File Transfer Agents

FTS Workshop

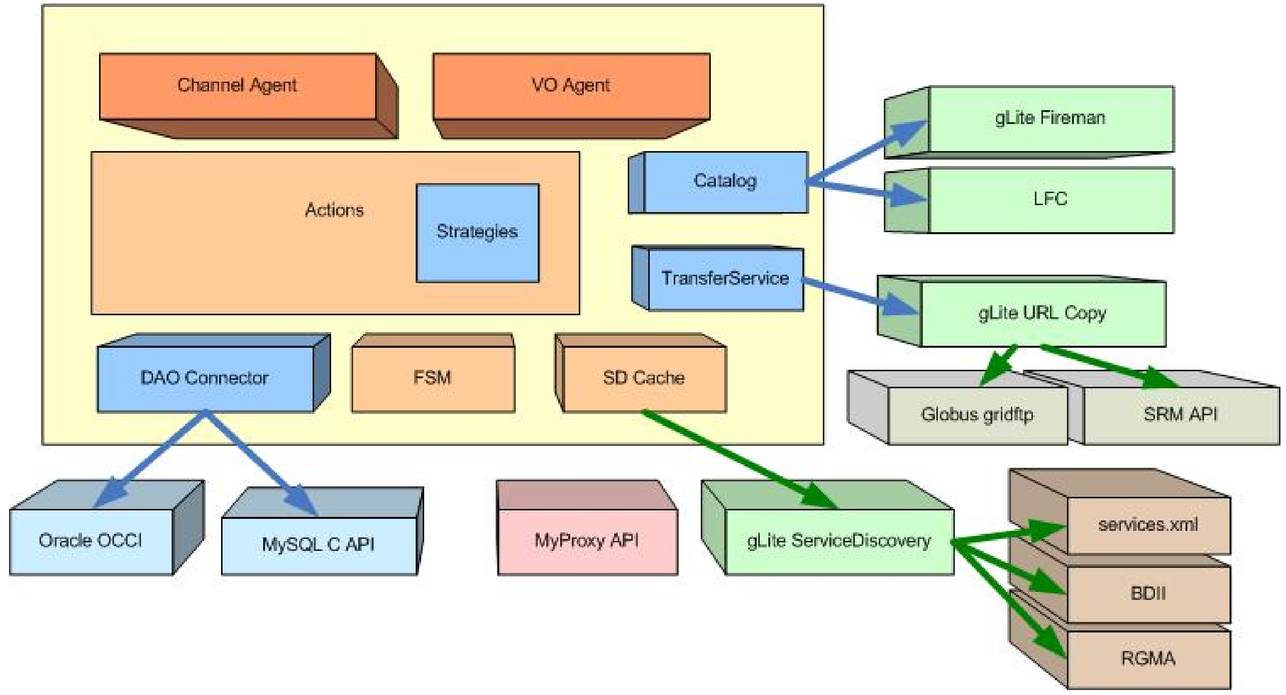
16 November 2005

Paolo Badino
IT/GD, CERN

Contents

Architecture

- 1 CVS module, several components organized in layers
- Plug-ins
  - Database (Oracle & MySQL)
  - Catalog (gLite Fireman, LFC?)
  - Transfer Service (glite-url-copy, srmcopy)
- Unit tests
  - 363, including some regression tests



Agent Structure

- 2 logical services
  - VO-Agent
  - Channel-Agent
- Share the same structure
  - Specific actions that are executed periodically
    - Command Pattern
    - Stateless
  - Flexible
    - Easy to maintain
    - Easy to extend

VO-Agent Types

- FTS
  - Default behavior
  - Transfer files using SURLs/TURLs
- FPS
  - Create replicas in a given Site/SE using Logical Names (LFN or GUID)
  - Can work also in “FTS” Mode
    - Accept SURLs/TURLs
    - If a logical name is provided, register also the new replica
    - Different final state (**Finished** instead of **Done**)

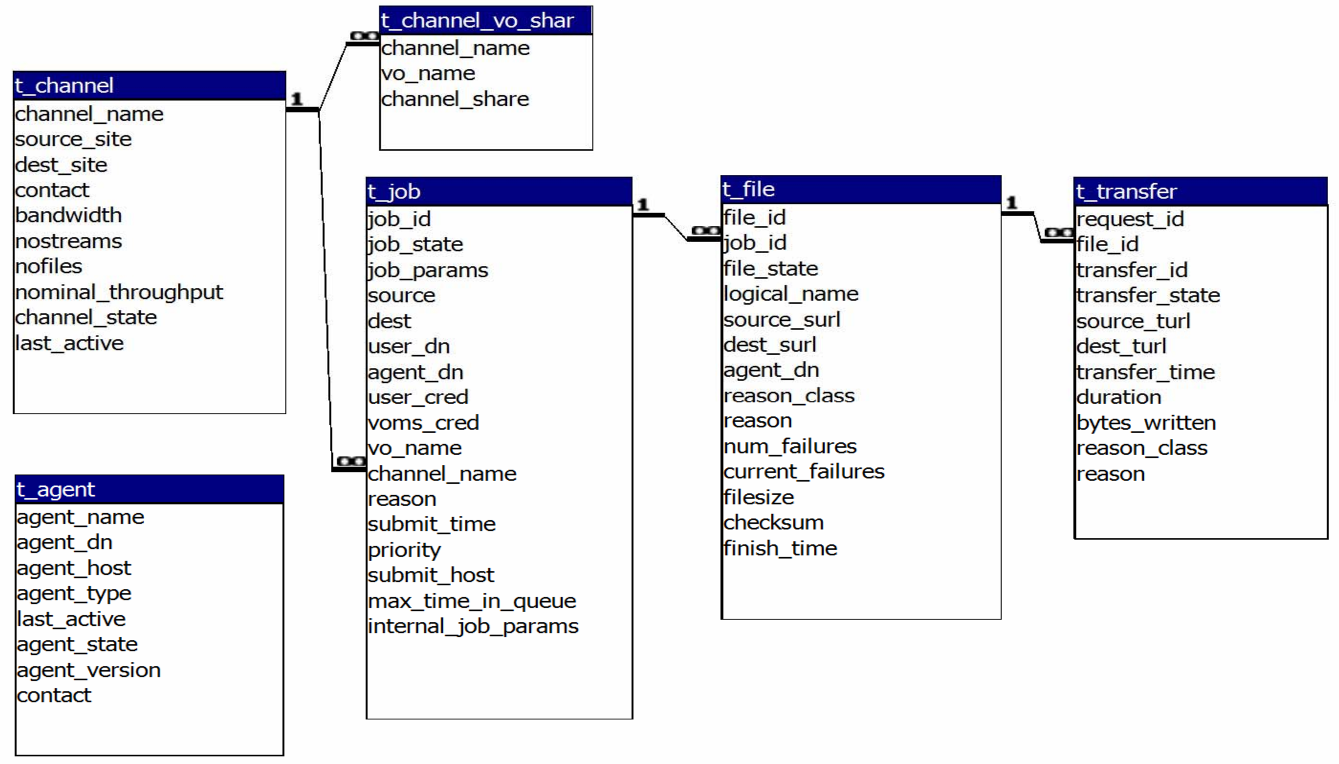
Responsibilities

Service	
VO Agent	
Responsibilities	**Collaborators**
▪ Channel Allocation ▪ Retry failed transfers ▪ Cancel pending transfers ▪ Intra-VO scheduling ▪ Resolve Logical Names into SURLs ▪ Register new replicas ▪ Prestaging ▪ Dynamic Priority	▪ Service Discovery (InfoSys) ▪ MyProxy ▪ Catalog (Fireman,…) ▪ SRM (for prestaging)

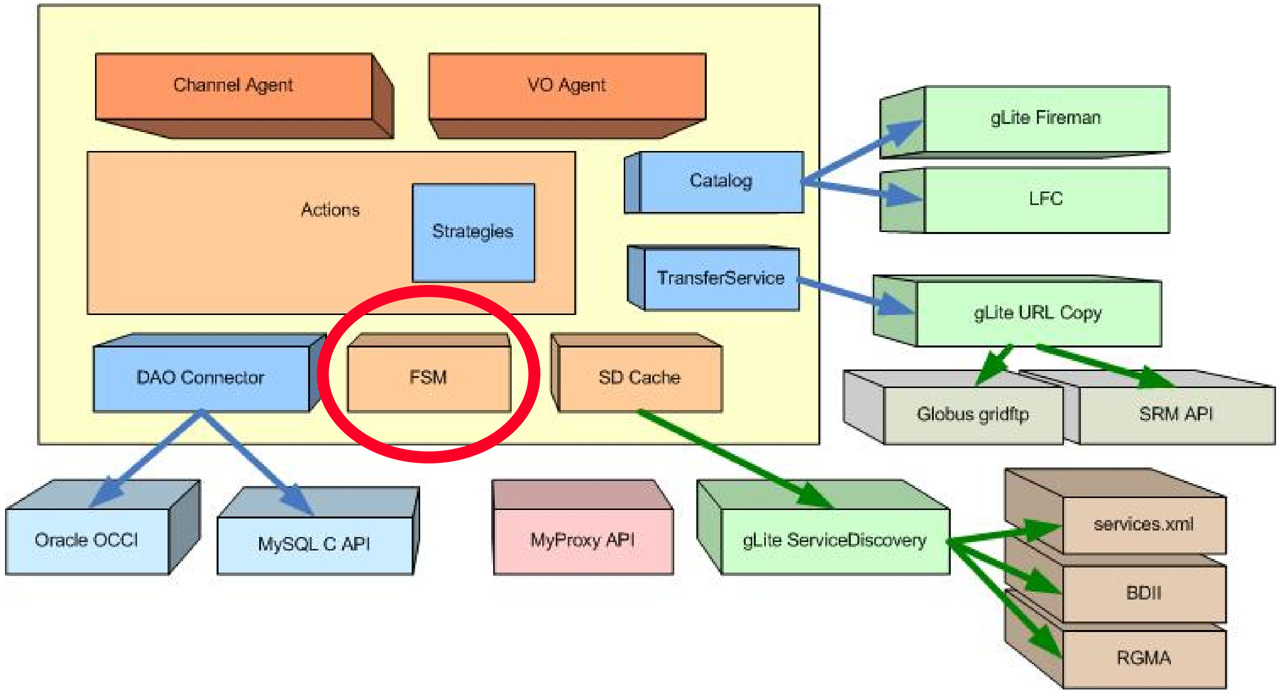
- Only FPS
- Not yet implemented

Service	
Channel Agent	
Responsibilities	**Collaborators**
▪ Start and monitor transfers ▪ Cancel active transfers ▪ Inter-VO scheduling (VO Share) ▪ Channel monitoring	▪ Service Discovery (InfoSys) ▪ MyProxy ▪ TransferService (glite-url-copy, srmcopy,…)

DB Schema


t_channel_vo_shar
channel_name
vo_name
channel_share
t_channel
channel_name
source_site
dest_site
contact
bandwidth
nostreams
nofiles
nominal_throughput
channel_state
last_active
t_agent
agent_name
agent_dn
agent_host
agent_type
last_active
agent_state
agent_version
contact
t_job
job_id
job_state
job_params
source
dest
user_dn
agent_dn
user_cred
voms_cred
vo_name
channel_name
reason
submit_time
priority
submit_host
max_time_in_queue
internal_job_params
t_file
file_id
job_id
file_state
logical_name
source_surl
dest_surl
agent_dn
reason_class
reason
num_failures
current_failures
filesize
checksum
finish_time
t_transfer
request_id
file_id
transfer_id
transfer_state
source_turl
dest_turl
transfer_time
duration
bytes_written
reason_class
reason
1
∞
1
∞
1
∞
∞

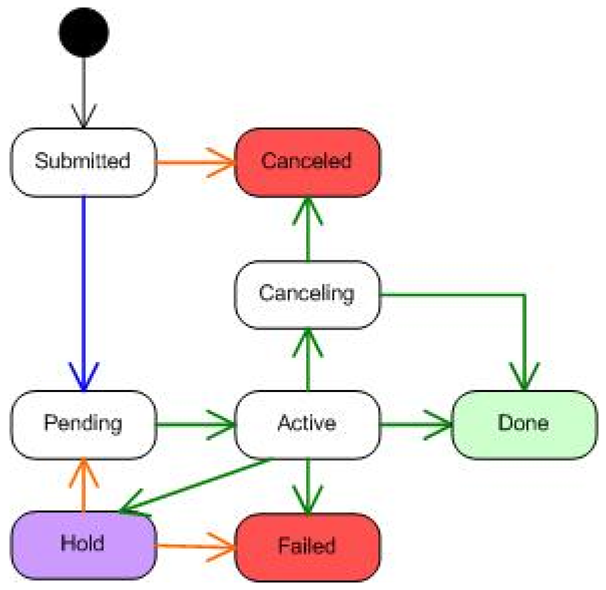
State Machines



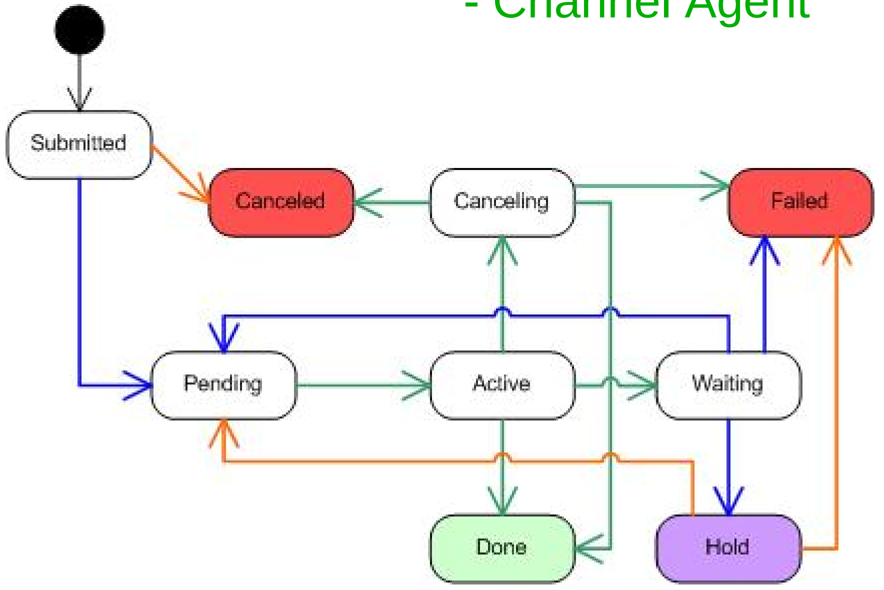
FTS State Machines

Simplified

- FTS WS
- VO Agent
- Channel Agent



Job

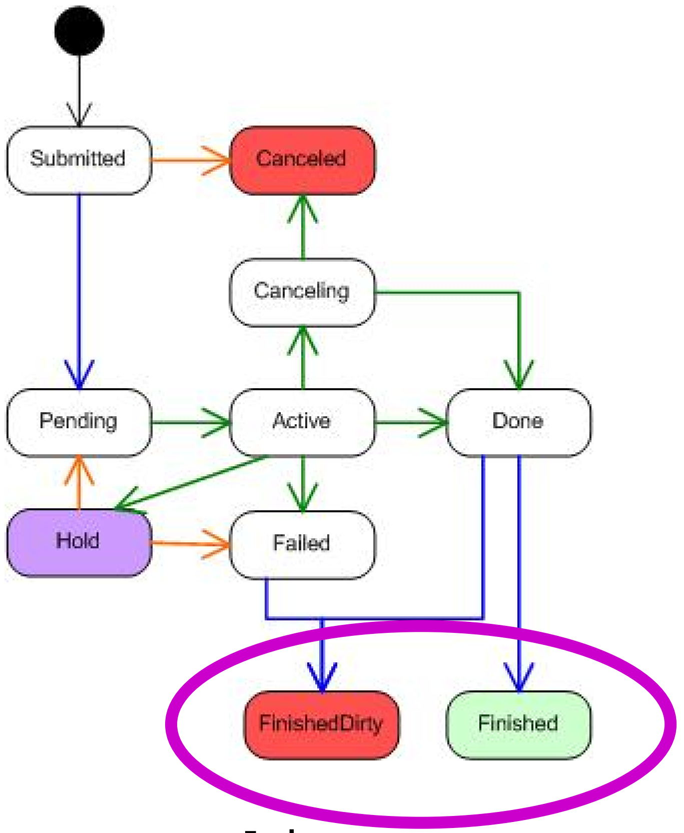


File

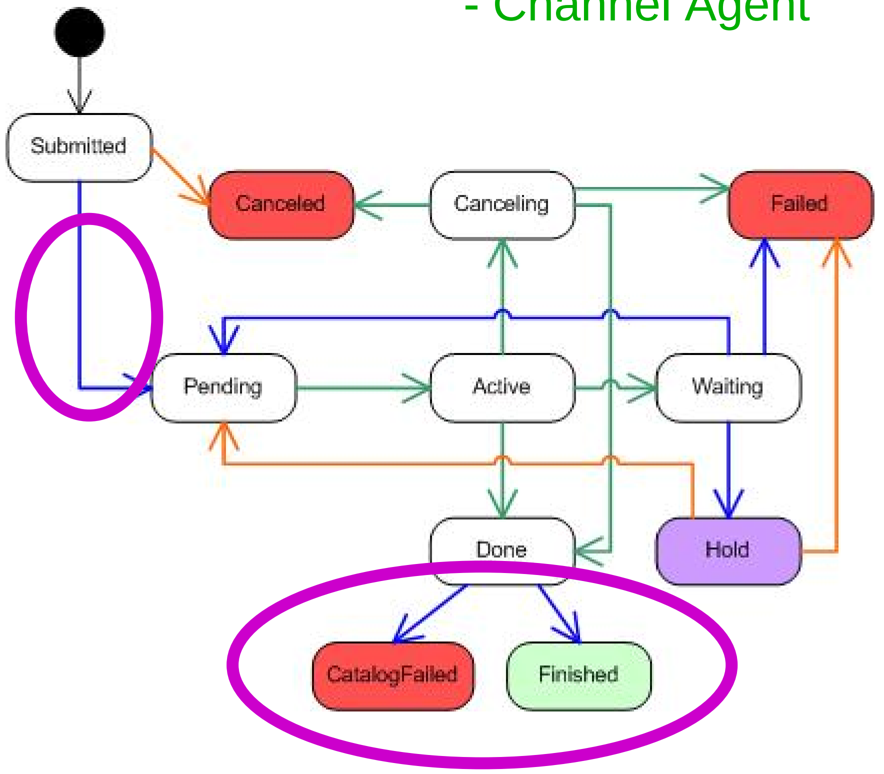
FPS State Machines

Simplified

- FTS WS
- VO Agent
- Channel Agent

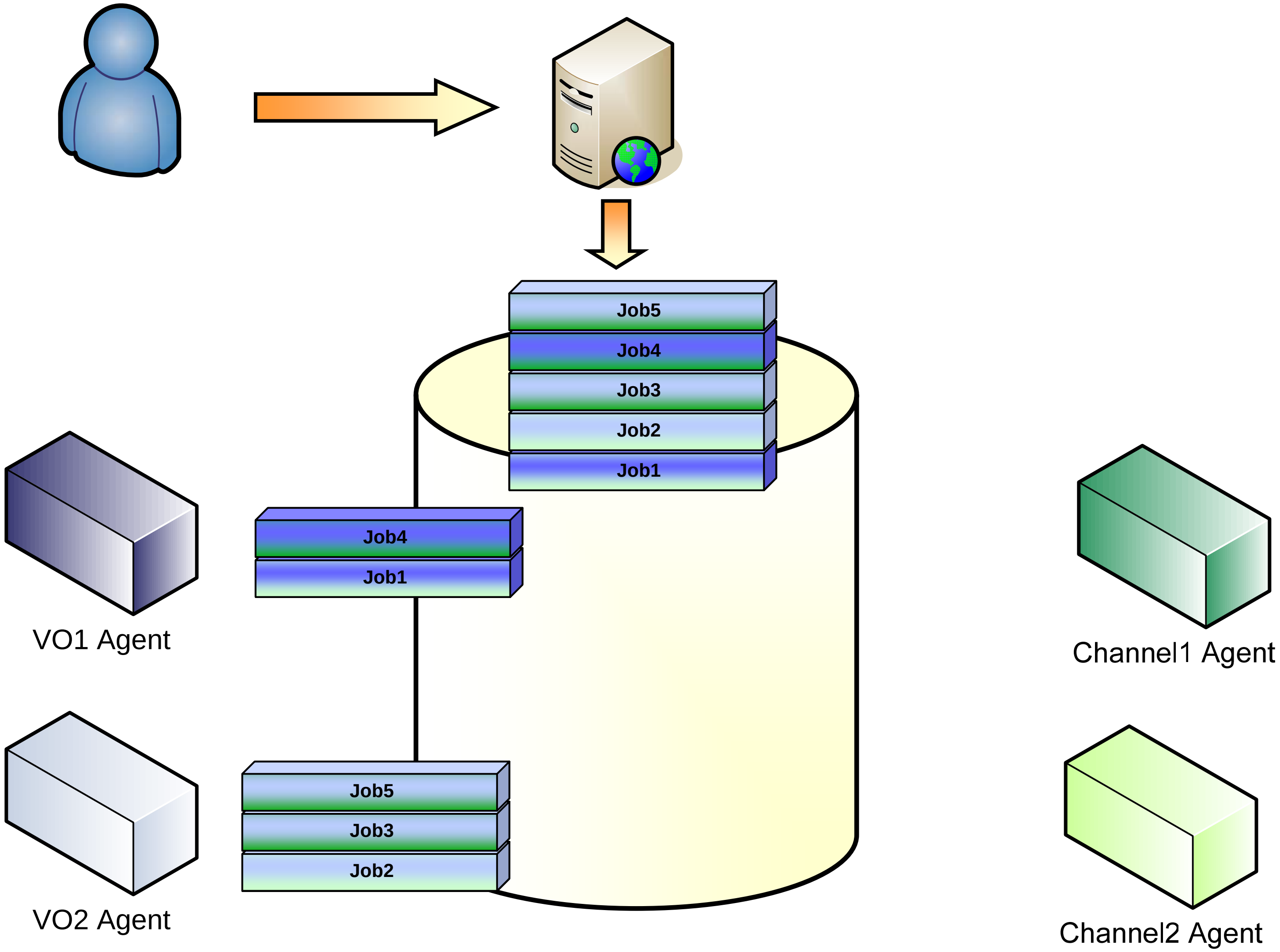


Job



File

Inter-VO Scheduling

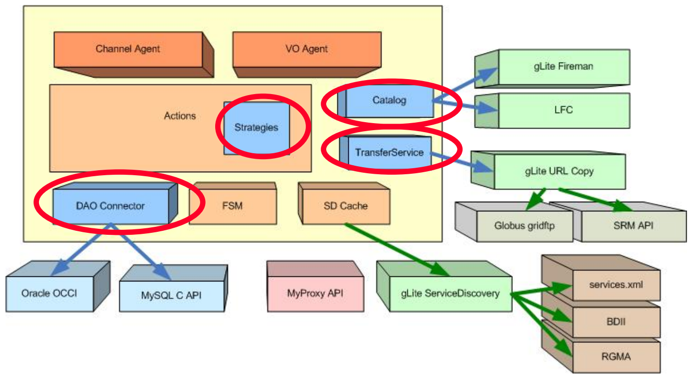


Intra-VO Scheduling

- Each job has a priority
  - 5 – High
  - 3 – Normal (Default)
  - 1 – Low
- Jobs are ordered within the same VO based on that priority and then the submit time
- Doesn't affect the scheduling of other VOs
- Priority can be modified by the VO Managers
- In the future, VOs will be able to modify the Jobs' priority on the queue

Agent Framework

- Framework where experiments can plug their own software to customize the Agent behavior
  - Catalog Interaction
  - Override Actions
- All the interaction with external libraries/services is handled using plug-ins
  - AbstractFactory Pattern
  - org.glite.data.config-service (ComponentConfigurator Pattern)



Catalog Interaction

- The CatalogService interface declares 3 methods
  - Expects bulk operations
  - Exists for gLite Fireman
  - Support for LFC has to be added

```
class CatalogService {
public:
    // Check if the user is authorized to replicate the file
    virtual bool checkPermissions(const StringArray& logicals) /*throw (CatalogServiceException)*/ = 0;

    // List the names of all the replicas associated to the given Logical Names
    virtual void listSurls(const StringArray& logicals,
                           std::vector<StringArray>& surls) /*throw (CatalogServiceException)*/ = 0;

    // Register the name of new replicas
    virtual void registerSurls(const StringPairArray& names) /*throw (CatalogServiceException)*/ = 0;
 };
```

- Support for multiple catalogs at the same time missing
  - Difficult to force consistency
  - Can be implemented by the experiment using the Delegation Pattern

Actions

- VO Agent
  - Allocate
  - Retry
  - Cancel Pending
  - Resolve
  - Register
  - Pre-stage
- Channel Agent
  - Fetch
  - Check State
  - Cancel Active
- All the actions are defined as C++ classes and can be overridden
  - Quite complex: requires to duplicate the code to get the relevant data from the DB
  - Not all actions need customization

⇨ Adopt the Strategy Pattern

Retry Strategy

- Basic Retry Strategy
  - Retry each file up to a configurable number of times (default: 3)
  - Move it to Hold (configurable - Failed in case of LHCb)
  - Doesn't take into account the error reason
  - …but we know this is not enough
- Possible improvements
  - Define a different policy for each user/group
  - Delete destination file if exists
  - Depending on the error reason, fail fast or wait more
- Too complex to define a generic policy that can be configured for all needs

⇨ Let the experiments to specify their strategies

Retry Strategy: C++

Provide a C++ Action class

- Define a class that provide the logic to decide when a file should be retried
  - Inherits from abstract class ***RetryStrategy***
  - Implements the ***apply*** method
- Define a class that overrides the default Retry:
  - Inherits from the ***Retry*** base class:
    - Specify an Action type name
    - Specify the RetryStrategy class to be used
- Define a class to load and configure the module dynamically
  - Inherits from ***glite::data:config::ComponentConfiguration***

Retry Strategy: C++ Example

- Define the ***RetryStrategy*** Class

```
class MyRetryStrategy : public RetryStrategy {
public:
    // ... constructors & destructor

    // Apply the policy defined by this strategy object
    virtual RetryResult apply(const model::Job& job,
                              const model::File& file,
                              const TransferArray& transfers) /*throw (ExecuteException)*/;
};
```

```
// apply the strategy
RetryStrategy::RetryResult MyRetryStrategy::apply(const Job& job,const File& file,
                                      const TransferArray& transfers) /*throw (ExecuteException)*/{
    RetryResult result = RetryStrategy::WAIT;
    time_t current;time(&current);
    if((file.finishTime + 600) > current){ // Check the elapsed time on last failure
        result = (0 == rand() % 2)?RetryStrategy::RETRY:RetryStrategy::HOLD;
    }
    return result;
}
```

Retry Strategy: C++ Example (2)

- Define the ***Retry*** Action Class

```
class MyRetry : public Retry {
    // Declare Action Factory Method
    DECLARE_FACTORY_METHOD( MyRetry , "my:myRetry");
public:
    // ... constructors & destructor
};
```

```
// Register ActionFactory Method
REGISTER_FACTORY_METHOD( MyRetry );

// Constructor
MyRetry::MyRetry():Retry("MyRetry",new MyRetryStrategy()){}
```

Retry Strategy: C++ Example (3)

- Define the Module Configuration Class

```
class MyRetryConfig :  public glite::config::ComponentConfiguration  {
public:
    // Called during initialization
    virtual int init(const Params& params){return 0;}
    // Configure during configuration
    virtual int config(const Params& params){return 0;}
    // Called when the agents is started, stopped or finalized
    virtual int start(){return 0;}
    virtual int stop(){return 0;}
    virtual int fini(){return 0;}
    // … constructors & destructor
};
```

```
extern "C" {
// Create Component instance
ComponentConfiguration * create_glite_component(){
    return new MyRetryConfig();
}
// destroy_glite_component
void destroy_glite_component(ComponentConfiguration * component){
    if(0 != r) delete r;
}
} // End extern "C"
MyRetryConfig::MyRetryConfig():ComponentConfiguration("tranfser-agent-my-retryr-action"){}
```

Retry Strategy: C++ Example (4)

- Define a new configuration template
  - Create a template files and store it in
```
/opt/glite/share/config/glite-data-transfer-agents
```
e.g.
```
cd /opt/glite/share/config/glite-data-transfer-agents/
cp glite-transfer-vo-agent-fts-oracle.config.xml glite-transfer-vo-agent-fts-myretry-
    oracle.config.xml
cp glite-transfer-vo-agent-fts-mysql.config.xml glite-transfer-vo-agent-fts-myretry-
    mysql.config.xml
```
  - Add the lines to load your module
```
<!-- … →
<component name="transfer-agent-vo-actions">
<!-- … →
</component>
<component name="transfer-agent-my-retry-action">
  <config-template>
    <description>The module that contains my retry action</description>
    <lib>libmy_retry_action.so</lib>
 </config-template>
</component>
<!-- … →
```
- Set the new Action Type
  - Edit the configuration file
```
<instance name="myvo" service="transfer-vo-agent-fts-myretry" description='Instance created by script'>
  <parameters>
     <transfer-vo-agent.Retry_Type value="my:MyRetry"/>
  </parameters>
</instance>
```

Retry Strategy: Python

Use a Python module

- Provide a python script that contains the retry logic
- The script is executed inside the VO-Agent process
- Configure the VO Agent in order to use the PythonRetry action
  - Few parameters to add to the XML configuration file:
    - PYTHONPATH
    - The name of the module

Only available in gLite 1.5

Expects Python 2.2

Retry Strategy: Python Example

- Define the Python script

```
import time
import glite.fts
import glite.fts.utils
# Declare the Retry Version supported by this script
def RetryVersion():
    return "1.0"
# Apply my Retry Strategy to the given File
def Retry(job,file,transfers):
    result = glite.fts.RetryResult.Wait
    glite.fts.utils.LogDebug("myretry","My Retry Script Called for file %s", % (file.id))
    # Check File Failures
    if(file.currentFailures >= 3) :
        glite.fts.utils.LogDebug("Too many failures (%d) for file %s" % (file.currentFailures,file.id))
        result = glite.fts.RetryResult.Hold
    else:
        transfer_time = 0
        current = time.time()
        # Check if it's time to resubmit the File
        if (current > (file.finishTime + 600)) :
            result = glite.fts.RetryResult.Retry
    return result;
```

Retry Strategy: Python Example (4)

- Use the new Retry Strategy
  - Edit the configuration file

```
<instance name="myvo" service="transfer-vo-agent-fts-python" description='Instance created by
    script'>
  <parameters>
     <transfer-agent-python.PythonPath value="${my.python.path}"/>
     <transfer-agent-vo-actions-python.RetryModule value="my_retry"/>
  </parameters>
</instance>
```

Python Retry

- The module should provide two methods:
  - Retry
    - Define the retry logic
    - Take as input the File instance to evaluate, the related Job and all the Transfers
    - Should return one of the following values:
      - RetryResult.Retry
      - RetryResult.Hold
      - RetryResult.Fail
      - RetryResult.Wait
  - RetryVersion
    - return the version associated to a given signature of the Retry method (now “1.0”)
    - Needed in order to support legacy scripts
      - The FTA will call the Retry function with the proper parameters depending on the value returned by this function

FTS Python Modules

- Wraps some FTA functions
  - ***glite.fts*** provides the classes and enums corresponding the tables in the DB schema
    - ***Job***, ***File***, ***Transfer***, ***Agent*** classes
    - ***GetSchemaVersion*** returns the version of the DB schema
  - ***glite.fts.utils*** wraps the log functions and provides some helpers
    - ***Log***, ***LogDebug***, ***LogInfo***, ***LogWarn***, ***LogError***
    - ***URL*** class: parse an SURL
    - ***GetProxyFileName***: retrieve the name of the local file containing the proxy certificate of the user owning a given job
  - ***glite.fts.sd*** wraps some SD functions
    - ***GetService***, ***GetServiceByType***, ***GetAssociatedService***, ***GetServiceProperty***, ***GetSiteName***
- Needs to be documented☹

Not intended to be used outside the FTA !

Extending the Agent

- Catalog Interaction
  - Replica Resolution
  - New Replica name generation
  - SURL Normalization
- Custom pre- and post-transfer actions
- Customize Prestaging
- Intra-VO Scheduling
  - Modify the jobs' priority dynamically
- State changes notification
  - Only for VO-Agent transitions:
    - Allocate -> Pending
    - Done -> Finished
    - Waiting -> Pending
  - Otherwise requires IPC or distributed event notification
- …

⇨ We need to define priorities

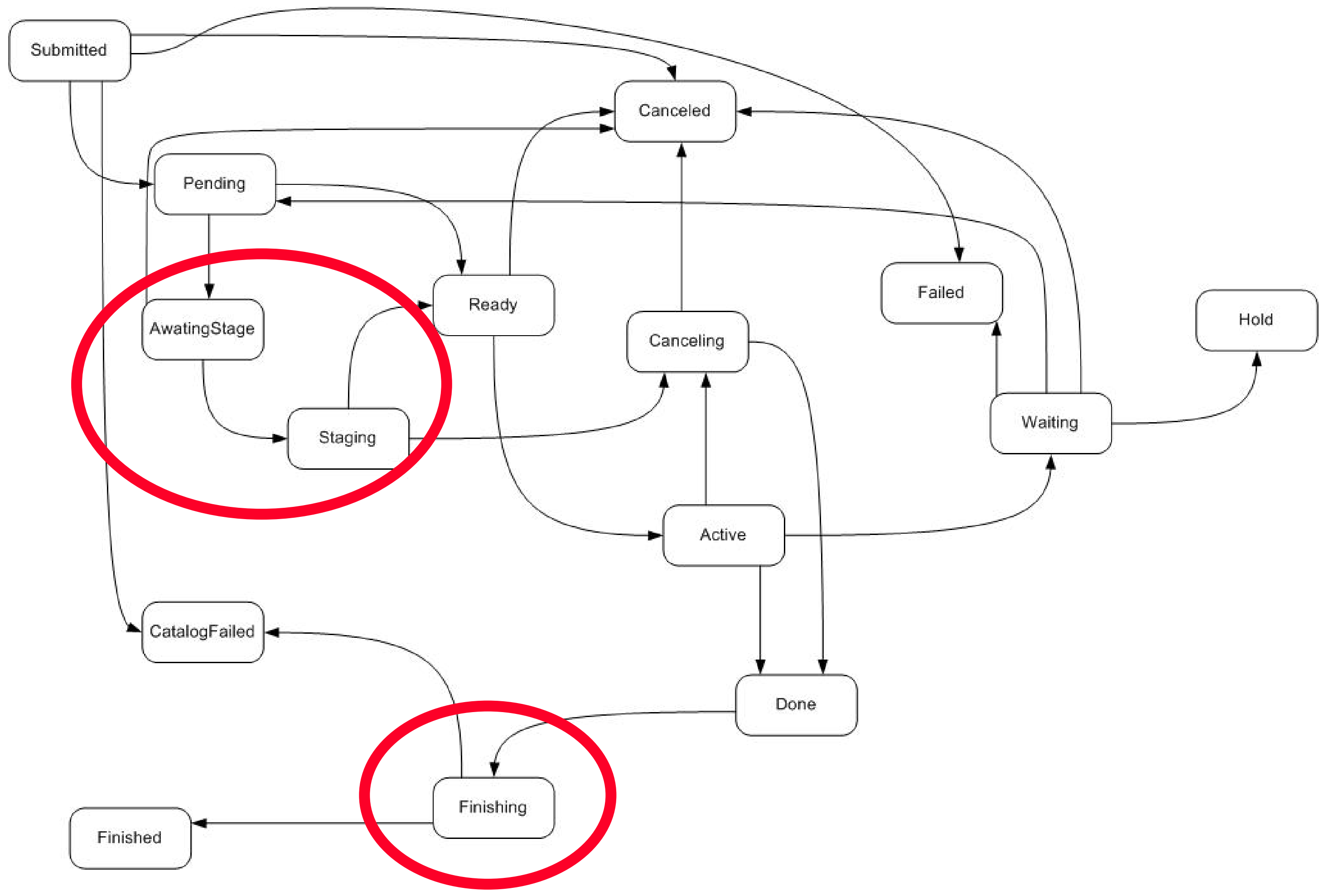
Known Issues

- Prestaging
  - Need additional states in the State Machines
  - Can we use SRM get?
- Catalog Registration
  - No retry on failures
    - Would require additional states
- FTS/FPS final state is different
  - In order to provide pre- and post-transfer hooks, FTS last transition should be performed by the VO-Agent

  ⇨ We need to modify the State Machines

- Evaluate how much the experiments' software depends on the actual states' names
  - Agree on names for final states

Next File State Machine



Documentation

- User & Developer Guides still missing ☹
- Together UML diagrams
  - org.glite.data.transfer-agents/doc/uml
  - Not completely up-to-date, but useful to understand the architecture
- README File
  - /opt/glite/share/doc/glite-data-transfer-agents
  - /opt/glite/share/doc/glite-data-config-service
- Source Code Comments

Questions