/ND000__genera</ns1:executable>
 ... ... ...
</aldl:requirement>
```

Figura 13. Sezione requirement: elemento executable nel file ast.out.xml.

```
<aldl:requirement name="ND000__generaConfiguration">
  <ns1:executable arch="i686"> Modules/bin/i686-pc-linux-gnu/ND000__genera </ns1:executable>
  <ns1:executable arch="powerpc">Modules/bin/power-pc/ND000__genera</ns1:executable>
  ... ... ...
</aldl:requirement>
```

Figura 14. Sezione requirement: elemento executable nel file ast.out.component.xml.

Oltre a queste informazioni possiamo trovare altri tre tipi di attributi all'interno dell'elemento executable.

- L'attributo *strategy* (figura 16) può assumere due valori (“yes” e “no”) ed indica se il processo è il processo strategy, ossia il processo CAM_s creato dal compilatore ASSIST nel caso in cui si voglia eseguire una riconfigurazione dinamica di ASSIST (compilazione con flag -R).
  L'attributo è opzionale e di default vale “no”.
  Ogni applicazione ASSIST può avere al massimo un processo strategy.

```
<aldl:requirement name="ND000__generaConfiguration">
  <ns1:executable arch="i686" strategy="yes" >/home/po/comp/bin/i686/CAM_s</ns1:executable>
  ... ... ...
</aldl:requirement>
```

Figura 15. Attributo strategy nell'elemento executable.

L'attributo *master* (figura 16) può assumere due valori (“yes” e “no”) ed indica se il processo è il processo master dell'applicazione ASSIST.
L'attributo è opzionale e di default vale “no”.
Ogni applicazione ASSIST deve avere un processo di tipo master.

```
<aldl:requirement name="ND000__generaConfiguration">
  <ns1:executable arch="i686" master="yes" >/home/po/comp/bin/i686/ND000__gen</ns1:executable>
  ... ... ...
</aldl:requirement>
```

Figura 16. Attributo master nell'elemento executable.

- L'attributo *shared* (figura 17) può assumere due valori (“yes” e “no”) ed indica se, in caso di NFS, è necessario trasferire il file oppure se lo si può considerare condiviso.
  Questa informazione è molto utile in caso di cluster con home condivise, in questo caso infatti sarà necessario trasferire i file una sola volta sul path shared del cluster (paragrafo 2.2.1) per eseguire i processi da una qualsiasi delle macchine che fanno parte del cluster.
  L'attributo è opzionale e di default vale “no”.

```
<aldl:requirement name="ND000__generaConfiguration">
  <ns1:executable arch="i686" shared="yes">/home/po/comp/bin/i686/ND000__gen</ns1:executable>
  ... ... ...
</aldl:requirement>
```

Figura 17. Attributo shared nell'elemento executable.

2.5.1.2 File di I/O del processo ASSIST (elemento file)

Per ogni requirement all'interno del file *ALDL* troviamo nessuno, uno o più elementi *file* contenenti le informazioni sui file da trasferire prima (*source*) o dopo (*target*) l'esecuzione del processo descritto in questo requirement.

```
<aldl:requirement name="ND000__generaConfiguration">
 ... ... ...
 <ns1:file   conf = "yes"  arch="all"   fileName="genera.in">
   <ns1:source   url="file:////tmp/genera.in"/>
 </ns1:file>
 <ns1:file  arch="all"  fileName="genera.out"  fileSystemName="/tmp">
   <ns1:target    url="return:////tmp/genera.out"/>
 </ns1:file>
 ... ... ...
</aldl:requirement>
```

Figura 18. Sezione requirement: elemento file.

Come si vede dalla figura 18, i file di input sono contrassegnati dall'etichetta source ed i file di output sono contrassegnati dall'etichetta target.
Per ogni file di input viene specificata la *URL* di partenza (protocollo e path del file) ed il nome del file sulla macchina remota (*fileName*), la directory dove il file verra' memorizzato sulla macchina remota viene decisa dal Loader.
Per ogni file di output viene specificato il path assoluto del file sulla macchina remota (dato dalla concatenazione dell'attributo *fileSystem* e *fileName*) e la *URL* di destinazione (protocollo e path del file).
Nel paragrafo 2.5.1.2.1 vengono descritti tutti i tipi di URL ammessi nel file *ALDL*.

L'attributo dell'architettura (*arch*) indica il tipo dell'architettura del file da trasferire, questo attributo è opzionale per l'elemento file ed è settato di default al valore *all*, questo significa che il file da trasferire è indifferente al tipo di architettura.

Nel caso in cui il file XML descrive una componente ASSIST, i path dei file sono relativi all'archivio AAR, nel caso in cui l'attributo *conf* valga *yes*, significa che il file di input è un file di configurazione (come ad esempio il file *RClass.xml* che viene prodotto dal compilatore per la riconfigurazione dinamica) e si trova nell'archivio della componente, ossia nell'archivio caricato con il comando LoadAC, mentre nel caso in cui l'attributo *conf* valga *no*, significa che il file di input è un file di input vero e proprio e si trova nel pacchetto dei dati che viene caricato con il comando ExecAC.

Oltre a queste informazioni possiamo trovare altri due tipi di attributi all'interno dell'elemento file.

- L'attributo *shared* (figura 19) può assumere due valori (“yes” e “no”) ed indica se, in caso di NFS, è necessario trasferire il file oppure se lo si può considerare condiviso.
  L'attributo è opzionale e di default vale “no”.
  A seconda del valore dell'attributo shared e degli identificatori della macchina di partenza (descritto nel file userconfig.xml) e della macchina/cluster di arrivo (descritto nel file cluster.xml, paragrafo 2.2.1) possiamo stabilire se il file sarà copiato oppure no.

```
<aldl:requirement name="ND000__generaConfiguration">
 ... ... ...
 <ns1:file arch="all" shared="yes" fileName="genera.in" >
   <ns1:source url="file:////home/po/genera.in"/>
 </ns1:file>
 ... ... ...
</aldl:requirement>
```

Figura 19. Attributo shared nell'elemento file.

La decisione sul trasferimento di un file di ingresso è identica alla decisione sul trasferimento di una libreria (tabella 29).

- L'attributo *symbolicName* (figura 20) è una stringa che indica il nome del file simbolico all'interno del programma ASSIST relativo a questo file.
  L'attributo è opzionale poiché non tutti i file sono di tipo simbolico.

```
<aldl:requirement name="ND000__generaConfiguration">
 ... ... ...
 <ns1:file          arch="all"   symbolicName="matrice.in"        fileName="genera.in" >
   <ns1:source   url="file:////tmp/genera.in"/>
 </ns1:file>
 <ns1:file          arch="all"   symbolicName="matrice.out"      fileName="genera.out"
                    fileSystemName="/tmp">
   <ns1:target    url="return:////tmp/genera.out"/>
 </ns1:file>
 ... ... ...
</aldl:requirement>
```

Figura 20. Attributo symbolicName nell'elemento file.

NOTA: FILE SIMBOLICI IN ASSIST (Caratteristica non testata nella presente versione)

I file simbolici in un programma ASSIST sono definiti dalla seguente sintassi:

proc fgenera (...)
file <in MATRICE_IN out MATRICE_OUT>

Nel caso in cui il programma ASSIST contenga dei file simbolici, allora i file ast.out.xml e ast.out.component.xml generati dal compilatore astCC non sono completi (figure 21 e 22), poiché manca il ***path*** del file nella URL dell'elemento source o target.

```
<ns1:file   fileName = "MATRICE_IN"  symbolicName = "MATRICE_IN" >
        <ns1:source url = "file:///MATRICE_IN" />
</ns1:file>
<ns1:file   fileName = "MATRICE_OUT"  symbolicName = "MATRICE_OUT" >
        <ns1:target url = "return:///MATRICE_OUT" />
</ns1:file>
```

Figura 21. File simbolici (in ingresso ed in uscita) generati nell'ast.out.xml.

```
<ns1:file  fileName = "MATRICE_IN"  symbolicName = "MATRICE_IN">
        <ns1:source url = "dataAar:///MATRICE_IN" />
</ns1:file>
<ns1:file  fileName = "MATRICE_OUT"  symbolicName = "MATRICE_OUT">
        <ns1:target url = "return:///MATRICE_OUT" />
</ns1:file>
```

Figura 22. File simbolici (in ingresso ed in uscita) generati nell'ast.out.component.xml.

Solo il programmatore conosce il path assoluto (o relativo in caso di componente) del file simbolico, di conseguenza è il programmatore che deve riempire il file XML con il path del file simbolico.

Nel caso di un'applicazione ASSIST il path e' assoluto per la SourceURL, ed e' relativo al pacchetto dei risultati per la TargetURL (figura 23).

```
<ns1:file   fileName = "MATRICE_IN"  symbolicName = "MATRICE_IN" >
        <ns1:source url = "file:////tmp/injacobi" />
</ns1:file>
<ns1:file   fileName = "MATRICE_OUT"  symbolicName = "MATRICE_OUT" >
        <ns1:target url = "return:///outjacobi" />
</ns1:file>
```

Figura 23. File simbolici (in ingresso ed in uscita) corretti nell'ast.out.xml.

Nel caso di una componente ASSIST il path e' relativo al pacchetto DATI per la SourceURL, ed e' relativo al pacchetto dei risultati per la TargetURL (figura 24).

```
<ns1:file  fileName = "MATRICE_IN"  symbolicName = "MATRICE_IN">
        <ns1:source url = "dataAar:///injacobi" />
</ns1:file>
<ns1:file  fileName = "MATRICE_OUT"  symbolicName = "MATRICE_OUT">
        <ns1:target url = "return:///outjacobi" />
</ns1:file>
```

Figura 24. File simbolici (in ingresso ed in uscita) corretti nell'ast.out.component.xml.

2.5.1.2.1 Tipi di protocolli di URL ammessi nell'elemento file

Ogni URL è costituita da due parti: il protocollo ed il percorso del file all'interno del File System. Il protocollo è a sua volta suddiviso in due parti: il nome del protocollo e l'host su cui si trova la URL, come si vede dalla figura 25.

url = "PROTOCOLLO/PATH"
ossia
url = "NomeProtocollo://HostProtocollo/PATH"

Figura 25. Definizione di URL.

Si possono distinguere due tipi di URL:

1. le *SourceURL* sono URL che indicano la sorgente del trasferimento, ossia la macchina dove viene lanciato il Loader.
2. le *TargetURL* sono URL che indicano la destinazione del trasferimento.

I tipi di protocollo che si possono specificare per le URL nel file ***ast.out.xml*** sono i seguenti:

- *file* e *gridftp* per le URL che indicano la SourceURL.

- *file*, *return* e *gridftp* per le URL che indicano la TargetURL.

I tipi di protocollo che si possono specificare per le URL nel file ***modules.aldl*** sono i seguenti:
- *file*, *aar*, *dataAar* e *gridftp* per le URL che indicano la SourceURL.
- *file*, *return* e *gridftp* per le URL che indicano la TargetURL.

Il protocollo *gridftp* è il protocollo standard utilizzato da Globus per il trasferimento di file, gli altri protocolli sono stati creati appositamente per gestire le diverse esigenze riscontrate.
Nel caso in cui il Loader utilizzi i meccanismi dell'SSH le URL di tipo gridftp vengono ignorate, mentre nel caso in cui il Loader utilizzi i meccanismi di Globus tutte le URL specificate vengono opportunamente trasformate in URL di tipo gridftp.

IL PROTOCOLLO FILE
Il protocollo file (***file:///***) indica una URL locale alla macchina su cui viene lanciato il Loader. Il path specificato (/tmp/genera.in) è il path assoluto del file.

```
<ns1:source  url="file:////tmp/genera.in"/>
```

IL PROTOCOLLO RETURN
Il protocollo return (***return:///***) indica una URL che si trova sulla macchina su cui viene lanciato il Loader. Del path specificato (genera.out) viene preso in considerazione solo il nome del file. Tale file viene trasferito dalla macchina remota alla macchina dove è stato lanciato il Loader nella directory (returnDir) creata dal Loader per memorizzare tutti i risultati dell'applicazione.
Il percorso contenuto nel path viene ignorato.

```
<ns1:target   url="return:///genera.out"/>
```

IL PROTOCOLLO AAR
Il protocollo aar (***aar:///***) indica una URL locale alla macchina su cui viene lanciato il Loader. Il path specificato (Modules/svc/RClass.xml) è il path relativo all'archivio AAR della COMPONENTE, ossia relativo all'archivio caricato durante l'esecuzione di un comando di tipo Load.

```
<ns1:source  url="aar:///Modules/svc/RClass.xml"/>
```

IL PROTOCOLLO DATAAAR

Il protocollo dataAar (***dataAar:///***) indica una URL locale alla macchina su cui viene lanciato il Loader. Il path specificato (injacobi0512) è il path relativo all'archivio AAR dei DATI, ossia relativo all'archivio caricato durante l'esecuzione di un comando di tipo Exec.

```
<ns1:source url="dataAar:///injaconbi0512"/>
```

IL PROTOCOLLO GRIDFTP

Il protocollo gridftp (***gsiftp://host/***) indica una URL che si trova sull'host specificato nel protocollo (andromeda.di.unipi.it). Il path specificato (/tmp/genera.in) è il path assoluto del file sulla macchina specificata.

```
<ns1:source url="gridftp://andromeda.di.unipi.it//tmp/genera.in"/>
```

2.5.1.2.2 Trasferimento dei file di I/O

Il trasferimento di un file di input avviene copiando il file dalla URL nell'elemento source nella macchina remota decisa dal Loader durante la fase di mapping. Il file viene creato in una directory temporanea decisa dal Loader ed il nome di destinazione del file è l'attributo *fileName*.
Il trasferimento di un file di output avviene copiando il file descritto dagli attributi *fileSystemName* e *fileName* dalla macchina remota in cui il processo è stato eseguito nel file descritto nella *URL* dell'elemento *target*.

L'attributo *fileName* è obbligatorio, mentre l'attributo *fileSystemName* è opzionale.

2.5.1.2.3 Le librerie necessarie al processo ASSIST (elemento lib)

Il file XML contiene nessuno, uno o più elementi *lib* con le informazioni sulle librerie necessarie al processo durante l'esecuzione.
Ogni processo ASSIST attualmente ha bisogno di due librerie per funzionare correttamente: la libreria ACE e la libreria XML2.

```
<aldl:requirement name="ND000__generaConfiguration">
 ... ... ...
 <ns1:lib    arch="i686"   fileName="libACE.so.5.3.0" >
   <ns1:source
         url="file:////usr/local/ACE_wrappers/ace/libACE.so.5.3.0"/>
 </ns1:lib>
 <ns1:lib    arch="i686"   fileName="libxml2.so.2"  >
   <ns1:source  url="file:////tmp/libxml2.so.2.6.16"/>
 </ns1:lib>
 ... ... ...
</aldl:requirement>
```

Figura 26. Sezione requirement: elemento lib.

Come si vede dalla figura 26, le librerie da trasferire sono contrassegnate dall'etichetta *source*.
Il trasferimento di una libreria avviene copiando il file descritto dalla *URL* nella macchina remota rinominandola con il nome scritto nell'attributo *fileName*. Il path di destinazione viene scelto dal Loader. L'attributo *fileName* è obbligatorio.

Anche per le librerie è necessario specificare il tipo di architettura a cui si riferiscono, poiché le librerie per linux e per macintosh solitamente sono completamente diverse. L'attributo dell'architettura (*arch*) è opzionale per l'elemento lib ed è settato di default al valore *all*, questo

significa che la libreria da trasferire è indifferente al tipo di architettura.

Oltre a queste informazioni possiamo trovare altri due tipi di attributi all'interno dell'elemento *lib*.

- L'attributo *executable* (figura 27) può assumere due valori (“yes” e “no”) ed indica se la libreria da trasferire deve anche essere lanciata.
  L'attributo è opzionale e di default vale “no”.

- L'attributo *shared* (figura 28) può assumere due valori (“yes” e “no”) ed indica se, in caso di NFS, è necessario trasferire la libreria oppure se lo si può considerare condivisa.
  L'attributo è opzionale e di default vale “no”.
  A seconda del valore dell'attributo shared e degli identificatori della macchina di partenza (descritto nel file userconfig.xml) e della macchina/cluster di arrivo (descritto nel file cluster.xml, paragrafo 2.2.1) possiamo stabilire se la libreria sarà copiata oppure no (tabella 29).

```
<aldl:requirement name="ND000__generaConfiguration">
 ... ... ...
 <ns1:lib   arch="i686"   fileName="libex.so"   executable="yes">
   <ns1:source
        url="file:////usr/bin-libex.so"/>
 </ns1:lib>
 ... ... ..
</aldl:requirement>
```

Figura 27. Attributo *executable* nell'elemento *lib*.

```
<aldl:requirement name="ND000__generaConfiguration">
 ... ... ...
 <ns1:lib   arch="i686"   shared="yes"   fileName="libACE.so.5.3.0">
   <ns1:source
        url="file:////home/pacifico/potiti/libACE.so.5.3.0"/>
 </ns1:lib>
 ... ... ...
</aldl:requirement>
```

Figura 28. Attributo *shared* nell'elemento *lib*.

Per decidere il trasferimento di una libreria (o di un file di ingresso) è necessario valutare tre parametri:

1. l'identificatore dell'NFS della macchina sorgente e della macchina destinazione,
2. l'attributo shared della libreria,
3. il path dove si trova in origine la libreria ed il path dove deve essere trasferito.

- Se la libreria *non* è *shared*, significa che la libreria non è condivisa, di conseguenza la libreria viene copiata sempre, anche se la macchina destinazione fa parte di un cluster la libreria viene copiata su ogni macchina del cluster utilizzata.

- Se la libreria è *shared* allora bisogna analizzare il path di origine ed il path di destinazione della libreria:
    - Se i *path* sono *diversi* significa che nonostante la libreria sia condivisa si vuole spostare la libreria da un path ad un altro, di conseguenza la libreria deve essere copiata sulla macchina destinazione, ma nel caso in cui la macchina faccia parte di un cluster, allora la libreria viene copiata una sola volta per cluster, dato che è una libreria *shared*.
    - Se i *path* sono *uguali*, allora bisogna analizzare gli identificatori della macchina sorgente e della macchina di destinazione:

        - Se gli *identificatori* sono *diversi* allora la macchina di partenza e quella di destinazione non hanno un Network FileSystem, di conseguenza la libreria deve essere copiata sulla macchina destinazione, ma nel caso in cui la macchina faccia parte di un cluster, allora la libreria viene copiata una sola volta per cluster, dato che è una libreria *shared*.
        - Se gli *identificatori* sono *uguali*, allora la macchina di partenza e quella di destinazione hanno un NFS, di conseguenza la libreria non deve essere copiata sulla macchina destinazione, in quanto è una libreria shared ed il path di origine è lo stesso di quello di destinazione.

Nella tabella 29 troviamo una descrizione dei casi che si possono presentare.

ID NFS	***Lib shared***	***Lib source*** ***Lib dest***	***Copia***
X	no	X	N copie
=	si	=	no
!=	si	=	1 copia x cluster
X	si	!=	1 copia x cluster

Tabella 29. Illustrazione dei casi in cui è necessario copiare i file di input e le librerie utilizzate da un programma ASSIST su una macchina destinazione.

2.5.1.4 Processi hoc associati al processo ASSIST (elemento hoc)

Per ogni requirement all'interno del file ALDL possiamo trovare un elemento *hoc* che contiene le informazioni sul processo hoc da lanciare prima del processo ASSIST. Nel caso in cui il requirement descriva un VPM, è possibile che i processi hoc da lanciare siano più di uno.
I processi hoc da lanciare si dividono in due categorie: i processi hoc “necessari” ed i processi hoc di tipo “bridge”. I processi hoc di tipo bridge servono unicamente per collegare un processo che viene lanciato su una macchina dove non è stato lanciato nessun hoc “necessario” agli altri hoc.

```
<aldl:requirement name="ND000__generaConfiguration">
 ... ... ...
  <ns1:hoc  nHocTot = "3"  prefixAlias = "ND000__seq_init__"
          hocExName = "hoc"  hocConfName = "hoc.conf"  bridge = "yes"  arch = "all">
    <ns1:source  urlExHoc = "file:////l/disc1/hoc" urlConfHoc = "file:////tmp/comp/bin/hoc.conf"/>
  </ns1:hoc>
</aldl:requirement>
... ... ...
<aldl:requirement name = "ND002__somma_sharedConfigurationVpm">
   <ns1:parallelism minPar = "1" maxPar = "2"/>
   ... ... ...
   <ns1:hoc  nHocTot = "3"  nHoc= "2"  prefixAlias =  “ND002_somma_shared_Req__"
          hocExName = "hoc"  hocConName = "hoc.conf"   bridge = "no"  arch = "all">
    <ns1:source  urlExHoc = "file:////l/disc1/hoc" urlConfHoc = "file:////tmp/comp/bin/hoc.conf"/>
   </ns1:hoc>
 </aldl:requirement>
```

Figura 30. Sezione requirement: elemento hoc.

All'interno dell'elemento hoc troviamo diverse informazioni:

- *nHocTot* indica il numero dei processi hoc “necessari” per tutta l'applicazione Assist. Questa informazione è ripetuta, sempre uguale, in tutti gli elementi di tipo hoc.
- *nHoc* indica il numero di processi hoc “necessari” per il processo ASSIST descritto nel requirement. Questa informazione è presente solo quando il processo ASSIST ha bisogno di hoc “necessari”.
- *prefixAlias* indica il prefisso da passare a riga di comando quando si lancia un processo hoc legato al processo descritto nel requirement.
- *hocExName* indica il nome dell'eseguibile del processo hoc.
- *hocConfName* indica il nome del file di configurazione del processo hoc.

- *bridge* indica se il processo hoc da lanciare è un hoc “necessario” (allora bridge vale *no*), o se il processo hoc da lanciare è di tipo bridge (allora bridge vale *yes*). Nel caso in cui bridge valga yes non deve essere specificato l'attributo nHoc; mentre nel caso in cui bridge valga no deve essere specificato l'attributo nHoc.
- *arch* indica il tipo dell'architettura per cui l'eseguibile è utilizzabile (i686/powerpc/all).
- all'interno del sottoelemento *source* troviamo i due attributi *urlExHoc* ed *urlConfHoc* che indicano rispettivamente la URL dove si trova l'eseguibile e la URL dove si trova il file di configurazione del processo hoc.

2.5.1.5 Grado di parallelismo del processo ASSIST (elemento parallelism)

L'elemento *parallelism* si può trovare all'interno di un requirement che riguarda un processo VPM ed indica il numero di VPM da lanciare. Questo numero viene scelto dal compilatore ASSIST in modo opportuno.
Come si vede dalla figura 31, l'elemento parallelism ha due attributi: *minPar* opzionale poiché di default vale 1, e *maxPar* obbligatorio.
Attualmente il Loader lancia esattamente maxPar processi.

```
<aldl:requirement name="ND001__p_array1ConfigurationVpm">
  <ns1:parallelism minPar="1" maxPar="4"/>
  ... ... ...
</aldl:requirement>
```

Figura 31. Sezione requirement: elemento parallelism.

2.5.1.6 File contenenti lo standard output e lo standard error del processo ASSIST (elementi stdout e stderr) (Funzionalita’ da verificare per la versione 1.1)

Gli elementi *stdout* e *stderr* indicano i path assoluti (sulla risorsa remota) dei file su cui salvare rispettivamente lo standard output e lo standard error del processo descritto nel requirement.
Per la presente versione e’ stata testata soltanto la redirezione dello Standard Output su file nell’uso di ASAP SSH.

```
<aldl:requirement name="ND000__generaConfiguration">
  ... ... ...
  <ns1:stderr>genera.err</ns1:stderr>
  <ns1:stdout>genera.out</ns1:stdout>
... ... ...
</aldl:requirement>
```

Figura 32. Sezione requirement: elementi stdout e stderr.

2.5.1.7 Caratteristiche della macchina su cui lanciare il processo ASSIST (elemento cpu e physicalmemory)

Gli elementi *cpu* e *physhicalMemory* indicano le caratteristiche della potenza di calcolo e della memoria fisica delle risorse da utilizzare.

```
<aldl:requirement name="ND000__generaConfiguration">
  ... ... ...
  <ns1:cpu platform="i686" vendor="Intel" clockSpeed="500" loadAvg15="0.1"/>
  <ns1:physicalMemory maxSize="128" freeSize="64"/>
... ... ...
</aldl:requirement>
```

Figura 33. Sezione requirement: elementi cpu e physicalmemory.

L'elemento *cpu* può contenere quattro diversi attributi:

- l'attributo *vendor* indica il tipo di macchina (Intel).
- l'attributo *platform* indica il tipo di architettura della macchina (i686 o powerpc).
- l'attributo *clockSpeed* indica la velocità di clock della macchina.
- l'attributo *loadAvg15* rappresenta la frazione di CPU minima disponibile per un nuovo processo (informazione dinamica, che per adesso è statica.....).

Allo stesso modo, l'elemento *physhicalMemory* può contenere due diversi attributi:

- l'attributo *maxSize* indica la dimensione della memoria della macchina.
- l'attributo *freeSize* indica la dimensione della memoria libera della macchina.

Tutti questi attributi sono opzionali.

2.5.1.8 IP della macchina su cui lanciare il processo ASSIST (elemento netaddress e netmask)

Gli elementi *netAddress* e *netMask* indicano l'indirizzo IP della macchina (o del gruppo di macchine) su cui lanciare il processo ASSIST.

```
<aldl:requirement name="ND000__generaConfiguration">
  ... ... ...
  <ns1:netAddress      value="131.114.0.0"/>
  <ns1:netMask         value="255.255.0.0"/>
</aldl:requirement>

<aldl:requirement name="ND002__fineConfiguration">
  ... ... ...
  <ns1:netAddress      value="131.114.2.110"/>
  <ns1:netMask         value="255.255.255.255"/>
</aldl:requirement>
```

Figura 34. Sezione requirement: elementi netAddress e netMask.

2.5.2 Esempi di requirement di sottoinsiemi di caratteristiche del processo ASSIST

2.5.2.1 Requirement con descrizione di una macchina

Nella figura 35 viene riportato il codice del requirement che descrive la macchina pegaso in termini di IP (netAddress) e di maschera (netMask) e nella figura 36 si vede come possiamo utilizzare il requirement nella sezione module.

```
<aldl:requirement      name="pegaso">
  <ns1:netAddress      value="131.114.2.110"/>
  <ns1:netMask         value="255.255.255.255"/>
</aldl:requirement>
```

Figura 35. Requirement con descrizione di una macchina.

```
<aldl:module name="ND000__generaMod">
  <aldl:requires name="tns:ND000__generaConfiguration"/>
  <aldl:requires name="tns:pegaso"/>
</aldl:module>
```

Figura 36. Utilizzo del requirement con descrizione di una macchina.

2.5.2.2 Requirement con descrizione di una sottorete di macchine

Nella figura 37 viene riportato il codice del requirement che descrive la sottorete di macchine netDI in termini di IP (netAddress) e di maschera (netMask) e nella figura 38 si vede come possiamo utilizzare il requirement nella sezione module.

```
<aldl:requirement     name="netDI">
  <ns1:netAddress     value="131.114.0.0"/>
  <ns1:netMask        value="255.255.0.0"/>
</aldl:requirement>
```

Figura 37. Requirement con descrizione di una sottorete di macchine.

```
<aldl:module name="ND000__generaMod">
  <aldl:requires name="tns:ND000__generaConfiguration"/>
  <aldl:requires name="tns:pegaso"/>
</aldl:module>
```

Figura 38. Utilizzo del requirement con descrizione di una sottorete di macchine.

2.5.2.3 Requirement con descrizione di un gruppo di librerie

Nella figura 39 viene riportato il codice del requirement che descrive un gruppo di librerie (libraries) e nella figura 40 si vede come possiamo utilizzare il requirement nella sezione module.

```
<aldl:requirement name = "libraries">
  <ns1:lib   fileName = "libACE.so.5.4.0"   arch = "i686"   executable = "no">
    <ns1:source   url = "file:////usr/local/ACE_wrappers/ace/libACE.so.5.4.0"/>
  </ns1:lib>
  <ns1:lib   fileName = "libxml2.so.2"   arch = "i686"   executable = "no">
    <ns1:source   url = "file:////usr/local/libxml2/lib//libxml2.so.2.6.17"/>
  </ns1:lib>
</aldl:requirement>
```

Figura 39. Requirement con descrizione di un gruppo di librerie.

```
<aldl:module          name = "ND002__fineMod">
  <aldl:requires      name = "tns:ND002__fineConfiguration"/>
  <aldl:requires      name = "tns:libraries"/>
</aldl:module>
```

Figura 40. Utilizzo del requirement con descrizione di un gruppo di librerie.

2.5.3 I gruppi dell'Applicazione ASSIST

Le informazioni specificate nella sezione dei gruppi sono di due tipi:

1. Le informazioni riguardanti un gruppo di istanze che fanno parte di un parmod (*gruppo di tipo parmod*).
2. Le informazioni riguardanti un gruppo di istanze che devono essere coallocate secondo il modello di sincronizzazione del supporto a runtime di ASSIST (*gruppo di tipo coallocazione*).

In figura 41 è riportata la sezione di codice che descrive un gruppo di tipo parmod, l'elemento ns1:parmod infatti contiene i nomi dei processi ism, vpm ed osm che fanno parte del parmod descritto.
L'informazione sui gruppi di parmod viene utilizzata nella sezione dei moduli (paragrafo 2.5.4).

```
<aldl:group   name = "ND001__p_array1Parmod">
  <ns1:parmod   ism = "ND001__p_array1ConfigurationIsm"
                vpm = "ND001__p_array1ConfigurationVpm"
                osm = "ND001__p_array1ConfigurationOsm"/>
 </aldl:group>
```

Figura 41. Sezione group: elemento parmod.

In figura 42 è riportata la sezione di codice che descrive un gruppo di tipo coallocazione.

Attualmente tutti i processi di un'applicazione ASSIST devono essere coallocati.
L'informazione sui gruppi di coallocazione viene utilizzata nella sezione delle instanze (paragrafo 2.5.5).

```
<aldl:group   name = "coallocation">
  <ns1:assist-coallocate/>
</aldl:group>
```

Figura 42. Sezione group: elemento assist-coallocate.

2.5.4 I moduli dell'Applicazione ASSIST

La sezione *module* serve per associare ad ogni processo descritto nella sezione dei requirement le informazioni sui gruppi di parmod e sulle macchine descritte nella sezione dei requirement.
Infatti all'interno dell'elemento module troviamo l'elemento requires che identifica un requirement. Ogni elemento module deve avere al suo interno un elemento requirement che descrive processo ASSIST (figura 43); inoltre può contenere un requirement che descrive delle macchine (figure 35, 40 e 44) ed un elemento memberOf che identifica il gruppo di tipo parmod a cui appartiene il processo (figura 45).
L'informazione sui moduli viene utilizzata nella sezione instance (paragrafo 2.5.5).

```
<aldl:module   name = "ND000__generaMod">
  <aldl:requires   name = "tns:ND000__generaConfiguration"/>
</aldl:module>
```

Figura 43. Sezione module di un processo sequenziale.

```
<aldl:module          name = "ND000__generaMod">
  <aldl:requires      name = "tns:ND000__generaConfiguration"/>
  <aldl:requires      name = "tns:netDI"/>
</aldl:module>

<aldl:module          name = "ND002__fineMod">
  <aldl:requires      name = "tns:ND002__fineConfiguration"/>
  <aldl:requires      name = "tns:pegaso"/>
</aldl:module>
```

Figura 44. Sezione module di due processi sequenziali con richiesta di esecuzione sulle macchine netDi e sulla macchina pegaso.

```
<aldl:module   name = "ND001__p_array1ModIsm">
  <aldl:requires   name = "tns:ND001__p_array1ConfigurationIsm"/>
  <aldl:memberOf   name = "tns:ND001__p_array1Parmod"/>
</aldl:module>
```

Figura 45. Sezione module di un processo che fa parte di un parmod.

2.5.5 La sezione instance

Nella sezione *instance* viene descritta l'associazione tra il modulo e il gruppo di coallocazione.
In figura 46, ad esempio, viene descritta l'istanza ND000__generaIns che contiene il modulo ND000__generaMod definito nella sezione dei moduli ed il gruppo di coallocazione coallocation definito nella sezione dei moduli.

```
<aldl:instance   name = "ND000__generaIns">
  <aldl:instanceOf   name = "tns:ND000__generaMod"/>
  <aldl:memberOf   name = "tns:coallocation"/>
</aldl:instance>
```

Figura 46. Sezione instance.

2.6 Il file initialize.xml della componente ASSIST

Il file *initialize.xml* contiene le indicazioni per inizializzare una componente.
Nel caso in cui la componente sia una componente Assist (unico caso attualmente implementato)nel file initialize.xml troviamo le informazioni sugli stream da registrare sul processo hocIndice e sui processi hocStream da lanciare per connettere la componente alle altre componenti.

```
<?xml version="1.0" ?>
<Communicators>
 <Streams>
  <Stream name="s1" />
  <Stream name="s2" />
 </Streams>
 <Events>
  <Event name="e1" />
 </Events>
 <RPCs>
  <RPC name="r1" />
 </RPC>
 <Communicator>
  <HOC config = "hocMIO.conf" >
   <Stream name = "s1" />
   <Event name = "e2" />
   <RPC name = "r4" />
  </HOC>
  <HOC>
   <Stream name = "s2" />
   <Stream name = "s3" />
   <Stream name = "s4" />
  </HOC>
  <HOC>
  </HOC>
 </Communicator>
</Communicators>
```

Figura 47. File XML initialize.xml.

Nel file XML, illustrato in figura 47, troviamo due tipi di informazioni:

- Gli elementi *Streams*, *Events* e *RPCs* contengono una lista di nomi rispettivamente di stream, eventi e RPC da registrare sul processo hocIndice. Questi canali di comunicazione sono gli stream, gli eventi e gli RPC non funzionali, come lo stream di terminazione...

- L'elemento *Communicator* descrive i processi hocStream che il Loader deve lanciare specificando eventualmente il nome del file di configurazione (*hocMIO.conf*) da utilizzare. Nel caso in cui il file di configurazione non sia specificato viene utilizzato il file di configurazione di default contenuto nel file XML *gateway.xml*. Anche il file eseguibile dei processi hocStream viene specificato nel file XML *gateway.xml* (paragrafo 2.2.2).
  La libreria *streamlib3* connetterà successivamente gli stream, gli eventi e gli RPC associati a ciascun processo hocStream con l'hocStream stesso.

Appendice1 : la libreria di memoria condivisa HOC

Vedere il file HOWTO_it.txt nella directory ADHOC.

Appendice 2: Bug e funzionalita’ ancora da implementare

Appendice 2.1: Bug libreria j2ssh

La libreria j2ssh è la libreria utilizzata dal Loader per eseguire lo stage ed il run dei processi tramite SSH.

Utilizzando la libreria j2ssh abbiamo riscontrato due bug:

1. Nel caso in cui i processi lanciati tramite Loader producano un gran numero di dati sullo standard output o sullo standard error, è possibile che l'applicazione si blocchi, perché la libreria non è piu in grado di scrivere i dati ricevuti dai processi in un suo buffer. Di conseguenza viene lanciata e catturata un'eccezione all'interno della libreria.
   Una possibile soluzione a questo bug è lo redirezione dello standard output e dello standard error su file. La redirezione su file è possibile per tutti i processi ASSIST all'infuori dei processi VPM. Per redirigere lo standard output e lo standard error su file è necessario inserire nel file XML che descrive l'applicazione ASSIST le righe riportate in figura 48.

```
<aldl:requirement name="ND000__generaConfiguration">
 ... ... ...
<ns1:stdout>/tmp/LogOUT_genera</ns1:stdout>
<ns1:stderr>/tmp/LogERR_genera</ns1:stderr>
</aldl:requirement>
```

Figura 48. Redirezione dello stdout e stderr nella sezione dei *requirement*.

2. Nel caso in cui viene chiuso il socket della connessione ssh, ad esempio perché abbiamo eseguito il comando KillProcess, la libreria non si accorge che il socket è stato chiuso e tenta di ricevere nuovi dati dal socket, di conseguenza viene lanciata e catturata l'eccezione riportata in figura 49. Per questo bug esiste una patch, ma onde evitare di renderci incompatibili con le future versioni della libreria abbiamo deciso di non introdurla.

3. Questo bug non determina uno scorretto funzionamento del Loader.

```
12746 [Transport protocol 12] (?:?)
ERROR com.sshtools.j2ssh.transport.TransportProtocolCommon
- The Transport Protocol thread failed
java.io.IOException: The socket is EOF
        at com.sshtools.j2ssh.transport.TransportProtocolInputStream.readBufferedData(Unknown Source)
        at com.sshtools.j2ssh.transport.TransportProtocolInputStream.readMessage(Unknown Source)
        at com.sshtools.j2ssh.transport.TransportProtocolCommon.processMessages(Unknown Source)
        at com.sshtools.j2ssh.transport.TransportProtocolCommon.startBinaryPacketProtocol(Unknown Source)
        at com.sshtools.j2ssh.transport.TransportProtocolCommon.run(Unknown Source)
        at java.lang.Thread.run(Thread.java:534)
```

Figura 49. Eccezione lanciata in caso di chiusura del socket.

Appendice 3: Creazione di un archivio AAR per una applicazione ASSIST

Mediante lo script aarer.sh e' possibile creare un archivio in formato AAR per una applicazione ASSIST contenente tutti i file necessari all'esecuzione.
Per creare questo archivio e' sufficiente posizionarsi sulla directory dove e' presente lo script e utilizzare il comando

aarer.sh [directory Compilazione ASSIST] [nomeFileAAR]

dove:

directory Compilazione ASSIST indica la directory dove vengono creati i file dal compilatore ASSIST (e quindi dove si trovano i file ast.out.xml e ast.out.component.xml) e *fileAAR* indica il nome del file (eventualmente completo di path assoluto) da creare.

Esempio

[pascucci@orione bin]$ ./aarer.sh ~/Assist/compiledAssist/ prova.aar

Eventuali errori sul fatto che non trovi il file hoc.conf nella directory di compilazione dell'applicazione ASSIST e' dovuto al fatto che si sta tentando ci creare un archivio per una applicazione e non per una componente ASSIST.