1. Introduction

In this document we give a technical description of the layout, data flow and features of the so-called “Milano Data Transfer”.

2. General Description

The main goal of the Milano Data Transfer (from now on called DT) is to move data from CERN to CNAF, located in Bologna, Italy, which will be the Master Copy of AMS data.
This version of DT has also been tested to transfer data from CERN to Milano and other test sites [1].

The system is also used, with a slight modification, to transfer MC and REC data from remote sites to CERN. This version of the DT is in production to transfer data from Milano and CNAF to CERN. It can be easily used by any other remote location.

3. Technical Description

The core of the DT is a finite state automa, written in Python, whose states are stored in a MySQL DataBase (DTDB) and whose transition functions are described in an XML file and implemented as Perl5 scripts, which can be run stand-alone as well. The actual transfer over the network is done by means of a Network Transfer Protocol, which can be **`bbftp`** ([2]) or **`globus-url-copy`** if the receiving site is grid-enabled.
The local and remote sites can communicate one with another using a Network Daemon, **`Nuovo_Protocollo_Server_DT_bbftp.pl`**, written in Perl5, which enables messages to be sent from one to the other and thus provide information about files on the other location.
For more details refer to [3] and [4].

Henceforth, the two locations will be called: **SENDER**, meaning the one that has the files to be sent, and **RECEIVER**, meaning the one that has to receive the files. Please note that this same nomenclature is used the directory tree used for implementation on the various servers.

3.1 States of the FSA

More detailed description of the various programs will be given later; we now want to provide a description of the possible states:

- TO_BE_TRANSFERED: new file found and ready for transferring.
  This state is pivoted by Perl5 script **`cerca_nuovi.pl`**

- TRANSFERRED: file has been transferred without Network Transfer Protocol errors.
  This state is pivoted by Perl5 script **`trasferisci.pl`**
- SIZE_OK: the size of the file at the RECEIVER and at SENDER are the same.
  This state is pivoted by Perl5 script **`controlla_size.pl`**
- VALIDATED: the CRC of the file at RECEIVER and at SENDER are the same.
  This state is pivoted by Perl5 script **`controlla_crc.pl`**

3.2 Finite State Automa

The Finite State Automa (FSA), based on Python and described in greater detail elsewhere ([4]), is contained in **`finitestates.py`**. The FSA states are med persistent in a MySQL Database, while FSA State Transitions are defined as *rules* in a FSA Description, implemented as an XML file.
The FSA runs continuously, as a daemon; when it finds an element with a given status in the database table, it looks for a rule that can be applied to it and runs the shell command. When the command returns, the FSA updates the status in the DB, depending on the return code of the shell command. Generally, the FSA tries to run simultaneously, in dierent shells, one command for each rule, as soon as items are available for processing. This means that commands from dierent rules are executed in parallel, on different items, and sequences of commands coming from the same rule, on different items, are performed in series. If the commands in two di fferent rules share the same resource, it is useful to impose that they are executed in series; this is done by using Pythons Queue mechanism for threads.
It should be noted that the version in production at the moment of writing, since based on Python 2.5, is fully multi-threaded, but due to Pythons internal Global Interpreter Lock (GIL) mehcanism, only one thread is run at a time. A new release, based on Python's **`multiprocessing`**, is currently under development. Nevertheless, given the DT goal, this is not an issue since different rules can be processed at the same time.

3.4 State Transition Functions

We now briefly review each single transition function Perl5 script. Common use functions and variables and site-dependent values are defined in a Perl Module, **`lib/McMover.pm`**

3.4.1 `cerca_nuovi.pl`

If RECEIVER is CERN, new files are looked for in AMS Oracle DataBase (XXXXX). All files are selected, and are then searched in the DT DataBase (DTDB).
If RECEIVER is some remote location (e.g. CNAF), new files are looked in some specific directory, and have to satisfy a particular condition about file name (!~ /pend/), which means that the file is not already being processed

by some DT process.
In either case, ,if a new file is found, the DTDB is updated with status **TO_BE_TRANSFERED**

3.4.2 `trasferisci.pl`

This script handles the actual network transfer. It can use bbftp or globus-url-copy. It hsold be stressed here that the *egee grid* framework can be used, at the moment of writing, only to transfer data from CERN to CNAF[1]. If the pure network transfer is successful, the DTDB is updated with status **TRANSFERED.**
In particular, if RECEIVER is CNAF, files are directly written into CASTOR, by means of globus-url-copy's **`SRM`** protocol

3.4.3 `controlla_size.pl`

This script compares local and remote file sizes. The local file size has already been read from Oracle, while the remote size is checked by means of the **`Nuovo_Protocollo_Server_DT_bbftp.pl`**. If **RECEIVER** is CNAF, size is read from CASTOR using RFIO commands, else it's read from file system. If remote and local size are equal, the DTDB is updated with status **SIZE_OK**; if not, the status is changed to **TO_BE_TRANSFERED** again**.**

3.4.4 `controlla_crc.pl`

This script compares local and remote file CRCs. The local file CRC is read as adler32 of the CASTOR copy of the file or, should it not be available, calculated as V. Choutko's MC CRCR on the disk copy of the file, while the remote CRC is checked by means of the **`Nuovo_Protocollo_Server_DT_bbftp.pl`**. If **RECEIVER** is CNAF, CRC is read as adler32 from CASTOR[2], otherwise as V. Choutko's MC CRCR on file system.
If the CRCs are equal, the DTDB is updated with status **VALIDATED**; if not, the status is changed to **TO_BE_TRANSFERED** again**.**

3.5 Sites Comminication

RECEIVER and **SENDER** communicate by means of the **`Nuovo_Protocollo_Server_DT_bbftp.pl`** script. This is based on Perl5 IO::Socket::SSL module and uses X509 certificates, issued by a dedicated CA, for host and user authentication.

The main actions it can take care of are checking size and checking crc.

---

1 Mainly because of bureaucratic issues related to the creation of an AMS *Virtual Organization*.
2 This is the only solution available, since files are directly written to CASTOR by mens of SRM. This induces a certain latency, since adler32 is calculated on CASTOR only after the file has been migrated to tape. Average latency has been evaluated by us ~ 20 hours

3.6 Running the DT

In general, the DT runs at **SENDER** as a dedicated stand-alone daemon, **`finitestates.py lancia.xml`**. All processes are owned by user **`mcmover`**.

At RECEIVER, the only daemon is **`Nuovo_Protocollo_Server_DT_bbftp.pl`**, owned by user **`mcmover`**.

3.7 Database replication

Since the FSA states are stored in a MySQL Database, which actually acts as a book-keeping DB as well, the DB is replicated at Milano Bicocca INFN AMS Data Center. The DB is replicated by means of a Perl5 script which runs every hour and updates at Milano a copy of the DB. At the moment of writing, we are evaluating to move the whole DTDB to AMS's ORACLE and to adopt ORACLE replication mechanisms.

XXX. References

[1] M. Boschini, “Redundancy for the Data Transfer System”
http://indico.cern.ch/getFile.py/access?contribId=3&resId=0&materialId=slides&confId=58285
[2] BBFTP: http://doc.in2p3.fr/bbftp/
[3] Boschini et al. “*New Generation Data Transfer for AMS-02*”,
http://villaolmo.mib.infn.it/ICATPP10th_2007/Software%20Applications/Boschini.pdf
[4] Boschini et al, “*Finite state automa for parallelization of time-expensive operations”*
http://villaolmo.mib.infn.it/ICATPP10th_2007/Software%20Applications/Brogioli.pdf

Appendix A: The FSA XML Definition

lancia.xml

```
<finitestates_Data sleep="120" log="log/transfer.log">


  <database host="pcamsd0.cern.ch" user="XXXX" passwd="YYY" db="AMSMC02" table
="ams_DT_Data" primarykey="name" status="status" time="insert_time">
    <field>dir</field>
    <field>name</field>
    <field>status</field>
    <field>adler32</field>
    <field>size</field>
    <field>insert_time</field>
  </database>

  <rule>
    <command>
      perl cerca_nuovi.pl
    </command>
    <if code="0" status="TO_BE_TRANSFERED"/>
  </rule>

  <rule>
    <status>TO_BE_TRANSFERED</status>
    <command>
      perl trasferisci.pl $dir $name
    </command>
    <if code="0" status="TRANSFERED" />
    <else status="TO_BE_TRANSFERED" />
  </rule>

  <rule>
    <status>TRANSFERED</status>
    <command>
      perl controlla_size.pl $dir $name $size
    </command>
    <if code="0" status="SIZE_OK" />
    <if code="1" status="TO_BE_TRANSFERED" />
  </rule>

  <rule sleep="36000">
    <status>SIZE_OK</status>
    <command>
      perl controlla_crc.pl $dir $name
    </command>
    <if code="0" status="VALIDATED" />
    <if code="1" status="TO_BE_TRANSFERED" />
  </rule>

</finitestates_Data>
```

Appendix B: DataBase

```
mysql> describe ams_XXX;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| dir         | varchar(80) |      | PRI |         |       |
| name        | varchar(80) |      | PRI |         |       |
| status      | varchar(20) | YES  |     | NULL    |       |
| adler32     | varchar(50) | YES  |     | NULL    |       |
| size        | bigint(128) | YES  |     | NULL    |       |
| insert_time | int(20)     | YES  |     | NULL    |       |
| other       | varchar(80) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

Tables:

ams_ASDC: table used for Redundancy Test

ams_DAVIDE: table used for Redundancy Test

ams_DT: table used for CERN->CNAF DT of MC data

ams_DT_RAW: table used for CERN->CNAF DT of RAW data

ams_DT_Data: table used for CERN->CNAF DT of REC data

Appendix C: Software Directory Tree

Appendix D: Involved Hosts

CERN

- **pcamsd0.cern.ch:** main server
- **pcamsd2.cern.ch:** redundancy server

CNAF

- **ui03-lcg.cr.cnaf.infn.it**: main server
- **ui04-lcg.cr.cnaf.infn.it:** redundancy server

Milano Bicocca

- **amsmc10.mib.infn.it**: DTDB replication server.