F.R.Cavallo, Bologna luglio 2001.

ESERCITAZIONE SU ORCA

1. Come costruirsi una federazione "privata"

2. Come creare OO–hits partendo da un fz file di CMSIM e scriverli sul database

3. Come creare DIGIS partendo dagli hits e aggiungerli al database

4. Come leggere i dati dal database con un programma di analisi che produce n–tuple

5. Un esempio di "azione su domanda"

1. Come creare una federazione privata

a) Nella dir personale /gwpool/home/cmsprod/user/ORCA_4_4_0 copiare il file:

/gwpool/cms/Releases/ORCA/ORCA_4_4_0/src/Utilities/Configuration/src/makefd.local

b) Sostituire nella definizione di lockhost a ${localhost},

infn–corsi98.corsi.infn.it

(i pc satelliti non hanno il lock server)

c) Creare la federazione:

```
> eval ‘scram runtime –csh‘
> source makefd.local  myfederation_dir
> cd  myfederation_dir
> ls –l
```

Lo script ha creato la directory che contiene la federazione.
Nella directory ci sono il bootfile e un fdinit.csh per l’inizializzazione e il riaccesso futuro.

2. Come creare SimHits da un fz–file e scriverli nel DB

Occorrente:

a) fz–file con gli eventi simulati da CMSIM:
/gwpool/scratch/single_mu_pt50.fz

b) rz–file con la geometria del detector utilizzata da CMSIM:
/gwpool/scratch/cms_geom_out.rz

(entrambi copiati da /afs/cern.ch/cms/00/Reconstruction/datafiles/cms120/
N.B. Prendere sempre fz–file e geometria dalla stessa dir!)

c) codice opportuno:

in ~/user/ORCA_4_4_0/src:
> cvs co –r ORCA_4_4_0 Examples/ExProduction

(contiene tutto quel che serve.)

Esecuzione:

1) building del programma:

> scram build shared
> scram build bin

Il BuildFile di /ExProduction prevede la creazione contemporanea di diversi binari. Tra essi writeHits serve a creare gli oo–hits dall'fz file.

2) definizione della geometria del detector e degli identificatori per il DB da scrivere:

in writeHits.env definire: DETINPUT
CARF_OUTPUT_DATASET_NAME
CARF_OUTPUT_OWNER

3) definizione del file di input e del numero di eventi da processare:

in writeHits.orcarc assegnare: FZInputFiles
SimApplication$MaxEvents

4) Preparazione dell'environment:

> eval ‘scram runtime –csh writeHits.env‘
> cp writeHits.orcarc .orcarc

5) run:

> writeHits

La federazione in /myfederation_dir viene popolata di simHits !!

N.B. L'indirizzo della federazione in cui scrivere e‘ stato passato al programma dalla variabile OO_FD_BOOT , settata da fdinit.csh.

3. Come creare DIGIS, partendo dai simHits, e aggiungerli al DB

Occorrente:

a) SimHits da usare come input (appena fatti!)

b) Codice opportuno (writeDigis.*, in ExProduction, gia' compilato!)

Esecuzione:

0) N.B. !!! Se questo passo viene eseguito dopo un logout, si deve riinizializzare la federazione (e.g. Definire OO_DB_BOOT):

> source makefd.local myfederation_dir
(oppure: > source ./myfederation_dir/fdinit.csh)

1) Definire gli identificatori per i DB di input e output :

In writeDigis.env definire: CARF_INPUT_DATASET_NAME
CARF_INPUT_OWNER

(N.B. !!! devono essere gli stessi identificatori usati per l'output nel passo precedente!!)

CARF_OUTPUT_DATASET_NAME
CARF_OUTPUT_OWNER

N.B. Il dataset_name deve essere lo stesso dell'input (stessi eventi)
L'output_owner deve essere diverso (diverse informazioni)

2) Definire le modalita' del pileup (obbligatorio per la scrittura di Digis)

In writeDigis.orcarc specificare gli identificatori (dataset_name e carf_owner) del file con i simHits del PU.

Se non si vuole il PileUp basta mettere PUGenerator:AverageEvents = 0

3) Fissare il numero di eventi da processare:
RecApplication$MaxEvents, sempre in writeDigis.orcarc

4) Fissare i parametri rilevanti per la ricostruzione (tagli, soglie, rumori, ecc., attinenti i singoli rivelatori. v. writeDigis.orcarc.)

5) Preparare l'evironment:

```
> eval `scram runtime –csh`
> eval `scram runtime –csh writeDigis.env`
> cp writeDigis.orcarc .orcarc
```

6) RUN:

```
> writeDigis
```

Il DB in ./myfederation_dir si popola di Digis !!!

4. Come accedere alla federazione che abbiamo preparato, con un programma di analisi.

Occorrente:

codice opportuno.

Da: ~/user/ORCA_4_4_0/src :

```
> cvs co –r  March2001 Examples/Tutorial
```

(la dir Tutorial si affianca a ExProduction dentro Examples)

Anatomia: 1 file .cpp, 2 files .cc (l'analisi vera e propria e la gestione della n–tupla) + i relativi files .h in /interface (guardare!)

Esecuzione:

1) Building:
Nel BuildFile: <Group name=RecReader> (lettura dei Digis)

> scram b shared
> scram b bin

2) Definizione dell'input:
In dbreading.env definire:
il bootfile: OO_FD_BOOT,
il dataset_name: CARF_INPUT_DATASET_NAME
e l'input_owner: CARF_INPUT_OWNER

Nota 1 Se si proviene dal passo 3 OO_FD_BOOT e' gia' definito;

Nota 2 L'intput_dataset_name e l'input_owner devono essere quelli dell'output del passo 3;

Nota 3 > unsetenv CARF_OUTPUT_OWNER
(non puo' essere uguale all'input!)

3) Preparazione dell'environment:

> eval ‘scram runtime –csh‘
> eval ‘scram runtime –csh dbreading.env‘

(volendo si puo‘ preparare un .orcarc, e.g. con il n. di eventi)

4) RUN:

> ExTutNtuple

Viene creata una n–tupla: myntpl.hbook. (guardare!)

5. Esempio di "azione su domanda"

La stessa analisi puo‘ girare sugli Hits invece che sui Digis:

a) Nel BuildFile: <Group name=SimReader>

se non si vuole soprascrivere l’eseguibile esistente, specificare un altro nome:

<bin file=ExTutNtuple.cpp name=ExTutNtuple_1 > </bin>

> scram build bin

b) In dbreading.env cambiare il CARF_INPUT_OWNER per puntare agli Hits

> eval ‘scram runtime –csh dbreading.env‘

c) se non si vuole soprascrivere l’n–tupla gia‘ fatta, provvedere a salvarla prima del nuovo run (e.g. : > mv myntpl.hbook rec_ntuple.hbook)

> ExTutNtuple_1

d) Guardare la nuova ntupla:
i dati sono stati letti correttamente (−−> evnt) ma le variabili sono tutte nulle: esse infatti si riferiscono ai RecHits che in questo run non sono stati aperti...

E‘ NECESSARIO INDICARE NEL BuildFile LE LIBRERIE RILEVANTI PER FARE LA RICOSTRUZIONE SU DOMANDA:

In questo caso il programma riempie la n−tupla con le informazioni dei calorimetri: le librerie che servono a produrre Digis dai SimHits dei calorimetri sono contenute in:

<Group name=CaloRHitWriter>

e) Rifare il building e il run

f) Riguardare la n−tupla: ora e‘ riempita!

g) Confrontare la nuova n−tupla con rec_ntuple.hbook: sono identiche?

Le differenze sono dovute ai parametri della ricostruzione (soglie, rumori, tagli,..): in writeDigis molti di essi erano specificati, in ExTutNtuple no, sono stati usati quindi i rispettivi valori di default.

Scegliendo le librerie nel BuildFile e i parametri di input nel file .orcarc si puo‘ governare ogni passo della ricostruzione!

http::/cmsdoc.cern.ch/ORCA

HAVE FUN !!!!