Capitolo 7

Architettura software del sistema

In questo capitolo viene descritta l'architettura software del sistema, definendo le caratteristiche e le funzionalità dei vari moduli, e le interazioni che esistono tra essi. Vengono presentati solo i moduli e le procedure più importanti, mentre non vengono trattati dettagli implementativi poco significativi, o procedure per le quali non si ritiene opportuna un'approfondita analisi (ad esempio le operazioni sui database).

Informazioni più dettagliate si possono comunque trovare in [1], nel documento in cui si descrive l'interfaccia delle varie classi.

7.1 Struttura del sistema

In figura 7.1 è rappresentata l'architettura del sistema DEMI, ad un primo livello di astrazione.

A livello globale, quindi, il sistema DEMI si compone di due moduli fondamentali: il **Global Interface** e il **DEMI Global Control System**. Il Global Interface viene utilizzato dai global administrator per interagire con il sistema: i comandi specificati vengono elaborati dal DEMI Global Control System, che "torna" poi all'amministratore i corrispondenti risultati.

Il DEMI Global Control System, nel corso della sua attività, accede ai database globali, sia per operazioni di lettura, che per operazioni di scrittura.

Per quel che riguarda l'architettura del sistema a livello locale, invece, si devono considerare altri due moduli: il **Local Interface** e il **DEMI Local Control System**. Gli utenti del sito locale (local administrator compresi) interagiscono con il sistema attraverso il Local Interface, e i comandi specificati vengono elaborati dal DEMI Local Control System, che può accedere sia ai database locali che ai database globali.

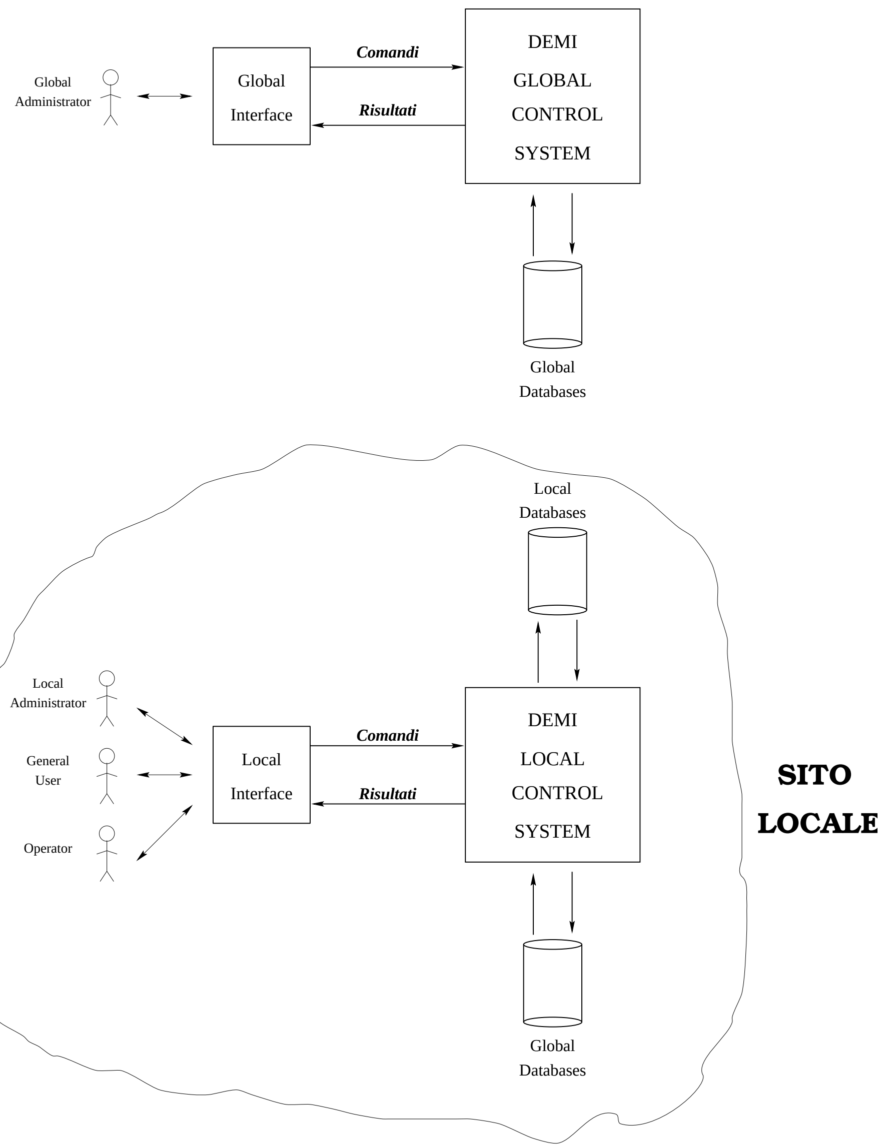

Figura 7.1: Architettura del sistema DEMI

Nei prossimi paragrafi vengono descritte le funzionalità di questi moduli.

7.2 Global Interface

Le operazioni che un global administrator può effettuare tramite il Global Interface sono:

- la modifica della propria password;
- la gestione del Global Administrator Database (definizione di nuovi amministratori, cancellazione di amministratori, operazioni di modifica, ricerca, visualizzazione);
- la gestione dell'Architecture Database (inserimento di nuovi elementi, cancellazione di architetture, modifiche, ricerche, visualizzazioni);
- la gestione del Tape Device Model Database (operazioni di inserimento, cancellazione, modifica, ricerca, visualizzazione);
- operazioni di ricerca e visualizzazione sul Data Database;
- operazioni di ricerca e visualizzazione sul Processing Database.

7.3 DEMI Global Control System

Il DEMI Global Control System elabora i comandi specificati dal global administrator. Si tratta di operazioni di inserimento, cancellazione, modifica, ricerca e visualizzazione effettuate sui database globali: per la descrizione di queste procedure si rimanda ai documenti presenti in [1].

Quando sarà realizzata la versione finale del sistema, nel DEMI Global Control System sarà implementato anche il Global Scheduler, che dovrà gestire lo scheduling delle varie elementary request, considerando un ambiente di calcolo distribuito geograficamente tra più siti.

7.4 Local Interface

Le operazioni che un utente può effettuare attraverso il Local Interface dipendono dal suo "ruolo" (un utente può essere un local administrator, e/o un utente generico, e/o un operatore).
Un local administrator può:

- modificare la sua password;
- gestire lo User Database (per operazioni di inserimento, cancellazione, modifica, ricerca, visualizzazione);
- effettuare operazioni di ricerca e visualizzazione sull'Architecture Database;
- effettuare operazioni di ricerca e visualizzazione sul Tape Device Model Database;
- gestire il Local Computer Database (per definire nuovi computer, per eliminare dal database calcolatori, per disabilitare e riabilitare computer, per modificare informazioni relative ad elementi definiti nel database, per operazioni di ricerca e visualizzazione);
- gestire il Local Buffer Database (per operazioni di inserimento, cancellazione, modifica, ricerca, visualizzazione);
- gestire il Local Tape Device Database (per operazioni di inserimento, cancellazione, modifica, ricerca, visualizzazione).

Un utente generico può:

- modificare la sua password;
- operare sui tipi di task (per definirne di nuovi, per specificare nuove implementazioni per determinate architetture, per operazioni di cancellazione, modifica, ricerca, visualizzazione);
- operare sulle production (per definirne di nuove, per modificarle, per cancellarle);
- operare sulle computational request (per sottomettere nuove richieste al sistema, per verificare lo stato di esecuzione di una richiesta, per "abortire" l'esecuzione di una richiesta);
- eseguire operazioni di ricerca e visualizzazione:
  - sull'Architecture Database;
  - sul Tape Device Model Database;
  - sul Local Computer Database;
  - sul Local Buffer Database;

  – sul Local Tape Device Database;
  – sul Data Database;
  – sul Processing Database.

Infine un operatore può semplicemente:

- modificare la sua password;
- visualizzare lo stato dei tape device definiti nel Local Tape Device Database.

7.5 DEMI Local Control System

Il DEMI Local Control System gestisce i comandi specificati dagli utenti attraverso il Local Interface.

Non si esaminano qui tutte le funzionalità di questo modulo (per un'analisi completa si rimanda ai documenti presenti in [1]): in questo paragrafo e nel successivo viene considerata solamente la parte fondamentale del DEMI Local Control System, ovvero l'elaborazione delle richieste computazionali sottomesse dagli utenti.

In questo paragrafo vengono descritti, ad alto livello, i vari moduli del DEMI Local Control System, che vengono esaminati più in dettaglio nel paragrafo 7.6.

La parte del DEMI Local Control System che gestisce l'elaborazione delle richieste computazionali è rappresentata in figura 7.2.

Innanzittutto, quindi, ogni richiesta computazionale sottomessa dagli utenti viene scomposta in elementary request, da un particolare modulo chiamato **Request Decomposer**.

Si consideri, ad esempio, una richiesta computazionale che fa riferimento alla produzione *prod1* rappresentata in figura 7.3 (*inp1*, *inp2*, *inp3* e *inp6* sono data box di input, *out7*, *out8* e *out9* sono data box di output, *dat4* e *dat5* sono data box che rappresentano dati temporanei).

Agli input di tipo singleton *inp1*, *inp3* e *inp6* siano associati rispettivamente i singleton input file *sinfile1*, *sinfile3* e *sinfile6*, mentre all'input di tipo stream *inp2* sia associato lo stream input set rappresentato dalla lista di file *file1*, *file2*, ..., *filen*, che devono quindi essere elaborati. Il Request Decomposer scompone questa richiesta in n elementary request: *er-1*, *er-2*, ... *er-n*, come mostrato in figura 7.4. I file associati agli input *inp1*, *inp2*, *inp3* e *inp6* per queste n elementary request sono rispettivamente (*sinfile1*, *file1*, *sinfile3*, *sinfile6*), (*sinfile1*, *file2*, *sinfile3*, *sinfile6*), ..., (*sinfile1*, *filen*, *sinfile3*, *sinfile6*).

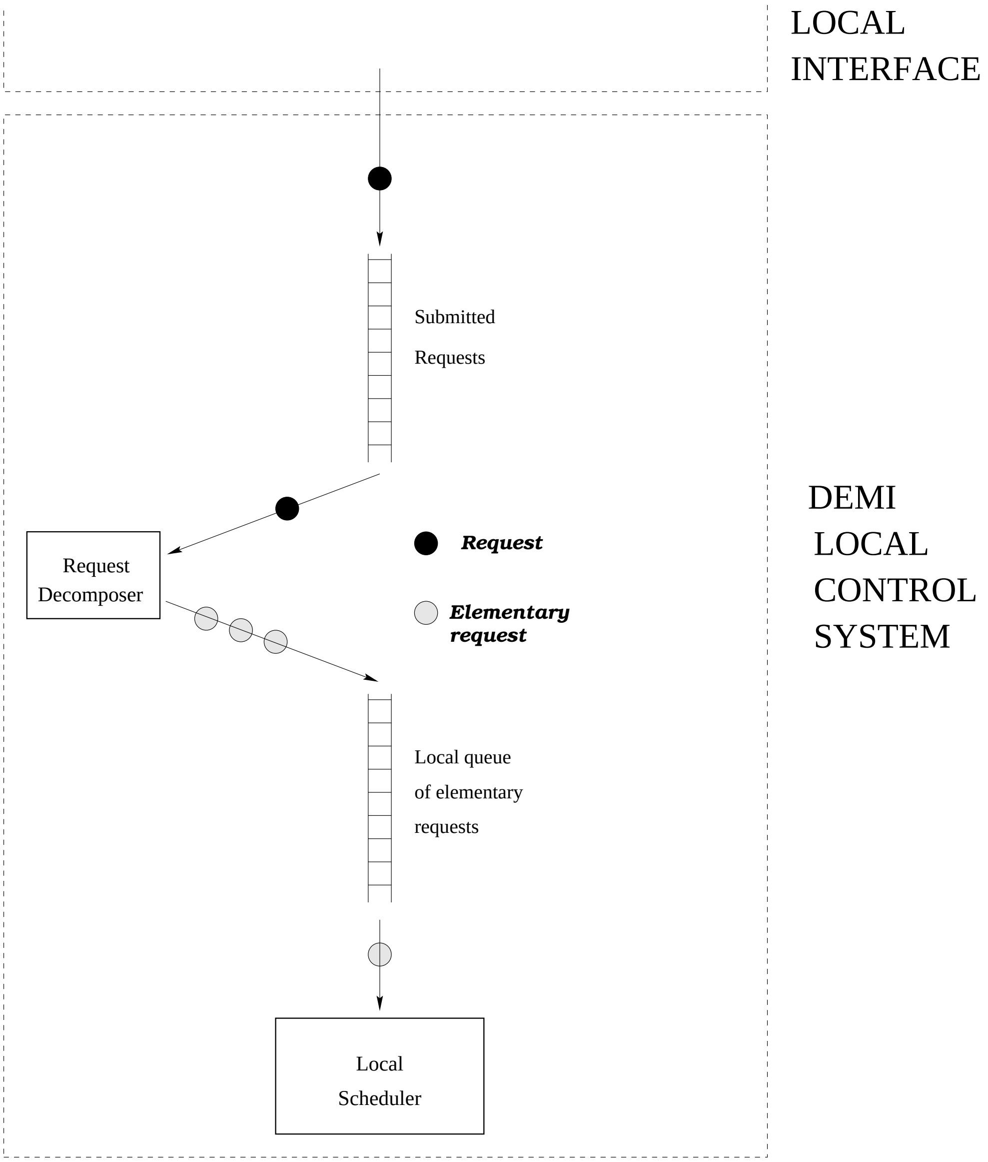

Figura 7.2: La gestione delle elaborazioni delle richieste computazionali

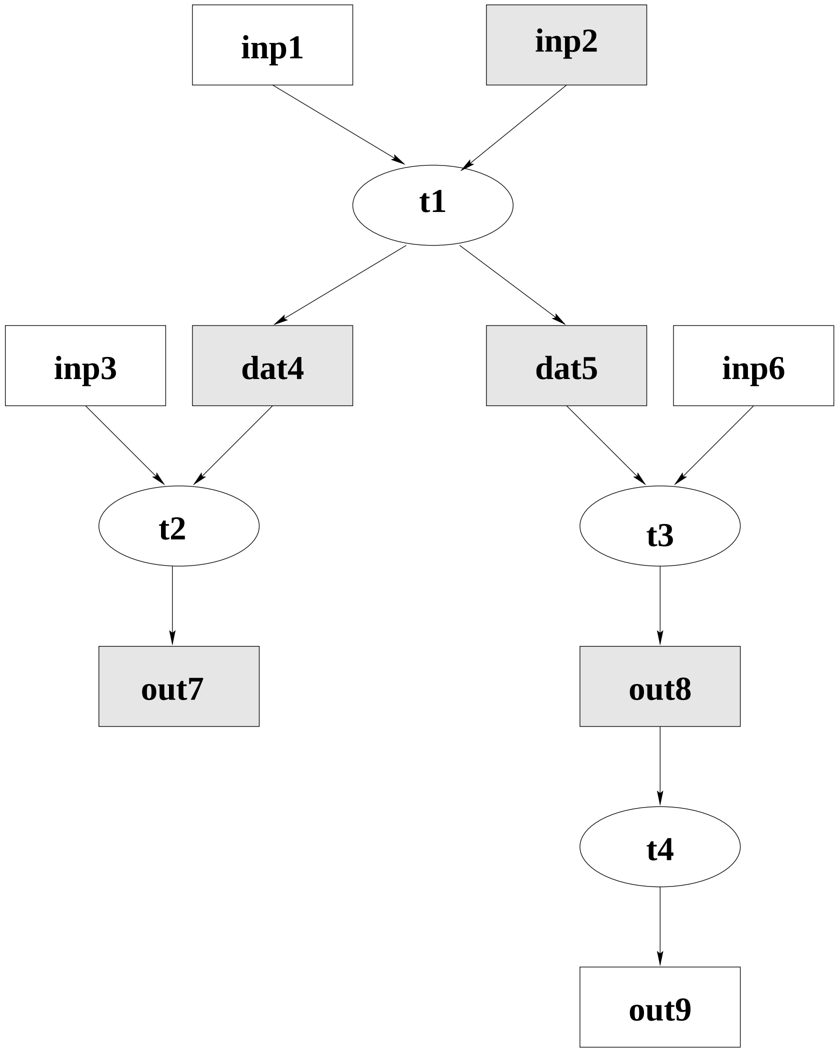

Figura 7.3: Produzione *prod1*

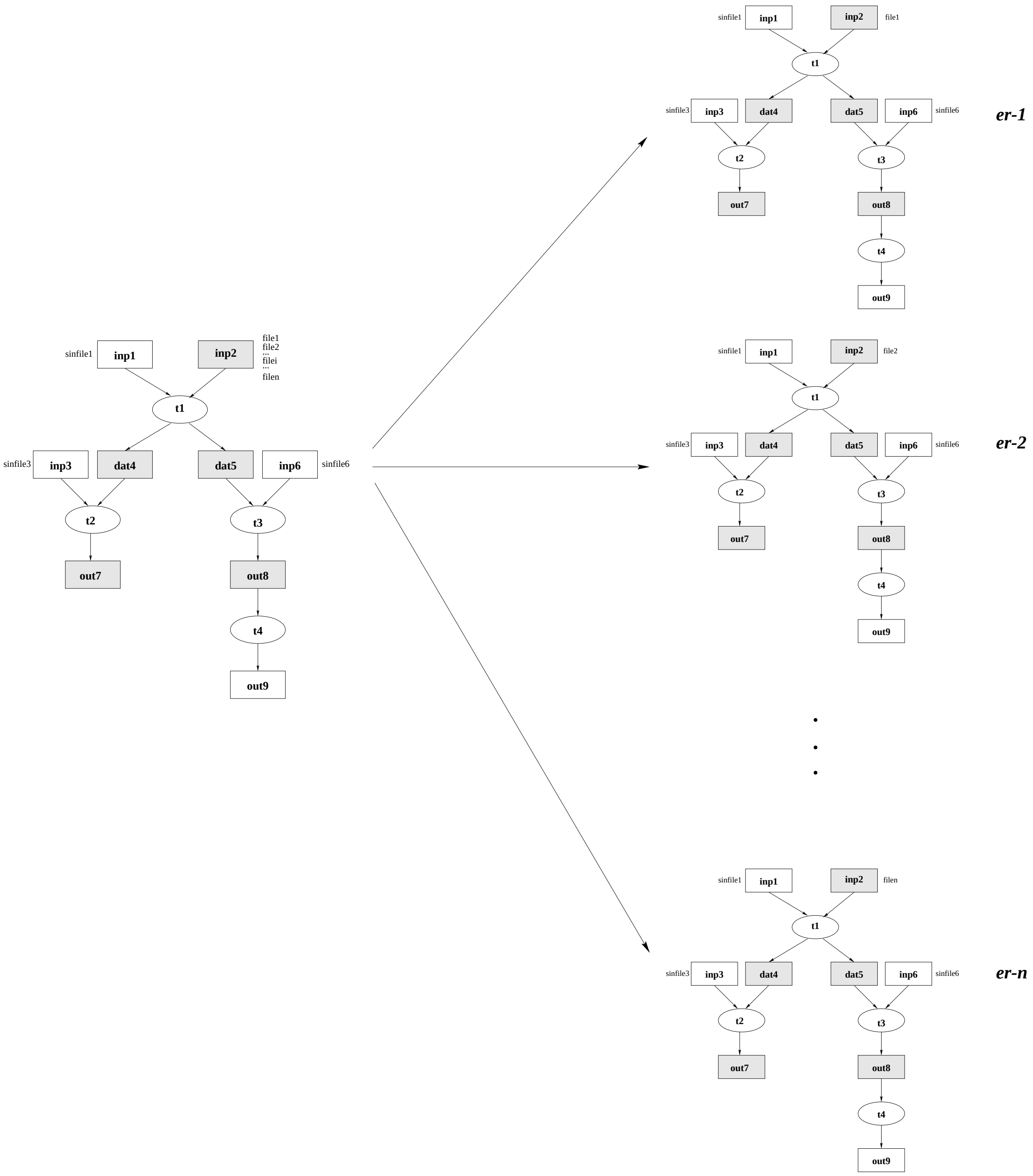

Figura 7.4: Scomposizione di una richiesta in elementary request, da parte del Request Decomposer

Se non ci fossero input di tipo stream, ogni elementary request corrisponderebbe ad un'iterazione (il numero di iterazioni deve essere stato specificato dall'utente che ha sottomesso la richiesta).

Le elementary request vengono inserite nella 'local queue of elementary requests', dove vengono poi esaminate, una alla volta (in base alla loro priorità), dal Local Scheduler.

Il **Local Scheduler** è il modulo fondamentale del DEMI Local Control System: questa "entità" è infatti responsabile della gestione delle elaborazioni delle elementary request, utilizzando le risorse di calcolo disponibili localmente.

L'architettura del Local Scheduler può essere schematizzata dal diagramma di figura 7.5.

Per ogni elementary request il Local Scheduler, prima di tutto, verifica se essa può essere effettivamente elaborata nel sito locale. L'elementary request viene poi esaminata da un particolare modulo, chiamato **Elementary Request Decomposer**, che la scompone nei task che costituiscono la corrispondente produzione.

Con riferimento all'esempio prima considerato, l'elementary request *er-i* viene scomposta in quattro task, *er-i-t1*, *er-i-t2*, *er-i-t3* e *er-i-t4*, associati rispettivamente ai task *t1*, *t2*, *t3* e *t4* della produzione *prod1*, come mostrato in figura 7.6.

Quindi, mentre il Request Decomposer scompone una richiesta in elementary request (ogni elementary request corrisponde ad un singolo stream input file dello stream input set o, se la corrispondente produzione non ha input di tipo stream, ad un'iterazione), l'Elementary Request Decomposer scompone ogni elementary request nei task che compongono la corrispondente produzione (se la produzione è costituita da n task, combinati opportunamente tra loro, l'elementary request viene scomposta in n task).

Il modulo **Task Starter** si occupa di verificare quali sono i task pronti per essere eseguiti (i task per cui sono disponibili tutti i relativi input).

Ad esempio, per l'elementary request *er-i*, il Task Starter rileva che il task *er-i-t1* può essere eseguito: è infatti l'unico task dell'elementary request *er-i* per cui sono disponibili tutti i relativi file di input (*sinfile1* e *filei*). Si sceglie un'architettura su cui eseguire il task, in base a vari parametri (le performance dei computer, il parametro 'cost of the task', i job già assegnati ai vari computer), e si sottomette il task al batch system, ovvero ad un sistema di load sharing in grado di gestire l'esecuzione di batch job, indicando che l'esecuzione deve avvenire su uno dei computer dell'architettura scelta. Prima però viene allocato sui disk buffer lo spazio per i file di output che saranno prodotti nell'esecuzione di questo task (nell'esempio considerato, quando il task *er-i-t1* deve essere eseguito, viene allocato lo spazio per la memorizzazione dei file associati ai data box *dat4* e *dat5*).

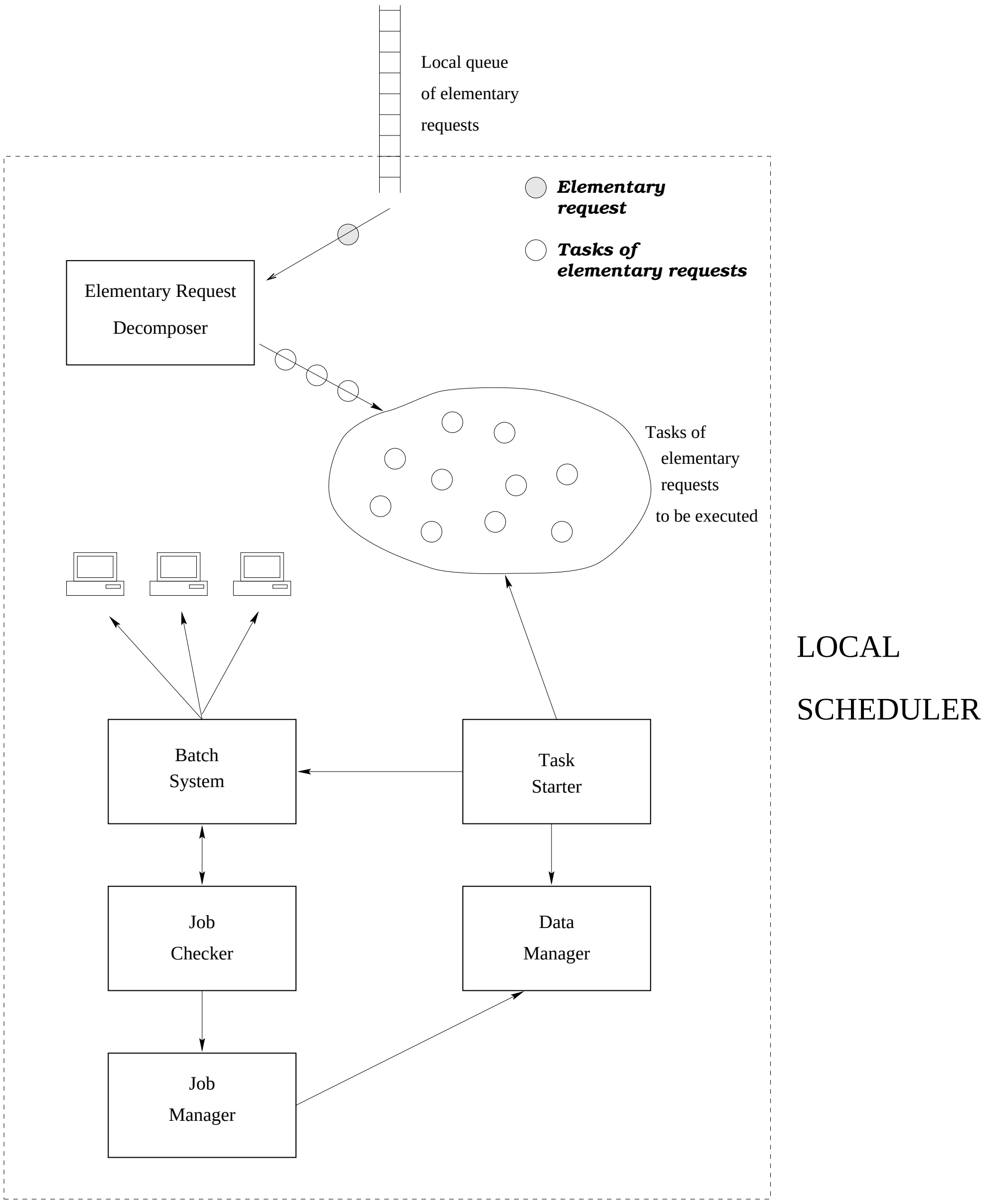

Figura 7.5: Architettura del Local Scheduler

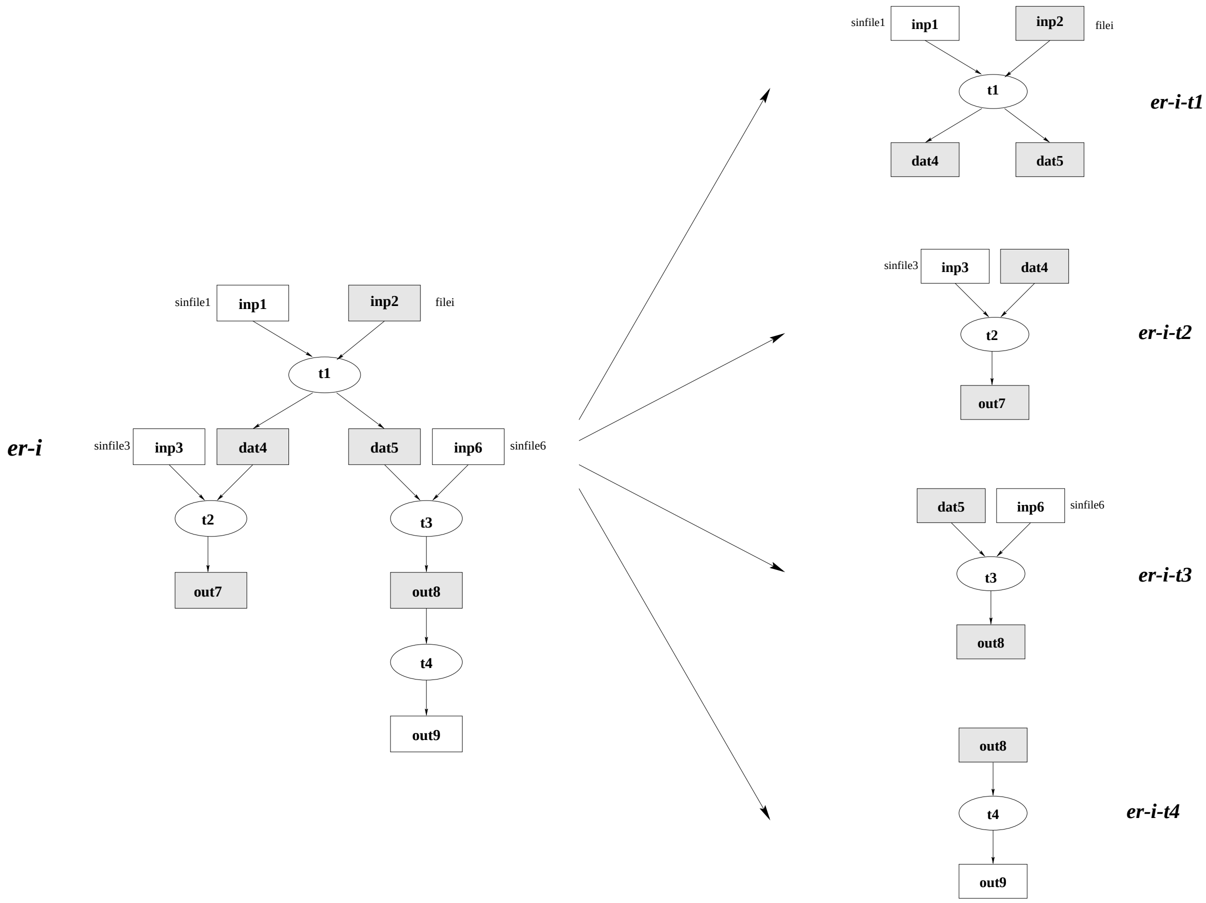

Figura 7.6: Scomposizione di una elementary request da parte dell'Elementary Request Decomposer

Ci si accerta che i file usati come input dal task (nel nostro caso i file *sinfile1* e *filei*) siano presenti su disco, ed eventualmente viene effettuata la copia da nastro.

È poi compito del batch system scegliere il computer che dovrà eseguire il task, in modo da ottimizzare l'uso complessivo delle varie risorse. Il batch system gestisce anche il suspend-resume dei job, secondo la configurazione considerata.

Un modulo chiamato **Job Checker** si "interfaccia" con il batch system, per verificare lo stato dei job DEMI. In particolare il Job Checker individua quando termina con successo l'esecuzione di un job, o quando l'esecuzione di un task fallisce. Il modulo **Job Manager** è responsabile nel gestire in maniera opportuna questi eventi.

Quando il Job Checker rileva che è terminata l'esecuzione di un task, si cancellano gli eventuali stream file "temporanei" non più necessari. Il Job Manager controlla poi se l'elaborazione dell'elementary request considerata è terminata, oppure se ci sono nuovi task da eseguire.

Sempre con riferimento all'elementary request *er-i* dell'esempio considerato, quando il task *er-i-t1* termina la sua esecuzione, il Task Starter sottomette al batch system per l'esecuzione i task *er-i-t2* e *er-i-t3*, per cui sono disponibili tutti i relativi file di input. Quando termina l'elaborazione del task *er-i-t2*, il file associato al data box *dat4* viene cancellato, perché non più necessario. Quando termina il processing del task *er-i-t3*, il file associato al data box *dat5* viene cancellato, e il Task Starter invoca l'esecuzione del task *er-i-t4*. Quando anche questo task termina la sua esecuzione, il processing dell'elementary request *er-i* è terminato, in quanto tutti i suoi task sono stati elaborati.

Quando l'elaborazione di un'elementary request è terminata, gli stream output file (quindi, nel nostro caso, i due file associati rispettivamente ai data box *out7* e *out8*), vengono registrati nel Data Database. Questi file di output vengono poi "spostati" dove richiesto dall'utente che ha sottomesso la richiesta computazionale (su nastro, su una directory privata, o mantenuti sui disk buffer).

Quando poi termina il processing di una richiesta, ovvero quando sono state elaborate tutte le sue elementary request, si analizzano i singleton file che sono stati prodotti: i file temporanei vengono cancellati, mentre i file di output vengono registrati nel Data Database, e spostati nel posto richiesto dall'utente (nell'esempio considerato l'unico singleton input file da considerare è quello associato al data box *out9*).

Se invece il Job Checker rileva che l'esecuzione di un task non è terminata con successo (ad esempio a causa di un crash del computer dove il job era in esecuzione), si invoca il Task Starter, per fare in modo che questo task venga rieseguito.

La gestione delle operazioni sui file di dati è affidata ad uno specifico modulo,

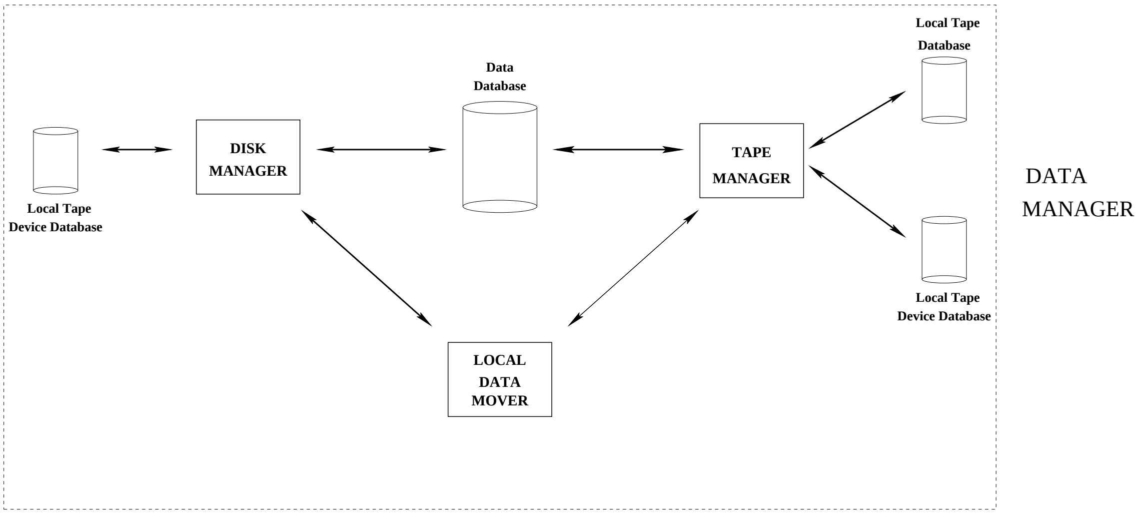

Figura 7.7: Architettura del Data Manager

chiamato **Data Manager**: le sue funzionalità vengono utilizzate sia dal Task Starter, che dal Job Manager.

L'architettura del Data Manager è rappresentata in figura 7.7.

Il Data Manager è costituito da tre moduli fondamentali: il **Disk Manager**, che gestisce le operazioni che riguardano i file memorizzati sui disk buffer, il **Tape Manager**, che si occupa delle procedure che coinvolgono i file di dati memorizzati su nastri, e il **Local Data Mover**, che gestisce le copie di file da disco a nastro, e viceversa.

7.6 Analisi del DEMI Local Control System

In questo paragrafo vengono esaminati più in dettaglio i moduli del DEMI Local Control System descritti nel paragrafo precedente, in particolare per analizzare le operazioni che essi devono gestire.

In questa descrizione si fa riferimento alle tabelle della base di dati relazione descritta nel capitolo 6, utilizzata per memorizzare in maniera persistente le varie informazioni.

7.6.1 Request Decomposer

Il seguente pseudo-codice descrive l'operazione di scomposizione delle richieste computazionali, effettuata dal Request Decomposer:

```
RequestDecomposer(req : Request);
begin
  prod = ProductionOfRequest(req);
  if (esiste inp input di tipo stream di prod)
    then for each f stream input file associato a inp
             do
             er = NewElementaryRequest();
             InsertIntoTable_StreamIOOfElementaryRequests(f);
             InsertIntoLocalQueue(er);  od
     else for i=1 to NumIt
             do
             er = NewElementaryRequest();
             InsertIntoLocalQueue(er);  od fi
end
```

Si esamina, quindi, se per la corrispondente production esiste un input di tipo stream (una produzione può avere al più un input di questo tipo). In caso positivo viene definita una elementary request per ogni stream input file associato a questo input. Se quindi per questo input si devono elaborare n file, vengono definiti nella tabella *ElementaryRequests* n elementi: ad ognuno di essi viene associato uno specifico identificatore (il campo 'ElemReqId'), e al campo 'Status' viene associato il valore 'Waiting to be processed'. Ognuno di questi stream input file viene definito nella tabella *StreamIOOfElementaryRequests*. Se invece la produzione non ha input di tipo stream, si definiscono un numero di elementary request pari al valore del parametro 'NumIt' della tabella *Requests* (il numero di iterazioni). In questo caso non viene definito nessun elemento nella tabella *StreamIOOfElementaryRequests*.

Le elementary request vengono poi inserite nella 'local queue of elementary requests', in modo che possano essere analizzate da parte del Local Scheduler.

Il Request Decomposer viene "attivato" ogni volta che un utente sottomette al sistema una nuova richiesta computazionale.

7.6.2 Local Scheduler

Il Local Scheduler "estrae" le elementary request dalla 'local queue of elementary requests' una alla volta: viene scelta l'elementary request con priorità più elevata (nel caso esistano più elementary request con stessa priorità, si sceglie quella che è stata inserita per prima nella coda).

Dopo aver verificato che l'elementary request può essere effettivamente elaborata nel sito locale (vedi par. 7.6.3), essa viene esaminata dall'Elementary Request Decomposer (vedi par. 7.6.4), che la scompone nei task che costituiscono la corrispondente produzione.

Il modulo Task Starter (vedi par. 7.6.5) si occupa di verificare quali sono i task pronti per essere eseguiti: per la loro esecuzione si utilizzano le funzionalità di un batch system.

Il Job Checker (vedi par. 7.6.7) si “interfaccia” con il batch system, per verificare lo stato dei job DEMI. In particolare individua quando inizia l'esecuzione di un task prima “sottomesso”, quando termina con successo l'esecuzione di un job, quando l'esecuzione di un task fallisce. Il Job Manager è responsabile nel gestire in maniera opportuna questi eventi (vedi par. 7.6.8).

7.6.3 Controlli preliminari sull'elementary request

La procedura che deve essere eseguita prima che un'elementary request venga scomposta dall'Elementary Request Decomposer può essere rappresentata dal flow chart di figura 7.8.

Si eseguono quindi dei controlli preliminari, per verificare che sia effettivamente possibile elaborare l'elementary request nel sito locale.
Si devono fare tre tipi di controlli:

1. verificare che i file di input della elementary request siano presenti nel sito locale;
2. verificare che nel sito siano presenti computer di tutte le architetture necessarie per l'esecuzione dei vari task;
3. verificare che nel sito siano presenti i tape device necessari per l'eventuale lettura/scrittura da/su nastro dei file di input/output di questa elementary request.

Se i test non hanno successo (e quindi non è possibile elaborare l'elementary request localmente):

- si avverte del problema l'utente che ha sottomesso la richiesta;
- si modifica l'attributo 'Status' della tabella *ElementaryRequests*, per l'elemento che rappresenta l'elementary request considerata, ponendo questo campo pari a 'Impossible to process';
- si controlla se si è terminato di elaborare la richiesta associata a questa elementary request e, in caso positivo, si gestisce opportunamente la situazione (vedi par. 7.6.10).

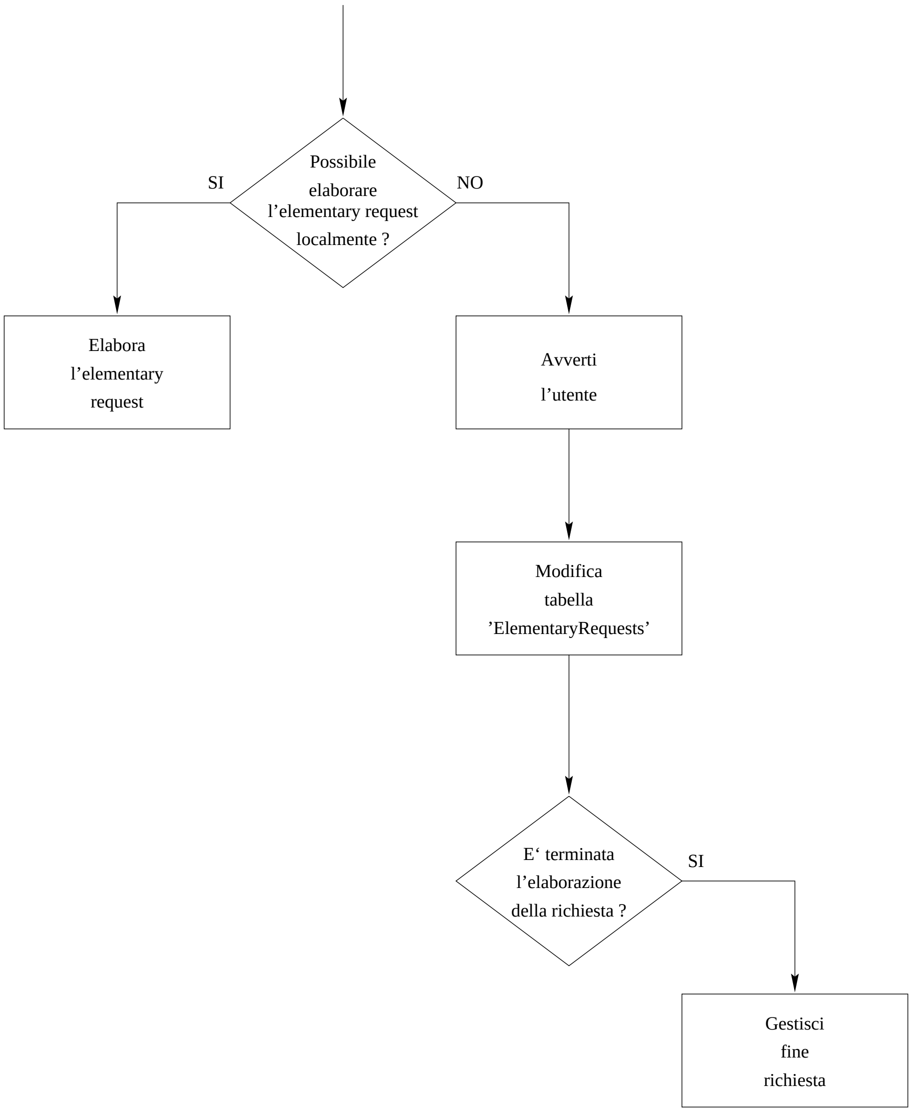

Figura 7.8: Analisi dell'elementary request

ElemReqId	TaskOfProdId	Status	JID
er-i	prod1.t1	succesfully processed	12413
er-i	prod1.t2	waiting inputs	
er-i	prod1.t3	waiting inputs	
er-i	prod1.t4	waiting inputs	
er-x	prod3.tx	succesfully processed	12432
er-x	prod3.ty	processing	12435
...	...	...	...

Tabella 7.1: Tabella 'TasksOfElementaryRequests'

Se invece i controlli preliminari hanno avuto esito positivo, l'elementary request è pronta per essere elaborata dall'Elementary Request Decomposer (vedi par. 7.6.4).

7.6.4 Elementary Request Decomposer

Il modulo Elementary Request Decomposer scompone ogni elementary request nei task che costituiscono la corrispondente produzione.

Prima di questa operazione, viene modificato il campo 'Status' della tabella *ElementaryRequests* per l'elemento considerato: viene definito il nuovo valore 'Processing', per indicare appunto che inizia l'elaborazione.

Per tener traccia dello stato dell'elaborazione dei vari task delle varie elementary request, si utilizza una particolare struttura dati, gestita in maniera persistente, chiamata *TasksOfElementaryRequests*, rappresentata in tabella 7.1.

Ogni elemento è rappresentato da una quaterna di valori: l'identificatore della elementary request, l'identificatore del task componente la corrispondente produzione, lo stato dell'elaborazione di questo task, e un Job Identifier (JID) in seguito descritto. Ogni task "generato" dall'Elementary Request Decomposer viene definito in questa tabella: vengono inizializzati i campi 'ElemReqId', 'TaskOfProdId' e 'Status' (con il valore 'Waiting inputs', per indicare che il task, per poter iniziare l'esecuzione, è in attesa dei relativi input), mentre per il momento all'attributo 'JID' non viene assegnato nessun valore. Considerando l'elementary request *er-i*, dell'esempio descritto nel paragrafo 7.5, nella tabella 'TasksOfElementaryRequests' vengono definiti quattro elementi, relativi ai quattro task della production *prod1*.

Quando si elabora una elementary request, è necessario conoscere anche quali sono i file associati ai vari data box della corrispondente produzione: questi file vengono infatti usati come input-output dai vari task. A questo scopo si utilizza

ElemReqId	DataBoxId	FileId
er-i	prod1.inp1	sinfile1
er-i	prod1.inp3	sinfile3
er-i	prod1.inp6	sinfile6
er-i	prod1.inp2	filei
er-i	prod1.dat4	filei-4
er-i	prod1.dat5	filei-5
er-x	prod3.ds1	filey
...	...	...

Tabella 7.2: Tabella 'FilesOfElementaryRequests'

un'altra struttura, chiamata *FilesOfElementaryRequests*, rappresentata in tabella 7.2.

Ogni elemento di questa tabella viene rappresentato da una terna: l'identificatore dell'elementary request, l'identificatore del data box della produzione, e l'identificatore del file. L'Elementary Request Decomposer, esaminando le tabelle *StreamIOOfElementaryRequests* e *SingletonIOOfRequests*, registra nella tabella *FilesOfElementaryRequests* gli elementi relativi agli input file (di tipo stream e di tipo singleton) dell'elementary request considerata. Sempre con riferimento all'elementary request *er-i* dell'esempio considerato, vengono definiti nella tabella 'FilesOfElementaryRequests' i file *sinfile1*, *sinfile2*, *sinfile6* e *filei*, associati rispettivamente ai data box *inp1*, *inp3*, *inp6* e *inp2*.

Dopo queste operazioni, viene "attivato" il Task Starter, per verificare quali task di questa elementary request sono pronti per essere eseguiti: a tal proposito si veda il paragrafo 7.6.5.

Queste procedure sono descritte nel seguente pseudo-codice:

```
ElementaryRequestDecomposer(er : ElementaryRequest);
begin
  ModifyTable_ElementaryRequests();
  req = RequestOfElementaryRequest(er);
  prod = ProductionOfRequest(req);
  for each t task di prod
      do InsertIntoTable_TasksOfElementaryRequests(t); od
  for each f1 singleton input file di req
      do InsertIntoTable_FilesOfElementaryRequests(f1); od
  for each f2 stream input file di er
      do InsertIntoTable_FilesOfElementaryRequests(f2); od
  TaskStarter(er);
end
```

L'Elementary Request Decomposer viene implementato come un processo daemon, che viene "attivato" quando sono presenti elementi nella 'local queue of elementary requests'.

7.6.5 Task Starter

Il Task Starter è il modulo che deve verificare quali sono i task pronti per essere eseguiti (i task per i quali i relativi input sono disponibili), e ne deve gestire l'esecuzione.

Per controllare quali sono i task di una elementary request che possono essere eseguiti, il Task Starter analizza la tabella *TasksOfElementaryRequests*, considerando tutti i task relativi all'elementary request considerata per cui l'attributo 'Status' sia pari a 'Waiting inputs' (i task, quindi, che devono ancora essere eseguiti, perché in attesa dei corrispondenti file di input). Per ognuno di questi task si controlla, esaminando la tabella *FilesOfElementaryRequests*, se tutti i relativi file di input sono disponibili. In caso positivo il task può essere eseguito.

Si consideri, ad esempio, l'elementary request *er-i*. Il contenuto delle tabelle *TasksOfElementaryRequests* e *FilesOfElementaryRequests* sia rispettivamente quello delle tabelle 7.1 e 7.2. In questo caso il Task Starter rileva che i task *t2* e *t3* sono pronti per essere eseguiti, in quanto per tutti i relativi data box di input della produzione (*inp3* e *dat4*, *dat5* e *inp6*) sono definiti i corrispondenti file.

Il seguente codice descrive sommariamente le operazioni che devono essere effettuate quando il Task Starter rileva che un task può essere eseguito:

```
TaskStarter(t : Task; er : ElementaryRequest);
begin
  ModifyTable_TasksOfElementaryRequests();
  for each f input file di t
      do
      if (necessary)
        then CopyFromTapeToBuffer(f); fi
      Lock(f); od
  for each outst stream output di t
      do
      siz = GetExpectedSize(outst);
      f1 = NewFile(siz); od
  for each outsin singleton output di t
      do
      if (necessary)
        then
              siz = GetExpectedSize(outsin);
              f2 = NewFile(siz); fi od
  tt = TaskType(t);
  arch = ChooseArchitecture(tt);
  Submit(t, arch);
end
```

Per prima cosa si modifica l'attributo 'Status' del corrispondente elemento della tabella *TasksOfElementaryRequests*, con il nuovo valore 'Processing'.

È poi necessario accertarsi che tutti i file di input del task siano presenti su disco (si controlla la tabella *FilesOnBuffers*). Se un file non è presente su un disk buffer, lo si copia da nastro a disco: questa operazione viene gestita dal Data Manager, come descritto nel paragrafo 7.6.11.

Per tutti i file di input del task si definisce uno stato di "lock", per evitare che questi vengano spostati altrove durante l'elaborazione.

Per ogni output di tipo stream del task, si analizza il corrispondente elemento della tabella *DataBoxesOfRequests*, per verificare qual è la massima dimensione che deve essere allocata per il file che verrà generato. Attraverso il Data Manager si definisce un nuovo elemento nel Data Database per questo file, allocando lo spazio necessario.

Si esaminano poi gli output di tipo singleton del task. Per ognuno di essi si controlla se il corrispondente file esiste già (potrebbe essere già stato creato da un'altra elementary request della stessa richiesta computazionale). In caso negativo è necessario definire il relativo file di output nel Data Database: la procedura è del tutto analoga a quella vista per gli output di tipo stream.

Si individua qual è il tipo del task considerato, che può essere stato implementato per diverse architetture: se ne deve scegliere una per l'esecuzione (questa procedura è descritta nel paragrafo 7.6.6).

Dopo queste operazioni il task è pronto per essere "sottomesso" al batch system per la relativa esecuzione, che dovrà essere eseguita su un computer dell'architettura selezionata.

Si "costruisce" la riga di comando che deve essere specificata per l'esecuzione del task, utilizzando le informazioni memorizzate nelle tabelle *ImplementationsOfTasks* e *ParametersForTasks*. Per quel che riguarda i parametri che rappresentano gli input-output, il pathname di questi file viene trovato esaminando la tabella *FilesOnBuffers*.

Analizzando la tabella *LocalComputers*, si verifica quali sono i computer dell'architettura prima selezionata presenti nel sito (si considerano solo i calcolatori il cui stato è pari a 'Enabled', quindi i computer che possono "accettare" task da eseguire). Utilizzando le apposite API fornite dal batch system, si sottomette al sistema la riga di comando che invoca l'esecuzione del task, indicando che l'elaborazione deve avvenire su uno di questi computer.

È compito del batch system decidere su quale computer eseguire l'elaborazione. Il batch system gestisce anche il suspend/resume dei job, secondo le regole specificate dall'amministratore del pool di workstation (vedi par. 5.1).

La routine usata per il submit del task "torna" un identificatore, chiamato JID (Job IDentifier). Utilizzando questo valore, si definisce l'attributo 'JID' della tabella *TasksOfElementaryRequests* per l'elemento che rappresenta questo task.

Infine si crea un elemento nella tabella *ProcessingsOfSingleTasks*, per definire l'elaborazione del task considerato: all'attributo 'Status' viene associato il valore 'Submitted', per indicare che il task è stato sottomesso al batch system per la relativa esecuzione.

7.6.6 Scelta di un'architettura per l'esecuzione di un task

L'algoritmo per scegliere su quale architettura eseguire un task può essere facilmente modificato (senza dover cambiare anche il codice di tutti gli altri moduli), in modo da definire nuovi parametri, nuovi criteri, ecc...

Al momento si utilizza un algoritmo molto semplice, in cui si considera il numero dei computer presenti per le varie architetture, le loro performance, il costo per l'esecuzione del task sulle varie architetture, i job già assegnati ai vari computer.

Siano A_1, A_2, ..., A_n le architetture per cui esiste un'implementazione del task considerato. Siano $C_{i,1}$, $C_{i,2}$, ..., $C_{i,Ni}$ i computer dell'architettura A_i abilitati ad elaborare task, e siano $P_{i,1}$, $P_{i,2}$, ..., $P_{i,Ni}$ le performance di questi computer (si considera l'attributo 'Performances' del Local Computer Database: vedi par. 4.3.1). Sia $COST_i$ il costo per l'esecuzione del task considerato sull'architettura A_i (il valore del parametro 'cost of the task' del Task Database: vedi par. 4.5.1). Sia n_i il numero di task assegnati ai computer dell'architettura A_i per l'esecuzione, la cui elaborazione deve ancora essere completata (task in quel momento già in esecuzione, o in attesa di essere eseguiti). Per l'esecuzione del task si sceglie l'architettura A_i per cui è massimo il rapporto:

$$\frac{\sum_{j=1}^{Ni} P_{i,j}}{COST_i + n_i}$$

7.6.7 Job Checker

Il **Job Checker** è il modulo del sistema DEMI che si occupa di rilevare lo stato dei job sottomessi al batch system.

In particolare viene rilevato l'inizio dell'esecuzione di un job (che passa dallo stato di suspend allo stato di run), l'elaborazione di un job completata con successo, e l'elaborazione fallita di un task. Questi eventi vengono gestiti dal Job Manager.

7.6.8 Job Manager

Inizio dell'esecuzione di un task

Quando si rileva che è iniziata l'esecuzione di un task su un certo calcolatore, si definiscono gli attributi 'CompName' e 'Status' (ponendo questo campo pari a 'Started') del corrispondente elemento della tabella *ProcessingsOfSingleTasks*.

Elaborazione di un task completata con successo

Il seguente pseudo-codice definisce le operazioni che devono essere eseguite dal Job Manager, quando è terminata l'elaborazione di un job di cui viene rilevato il corrispondente job identifier (JID):

EndTask(*JID* : *Integer*);
begin
  t = *TaskOfJID*(*JID*);
  er = *ElementaryRequestOfTask*(*t*);
  ModifyTable_TasksOfElementaryRequests();
  ModifyTable_ProcessingsOfSingleTasks();
  for *each f1 file generato da t*
    do *InsertIntoTable_FilesOfElementaryRequests*(*f1*); **od**
  for *each f2 stream file temporaneo usato come input da t e non piu' necessario*
    do *Delete*(*f2*); **od**
  if (*EndElReq*(*er*))
    then *ManageEndElReq*(*er*);
    else *TaskStarter*(*er*); **fi**
end

Analizzando la tabella *TasksOfElementaryRequests*, si identifica qual è il task che è terminato: per questo elemento della tabella *TasksOfElementaryRequests* si modifica il valore del campo 'Status', ponendolo pari a 'Succesfully processed'.

Si modifica anche il corrispondente elemento della tabella *ProcessingsOfSingleTasks*, definendo un nuovo valore per il campo 'Status' ('Succesfully processed'), e specificando la data e l'ora in cui l'elaborazione è terminata.

Si definiscono poi nella tabella *FilesOfElementaryRequests* gli output file (di tipo stream e di tipo singleton) prodotti nell'elaborazione di questo task. Gli stream file temporanei usati come input dal task, e non più necessari, vengono cancellati.

Si verifica poi se si è terminato di elaborare l'elementary request associata a questo task (si controlla il campo 'Status' della tabella *TasksOfElementaryRequests*, per vedere se esistono task di questa elementary request che devono ancora essere elaborati) e, in caso positivo, si gestisce in modo opportuno questa situazione (vedi par. 7.6.9). Se invece l'elaborazione dell'elementary request non è stata completata, si attiva il Task Starter per verificare se per essa ci sono nuovi task pronti per essere eseguiti (vedi par. 7.6.5): dovranno essere eseguiti quei task che usano come input i file di output del task appena terminato.

Elaborazione "fallita" di un task

Il codice sotto riportato descrive le operazioni che il Job Manager esegue quando il Job Checker rileva che l'esecuzione di un task, di cui viene rilevato il corrispondente Job Identifier, è fallita (non è stata completata con successo):

CrashedTask(*JID* : *Integer*);
begin
  t = *TaskOfJID*(*JID*);
  ModifyTable_ProcessingsOfSingleTasks();
  NotifyUser();
  for *each f stream file prodotto da t*
    do
    Delete(*f*); **od**
  ModifyTable_TasksOfElementaryRequests();
  er = *ElementaryRequestOfTask*(*t*);
  TaskStarter(*er*);
end

Analizzando la tabella *TasksOfElementaryRequests* si identifica qual è il task associato al job identifier considerato. L'utente che ha sottomesso la corrispondente richiesta viene avvertito riguardo il problema. Si modifica poi il campo 'Status' dell'elemento della tabella *ProcessingsOfSingleTasks* relativo a questo task: viene definito il nuovo valore 'Crashed'. Si invoca poi il Data Manager, per eliminare gli stream file prodotti dal task. Si aggiorna il campo 'Status' dell'elemento della tabella *TasksOfElementaryRequests* corrispondente a questo task: il nuovo valore che viene inserito è 'Waiting inputs' (il task dovrà infatti essere rieseguito). Infine si invoca il Task Starter, per controllare se per l'elementary request corrispondente a questo task ci sono nuovi task pronti per essere eseguiti (vedi par. 7.6.5): con questa operazione si "forza" la riesecuzione del task considerato.

7.6.9 Elaborazione completata di una elementary request

Quando l'elaborazione di una elementary request è stata completata, per prima cosa si modifica il campo 'Status' del corrispondente elemento della tabella *ElementaryRequests*, definendo il nuovo valore 'Succesfully processed'.

Tutti gli output stream file che sono stati prodotti in questa elementary request devono essere "registrati". Questi file di output devono poi essere "spostati" dove specificato dall'utente che ha sottomesso la richiesta. Queste operazioni vengono gestite dal Data Manager (vedi par. 7.6.11).

I file di output devono anche essere definiti nella tabella *StreamIOOfElementaryRequests*, la struttura dati usata per memorizzare gli input-output di tipo stream delle elementary request.

Si cancellano poi tutti gli elementi delle tabelle *TaskOfElementaryRequests* e *FilesOfElementaryRequests* relativi all'elementary request considerata.

Si considerano gli stream input file utilizzati in questa elementary request: per ognuno di essi viene rimosso lo stato di “lock”, a meno che il file non sia stato “allocato” in un altro processing.

Infine si verifica se si è terminato di elaborare anche la corrispondente richiesta computazionale (esaminando la tabella *ElementaryRequests*, si controlla se esistono elementary request associate alla richiesta considerata per cui l'elaborazione è ancora in corso, o deve ancora avvenire), e in caso si gestisce opportunamente la situazione (vedi par. 7.6.10).

La seguente procedura descrive tutte queste operazioni:

$ManageEndElReq(er : ElementaryRequest)$;
begin
  ModifyTable_ElementaryRequests();
  for *each f stream output file prodotto in er*
    do
    Register(f);
    MoveOutputFile(f);
    InsertIntoTable_StreamIOOfElementaryRequests(f); **od**
  ModifyTable_TasksOfElementaryRequests();
  ModifyTable_FilesOfElementaryRequests();
  for *each f stream input file usato come input in er*
    do *Unlock*(f); **od**
  req = *RequestOfElementaryRequest*(*er*);
  if (*EndReq*(*req*))
    then *ManageEndReq*(*req*); **fi**
end

7.6.10 Elaborazione completata di una richiesta

Quando è terminata l'elaborazione di una richiesta computazionale, per prima cosa si modifica il campo 'Status' del corrispondente elemento della tabella *Requests*, ponendo questo attributo pari a 'Succesfully processed'.

Si analizzano poi i singleton output file che sono stati prodotti nell'elaborazione di questa richiesta: i file temporanei devono essere cancellati, mentre i file che corrispondono ad output della produzione devono essere “registrati”. È anche necessario spostare questi singleton output file dove indicato dall'utente che ha definito la richiesta: tutte queste procedure vengono gestite dal Data Manager, come descritto nel paragrafo 7.6.11.

I singleton output file devono anche essere definiti nella tabella *SingletonIOOfRequests*.

Si rimuove, poi, lo stato di “lock” associato a tutti i singleton input file utilizzati in questa richiesta, a meno che questi file non siano stati “allocati” anche in altri processing ancora in corso.

Infine si deve riportare il risultato dell'elaborazione al corrispondente utente: esaminando le tabelle *SingletonIOOfRequests* e *StreamIOOfElementaryRequests*, si trovano rispettivamente tutti i singleton output file associati alla richiesta considerata, e gli stream output file associati alle sue elementary request. La lista degli identificatori di tutti questi file viene inviata, via mail, all'utente che ha sottomesso la richiesta.

Il seguente algoritmo rappresenta in maniera schematica la procedura appena descritta:

ManageEndReq(*req* : *Request*);
begin
  ModifyTable_Requests();
  for *each f singleton output file prodotto in req*
    do
    if (*f e' un file temporaneo*)
      then *Delete*(*f*);
      else *Register*(*f*);
        MoveOutputFile(*f*);
        InsertIntoTable_SingletonIOOfRequests(*f*); **fi** **od**
  for *each f singleton input file usato come input in req*
    do *Unlock*(*f*); **od**
  NotifyResultsToUser();
end

7.6.11 Data Manager

Il Data Manager è il modulo che si occupa di gestire tutte le operazioni che riguardano i file di dati.
Come detto nei paragrafi precedenti, in particolare il Data Manager:

- copia i file di input dei task da nastro a disco;
- definisce nel Data Database i nuovi file di dati;
- registra i file di output generati nelle elaborazioni dei vari task;
- cancella i file temporanei generati nelle elaborazioni dei vari task, e i file prodotti dai task la cui elaborazione è fallita;
- sposta i file di output dove specificato dall'utente che ha sottomesso la richiesta computazionale.

Queste operazioni sono descritte di seguito.

Copia di un file da nastro a disco

Questa procedura viene attivata quando un file, non presente su disco, deve essere usato come input di un task: è quindi necessario copiarlo su un disk buffer.

Si verifica qual è la dimensione del file (controllando l'attributo 'Size' della tabella *LocalDataFiles*), per poter scegliere un opportuno disk buffer per la memorizzazione: questa operazione viene gestita dal Disk Manager (vedi par. 7.6.12).

Analizzando la tabella *FilesOnTapes*, si individua su che nastro il file considerato è presente. Si "attiva" poi il Local Data Mover, per effettuare la copia del file dal nastro al disk buffer prima individuato (vedi par. 7.6.14).

Quando la copia è stata completata, si inserisce un nuovo elemento nella tabella *FilesOnBuffers*, per definire il file copiato su disco. Infine il Disk Manager aggiorna opportunamente il Local Buffer Database, per registrare il nuovo file copiato su disco (vedi par. 7.6.12).

Definizione di un nuovo file

Questa procedura del Data Manager viene invocata quando si deve definire nel Data Database un nuovo file di dati, che sarà generato da un task pronto per essere eseguito.

Viene creato un nuovo elemento nella tabella *DataFiles*, per definire il file considerato: all'attributo 'FileId' viene associato un nuovo valore. Viene anche inserito un elemento (con stesso FileId) nella tabella *LocalDataFiles*. Il Disk Manager sceglie poi un disk buffer per la memorizzazione del file (vedi par. 7.6.12), e di conseguenza si definisce un corrispondente elemento nella tabella *FilesOnBuffers*.

Registrazione di un file

Quando termina l'elaborazione di una elementary request, è necessario registrare i corrispondenti stream output file (questi file erano già stati definiti nel Data Database prima della loro effettiva creazione). Analogo discorso per i singleton output file, che devono essere registrati quando termina l'esecuzione della relativa richiesta computazionale.

Quando si deve registrare un file, per prima cosa se ne rileva la dimensione reale, in modo da definire l'attributo 'Size' del corrispondente elemento della tabella *LocalDataFiles*. Si invoca poi il Disk Manager, che aggiorna in maniera opportuna il Local Buffer Database (vedi par. 7.6.12).

Cancellazione di un file

I file temporanei di tipo stream e di tipo singleton prodotti nelle varie elaborazioni devono essere cancellati. L'operazione di cancellazione è necessaria anche quando un task non termina con successo la sua esecuzione: prima di rieseguire il task, gli stream file prodotti fino a quel momento devono essere cancellati.

Dopo aver fisicamente cancellato il file da disco, si aggiornano le tabelle *DataFiles*, *LocalDataFiles* e *FilesOnBuffers*, eliminando gli elementi relativi al file considerato. Il Disk Manager aggiorna poi il Local Buffer Database, per "rilasciare" lo spazio prima occupato dal file.

Copia di un file di output

Gli stream output file prodotti in una elementary request, e i singleton output file prodotti in una computational request, devono essere spostati dove indicato dall'utente al momento del submit della richiesta.

Analizzando la tabella *DescriptionsOfOutputsOfRequests*, si verifica dove il file di output considerato deve essere spostato.

Se il file deve essere memorizzato in un disk buffer non si deve fare nulla: si rimuove semplicemente il "lock" associato al file.

Se il file deve essere memorizzato in una directory privata, si esegue la copia su questa directory, mantenendo comunque l'istanza del file presente sul disk buffer (su questo file viene rimosso lo stato di "lock").

Il caso più complesso da considerare è quello per cui il file deve essere memorizzato su un nastro: questa operazione viene gestita dal Tape Manager (vedi par. 7.6.13).

7.6.12 Disk Manager

Il Disk Manager è il modulo che si occupa delle procedure che riguardano i file di dati memorizzati sui disk buffer.
In particolare gestisce le seguenti operazioni:

- scelta di un disk buffer per la memorizzazione di un file;
- modifica del Local Buffer Database per la registrazione di un file;
- modifica del Local Buffer Database per la cancellazione di un file.

Scelta di un disk buffer per la memorizzazione di un file

Il diagramma di figura 7.9 descrive le operazioni che devono essere eseguite dal Disk Manager quando è necessario scegliere un buffer dove memorizzare un file, di cui si conosce la dimensione (o una stima della dimensione).

Si considera l'elemento della tabella *LocalBuffers* che sia "abilitato", e per cui sia più alto il valore del parametro 'FreeSpac' (cioè il disk buffer che ha più spazio libero). Se il valore 'FreeSpac' è comunque inferiore alla dimensione del file che deve essere memorizzato, si considera il file più grande memorizzato su questo buffer, che non sia "marcato" come 'locked': il Tape Manager individua un nastro dove memorizzare questo file (vedi par. 7.6.13), e il Local Data Mover lo sposta da disco a questo nastro (vedi par. 7.6.14). Dopo questa operazione si aggiornano opportunamente le tabelle *LocalBuffers*, *FilesOnBuffers* e *FilesOnTapes*.

Se necessario si ripete la procedura più volte.

Se al termine di questo ciclo il valore del parametro 'FreeSpac' è comunque inferiore alla dimensione richiesta per il file, si ripete l'operazione considerando un altro disk buffer.

Una volta trovato il buffer dove memorizzare il file, si modifica il valore del parametro 'FreeSpac' dell'elemento della tabella *LocalBuffers* relativo a questo disco, sottraendo la dimensione del file (per indicare che su questo buffer si "alloca" uno spazio pari a questo valore).

Se, in ogni caso, l'operazione non ha successo (ovvero non viene trovato un disk buffer che possa essere usato per la memorizzazione del file), il local administrator viene avvertito riguardo il problema. Il Disk Manager continua comunque a controllare se su un disco si rende disponibile lo spazio necessario.

Registrazione di un file di dati

Quando un file di dati deve essere "registrato", sull'elemento della tabella *LocalBuffers* corrispondente al disco dove il file è memorizzato, si modificano i parametri 'FreeSpac' e 'UsSpac', per definire la dimensione reale del file (prima lo spazio era stato solamente allocato).

Cancellazione di un file di dati

Quando un file di dati viene cancellato, sull'elemento della tabella *LocalBuffers* corrispondente al disco dove il file era memorizzato, si modifica il parametro 'FreeSpac', per indicare che lo spazio precedentemente allocato viene rilasciato.

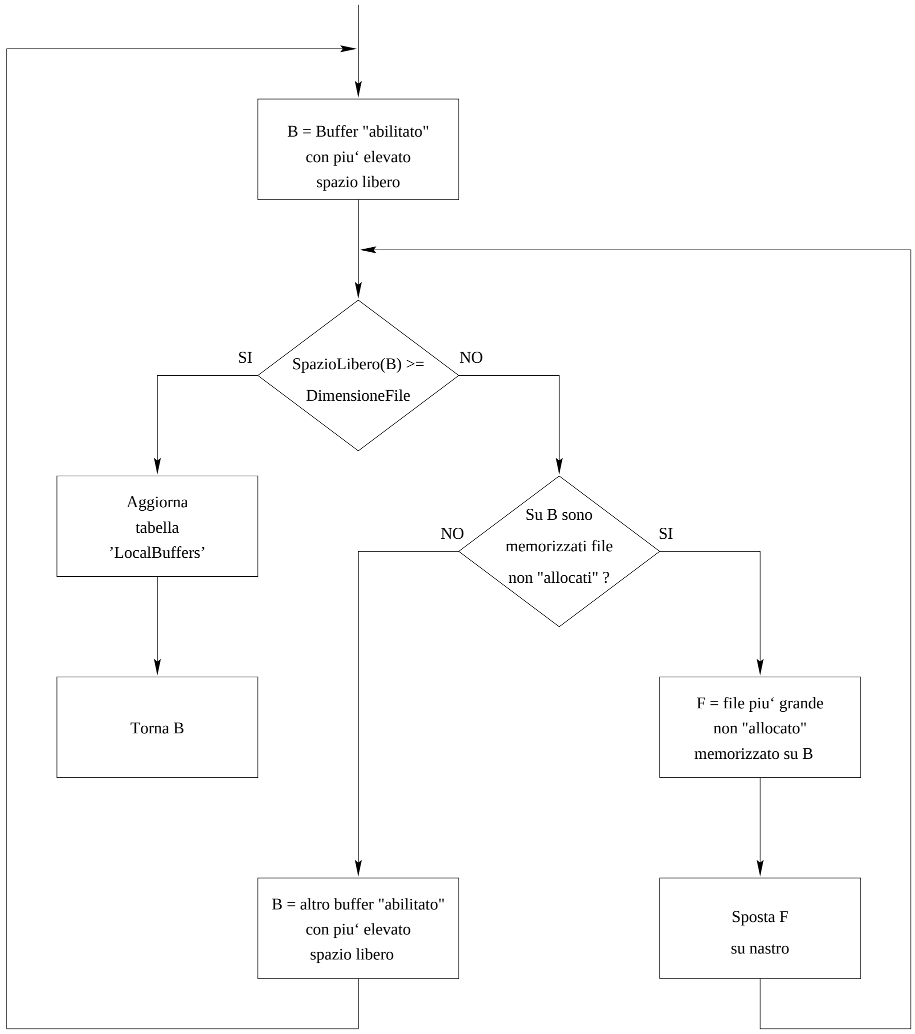

Figura 7.9: Scelta di un disk buffer per la memorizzazione di un file

ReqId	DataBoxId	TapeId
req00001	prod1.out7	tapeid221
req00001	prod1.out8	tapeid342
...	...	...

Tabella 7.3: Tabella 'OutputsOnTapes'

7.6.13 Tape Manager

Il Tape Manager è il modulo che gestisce le operazioni che riguardano i file di dati memorizzati sui nastri.
In particolare questo modulo:

- si occupa di spostare i file di output da disco a nastro;
- individua un nastro dove copiare un file di dati, che deve essere spostato da un disk buffer su cui c'è bisogno di liberare spazio.

Spostamento di un file di output da disco a nastro

È opportuno che, per ogni richiesta computazionale, i corrispondenti file di output vengano memorizzati su "nuovi" nastri (che non contengono cioè altri file). È opportuno inoltre che i file di output associati agli stessi data box siano memorizzati sugli stessi nastri.

Il Tape Manager gestisce allora una struttura dati, chiamata *OutputsOnTapes*, rappresentata in tabella 7.3, in cui ogni elemento è definito da una terna: l'identificatore della richiesta, l'identificatore del data box della corrispondente produzione, e l'identificatore del nastro.

La procedura che il Tape Manager deve considerare quando si deve copiare un file di output da disco a nastro è descritta dal flow chart di figura 7.10.

Si verifica, quindi, se esiste già nella tabella un elemento i cui campi 'ReqId' e 'DataBoxId' corrispondono alla richiesta e al data box associati al file considerato. In caso positivo il file deve essere copiato nel nastro il cui identificatore è rappresentato dal campo 'TapeId'. Si controlla, esaminando l'attributo 'FreeSpac' del corrispondente elemento della tabella *LocalTapes*, se su questo nastro c'è spazio a sufficienza per la memorizzazione del file. In caso positivo si invoca il Local Data Mover, che procede con la copia (vedi par. 7.6.14). Dopo questa operazione viene creato un nuovo elemento nella tabella *FilesOnTapes*, per definire il file copiato su nastro, e si aggiorna il campo 'FreeSpac' del corrispondente elemento della tabella *LocalTapes*.

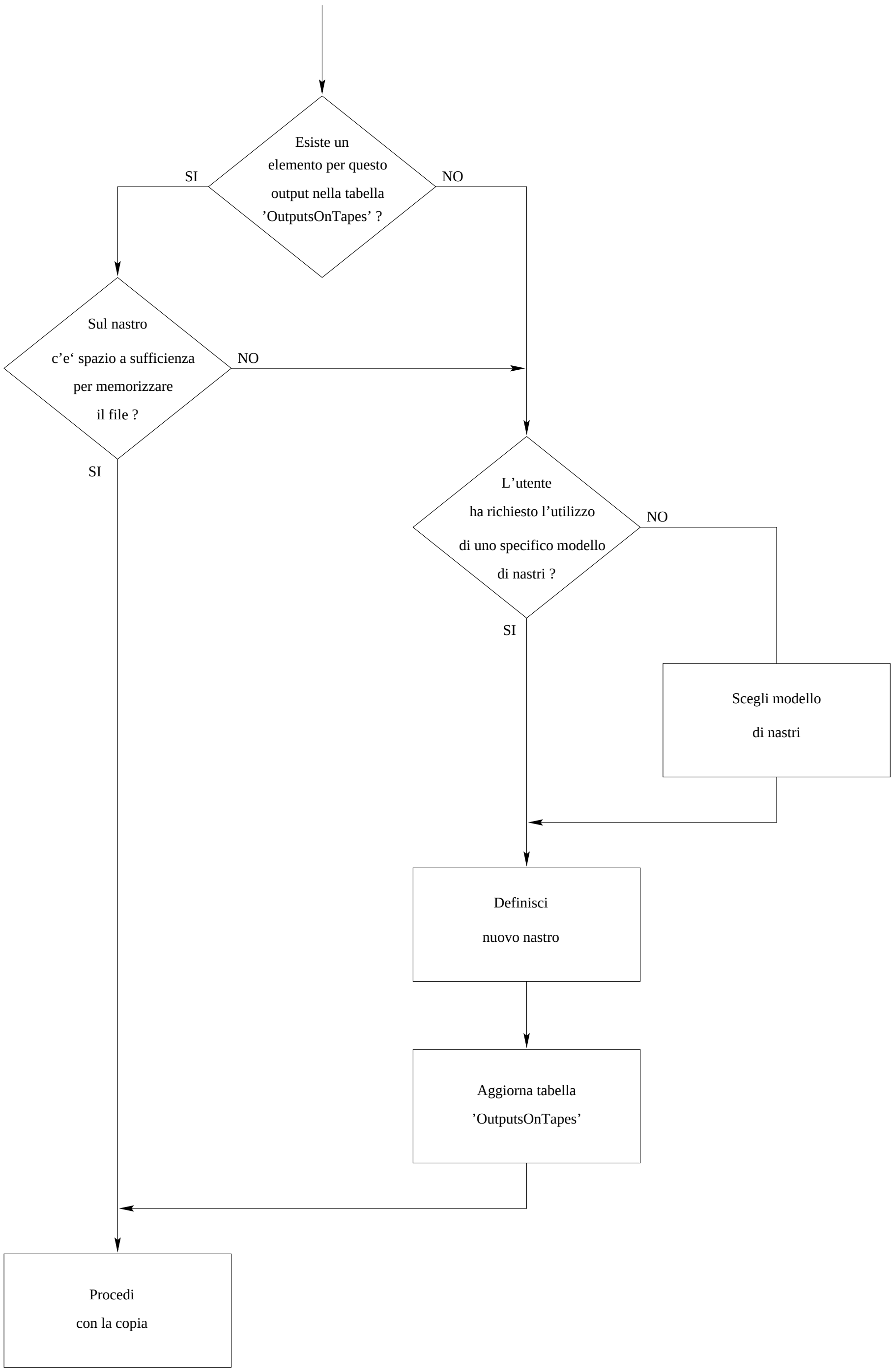

Figura 7.10: Copia di un file di output da disco a nastro

Se invece sul nastro non c'è spazio a sufficienza, se ne deve utilizzare uno nuovo. Prima, però, è necessario scegliere il tipo di nastro da considerare, a meno che l'utente non abbia richiesto l'utilizzo di uno specifico modello (le procedure di scelta del modello di nastro, e di "definizione" di un nuovo nastro sono di seguito descritte). Si può poi aggiornare il campo 'TapeId' della tabella *OutputsOnTapes*, relativamente all'elemento considerato, con l'identificatore del nuovo nastro. Il Local Data Mover procede poi con la copia.

Se invece non esiste un elemento nella tabella *OutputsOnTapes* per questo output (quindi questo è il primo file prodotto associato a questo data box di output, relativamente alla richiesta considerata), lo si deve definire. Anche in questo caso si devono considerare due casi: l'utente può aver definito o meno il tipo di nastro da utilizzare. Nel secondo caso come prima operazione è quindi necessario selezionare un modello. Si definisce poi un nastro del modello considerato.

A questo punto può quindi essere definita la terna (ReqId, DataBoxId, TapeId), che viene inserita nella tabella *OutputsOnTapes*. Il Local Data Mover procede poi con la copia.

Scelta di un modello di nastro

Quando si deve scegliere un modello di nastri (per poi utilizzare un nastro di questo tipo per la memorizzazione di file), si controlla nella tabella *LocalTapeDevices* quali sono i tape device che non sono "disabilitati" (quindi "liberi", o in quel momento usati per una lettura/scrittura da/su nastro): si sceglie il modello di tape device per cui è maggiore il numero di istanze.

Utilizzo di un nuovo nastro

Questa procedura viene invocata quando c'è la necessità di utilizzare un nuovo nastro, di un certo modello specificato, e di cui si indica anche il tipo dei file che dovranno essere memorizzati su di esso (come detto, infatti, i file memorizzati su un nastro devono essere tutti dello stesso tipo).

Viene creato un nuovo elemento della tabella *LocalTapes*, a cui si inizializzano gli attributi 'TapeDevModId' e 'Description'. Al campo 'TapeId' viene semplicemente associato un nuovo valore. Infine l'attributo 'FreeSpac' viene posto pari al valore del campo 'Capacity' del corrispondente elemento della tabella *TapeDeviceModels*.

Il valore dell'identificatore 'TapeId' viene fisicamente scritto su nastro, in modo che lo Staging Manager possa controllare che effettivamente vengano "montati" i nastri richiesti (vedi par. 7.6.14).

Individuazione di un nastro dove copiare un file

Quando si deve individuare un nastro per copiare un file di dati (che deve essere spostato da un disk buffer sul quale è necessario liberare spazio), tra gli elementi della tabella *LocalTapes* se ne seleziona uno, per cui il campo 'Description' coincida con il tipo del file considerato, e per cui ci sia sufficiente spazio per la memorizzazione. Se questo elemento non viene trovato, è necessario "definire" un nuovo nastro. Prima però si deve decidere che modello di nastro considerare (entrambe queste procedure sono state precedentemente descritte).

Il Local Data Mover può poi procedere con la copia.

Il seguente pseudo-codice riassume quanto detto:

$MoveFromBufferToTape(b : Buffer;\ f : File);$
begin
  typ = *DescriptionOfFile*(f);
  siz = *SizeOfFile*(f);
  if (*non esiste t nastro con siz spazio libero che contiene file di tipo typ*)
    then
      tm = *ChooseTapeModel*();
      t = *NewTape*(*tm*); **fi**
  CopyToTape(b, f, t);
end

7.6.14 Local Data Mover

Il Local Data Mover gestisce la copia di file di dati da dischi a nastri, e viceversa.

Quando si deve copiare un file da uno specifico nastro ad uno specifico disk buffer (o viceversa), per prima cosa si fa richiesta di staging del nastro considerato, ovvero si richiede che tale nastro venga "montato" su un tape device.

Un particolare modulo del sistema DEMI, chiamato **Staging Manager** (rappresentato in figura 7.11), viene utilizzato per la gestione delle richieste di staging.

Lo Staging Manager riceve la richiesta di "mount" di un nastro, e "risponde" a questa richiesta con l'identificatore del tape device sul quale il nastro è stato "montato".

Quando è richiesto di "montare" un certo nastro su un tape device, lo Staging Manager per prima cosa ne identifica il modello, esaminando l'attributo 'TapeDevModId' del corrispondente elemento della tabella *LocalTapes*. Cerca poi nella tabella *LocalTapeDevices* un elemento che rappresenti un tape device di questo modello che sia abilitato, e che in quel momento non sia utilizzato. Se questo

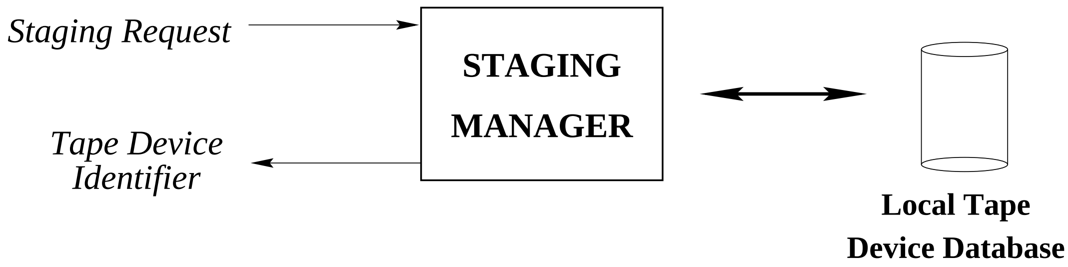

Figura 7.11: Staging Manager

tape device non viene trovato, lo Staging Manager attende finché non se ne rende disponibile uno.

Quando il tape device viene individuato, questo viene "marcato" come allocato, in modo che non venga usato per qualche altra copia. Si cerca poi nella tabella *Users* tutti gli utenti che siano operatori: viene inviato un mail a queste persone, con la richiesta di eseguire il mount del nastro nello specifico tape device considerato.

Non appena il nastro è stato "montato" sul tape device considerato, si effettua la copia del file (da disco a nastro, o viceversa).

Dopo questa operazione, il tape device utilizzato viene definito come "libero" (non più allocato), e quindi disponibile per altre copie.

Lo Staging Manager viene implementato come un processo daemon, che viene "attivato" quando vi sono richieste di "mount" di nastri.