

```
## request task
__requestTask = None
## requestCallback = None
## exceptionCallback = None
__configPath = ""
```

# DataManagementSystem & RequestManagementSystem Status

```
return self.__processPool

def registerCallBack( self, callback ):
    """ register callback function executed after
    :param self: self reference
    :param callback: function definition
    """
    if not callable( callback ):
        return S_ERROR("Request callback cannot be registered, project '%s' isn't callable." % str(callback) )
    self.__requestCallback = callback
    return S_OK()

def registerExceptionCallback( self, exceptionCallback ):
    """ register exception callback, executed when requests fail
    :param self: self reference
    :param exceptionCallback: function definition
    """
    if not callable( exceptionCallback ):
        return S_ERROR("Exception callback cannot be registered, project '%s' isn't callable." % str(exceptionCallback) )
    self.__exceptionCallback = exceptionCallback

def setRequestTask( self, requestTask ):
    """ set requestTask type
    :param self: self reference
    :param type requestTask: RequestTask-derived class definition
    """
    if not issubclass( requestTask, RequestTask ):
        raise Exception("requestTask must be a subclass of RequestTask")
```

Krzysztof Daniel Ciba

29-31/10/2012



CERN  
THE INTERWARE

# DMS

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear ProcessPool'
    :param self: self reference
    :param type processPool: ProcessPool
    """
    if not self.__processPool:
        on subsequent calls, pass ProcessPool instance on first call, the same instance
        self.__processPool = processPool( minSize, maxQueueSize )
    else:
        self.__processPool( minSize, maxQueueSize )

    return self.__processPool

def registerRequestTask( self, callback ):
    """ register a check function executed after requestTask call
    :param self: self reference
    :param callable callback: function definition
    """
    if not callable(callback):
        raise S_ERROR("Exception callback must be callable, passed object '%s' isn't callable." % str(callback) )
    self.__requestCallback = callback
    return S_OK()

def FTSCleaningAgent( self, exceptionCallback ):
    """ a cleanup agent that raise an exception
    :param self: self reference
    :param callable exceptionCallback: function definition
    """
    if not callable(exceptionCallback):
        return S_ERROR("Exception callback cannot be registered, passed object '%s' isn't callable." % str(exceptionCallback) )
    self.__exceptionCallback = exceptionCallback

def setRequestTask( self, requestTask ):
    """ set requestTask type
    :param self: self reference
    :param type requestTask: RequestTask-derived class definition
    """
    if not issubclass( requestTask, RequestTask ):
        raise S_ERROR("requestTask must be a subclass of RequestTask")

    self.setRequestTaskType( requestTask )
```

- PART I: requests agents

- DMS requests at a glance
- multiprocessing
- refactoring

- PART II: checksum check

- FTS
- SE

- PART III: TransferDB cleaning

- FTSCleaningAgent

- PART IV: a glimpse of the future

- FTS3 and GFAL2
- XROOT SE



# PART I: DMS agents refactoring (DIRAC v6r3)

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear ProcessPool'
        - Mr. Spock
    """
    pass
    # on subsequent calls
    # ProcessPool instance on first call, the same instance
    # is used on subsequent calls
    if self._processPool is None:
        minSize = max( 2, self.__minProcessPerCycle )
        maxSize = min( self.__maxProcessPerCycle, self._maxQueueSize )
        self._processPool = ProcessPool( minSize, maxSize, self, __maxQueueSize )
    self._processPool.daemonize()
    return self._processPool

def registerCallBack( self, callback ):
    """ @param self: self reference
    """
    if not callable(callback):
        raise TypeError("Passed object '%s' isn't callable." % str(callback) )
    self.__requestCallback = callback
    return S_OK()

def registerExceptionCallback( self, exceptionCallback ):
    """ @param self: self reference
    """
    if not callable(exceptionCallback):
        return S_ERROR("Exception callback cannot be registered, passed object '%s' isn't callable." % str(exceptionCallback) )
    self.__exceptionCallback = exceptionCallback

def setRequestTask( self, requestTask ):
    """ set requestTask type
    """
    pass
    #param self: self reference
    #param type requestTask: RequestTask-derived class definition
    #param type requestTask, RequestTask:
    #param subclass( requestTask, RequestTask ):
```

A DIRAC request is:

- 1 or n **subrequests** + their **execution order** + **owner** + set of **subrequest's files**
- (de-)serialisation done by XML layer + DEncode
- kept in RequestDB (MySQL or fs backend)

Request processing – "state machine" using various agents:

```
if not callable(callback):
    raise TypeError("Passed object '%s' isn't callable." % str(callback) )
self.__requestCallback = callback
return S_OK()

def 'register' - RegistrationAgent
def 'removal' - RemovalAgent
def registerExceptionCallback( self, exceptionCallback ):
    """ @param self: self reference
    """
    if not callable(exceptionCallback):
        return S_ERROR("Exception callback cannot be registered, passed object '%s' isn't callable." % str(exceptionCallback) )
    self.__exceptionCallback = exceptionCallback

def setRequestTask( self, requestTask ):
    """ set requestTask type
    """
    pass
    #param self: self reference
    #param type requestTask: RequestTask-derived class definition
    #param type requestTask, RequestTask:
    #param subclass( requestTask, RequestTask ):
```



# PART I: old agents' (prior to DIRAC v6r2)

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear processor'
    :param self: self reference
    :return: new shiny Processpool instance on the fly
    """
    if not self.__processPool:
        minSize = max( 2, self.__minSize )
        self.__processPool = ProcessPool( self.__minSize, self.__maxProcessPerCycle )
        self.__processPool.demonize()
    return self.__processPool

def registerCallBa...:
    """ register call back
    :param self: self reference
    :param callback: callable object
    """
    if not callback:
        return S_ERROR("Request callback cannot be registered, passed object '%s' isn't callable." % str(callback))
    self.__requestCa...
    return S_OK()

def registerExcept...:
    """ register exception
    :param self: self reference
    :param exceptionCallback: callable object
    """
    if not exceptionCallback:
        return S_ERROR("Exception callback cannot be registered, passed object '%s' isn't callable." % str(exceptionCallback))
    self.__exceptionC...
    def setRequestTask( self, requestTask ):
        """ set requestTask type
        :param self: self reference
        :param type requestTask: RequestTask-derived class definition
        """
        if not issubclass( requestTask, RequestTask ):
            raise ValueError("requestTask must be a subclass of RequestTask")


```

- similar code everywhere (DRY!)
- obsolete operations (i.e. *replicate*, *replicateAndRegisterAndRemove*)
- parallel processing using ThreadPool
  - in plus
- speed up
  - low memory usage (shared instances)
  - global objects easily accesible (gConfig, gLogger, ...)
- in minus
  - common global environment for all threads (i.e. resetting of X509\_USER\_PROXY in one thread automatically transferred to all)
  - threads implementation in python (GIL) – not parallel at all (sic!)
- spaghetti code (if-elif-elif-...-elif-elif-else) everywhere
- tests and documentation missing!



## PART I: old inheritance

```

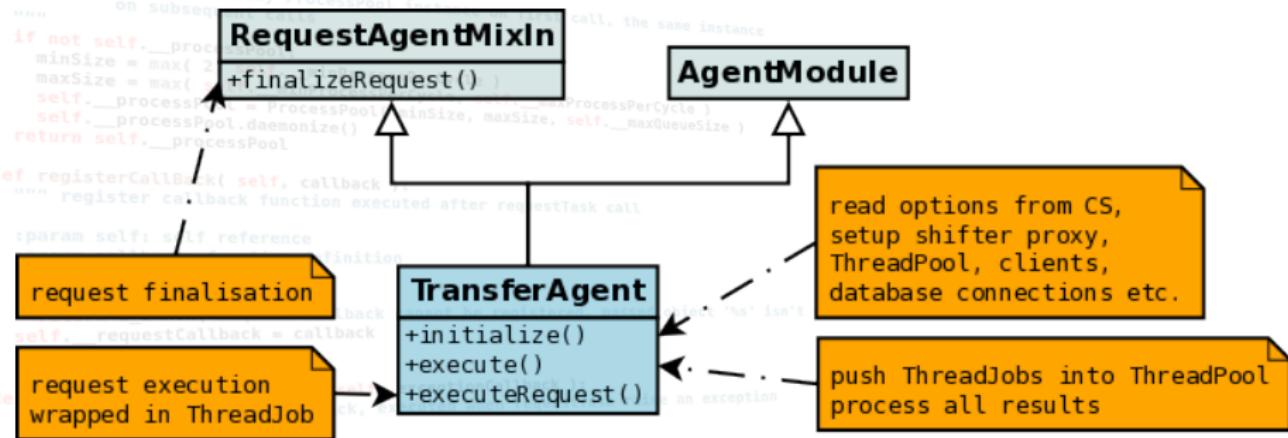
classDiagram
    class RequestAgentMixin {
        +finalizeRequest()
    }
    class TransferAgent {
        +initialize()
        +execute()
        +executeRequest()
    }
    class AgentModule

    RequestAgentMixin <|-- TransferAgent
    RequestAgentMixin --> AgentModule : call, the same instance
    TransferAgent --> AgentModule : read options from CS
    TransferAgent --> AgentModule : setup shifter proxy, ThreadPool, clients, database connections
    TransferAgent --> AgentModule : push ThreadJobs into process all results
    TransferAgent --> AgentModule : an exception
    RequestAgentMixin --> TransferAgent : request finalisation
    RequestAgentMixin --> TransferAgent : request execution wrapped in ThreadJob

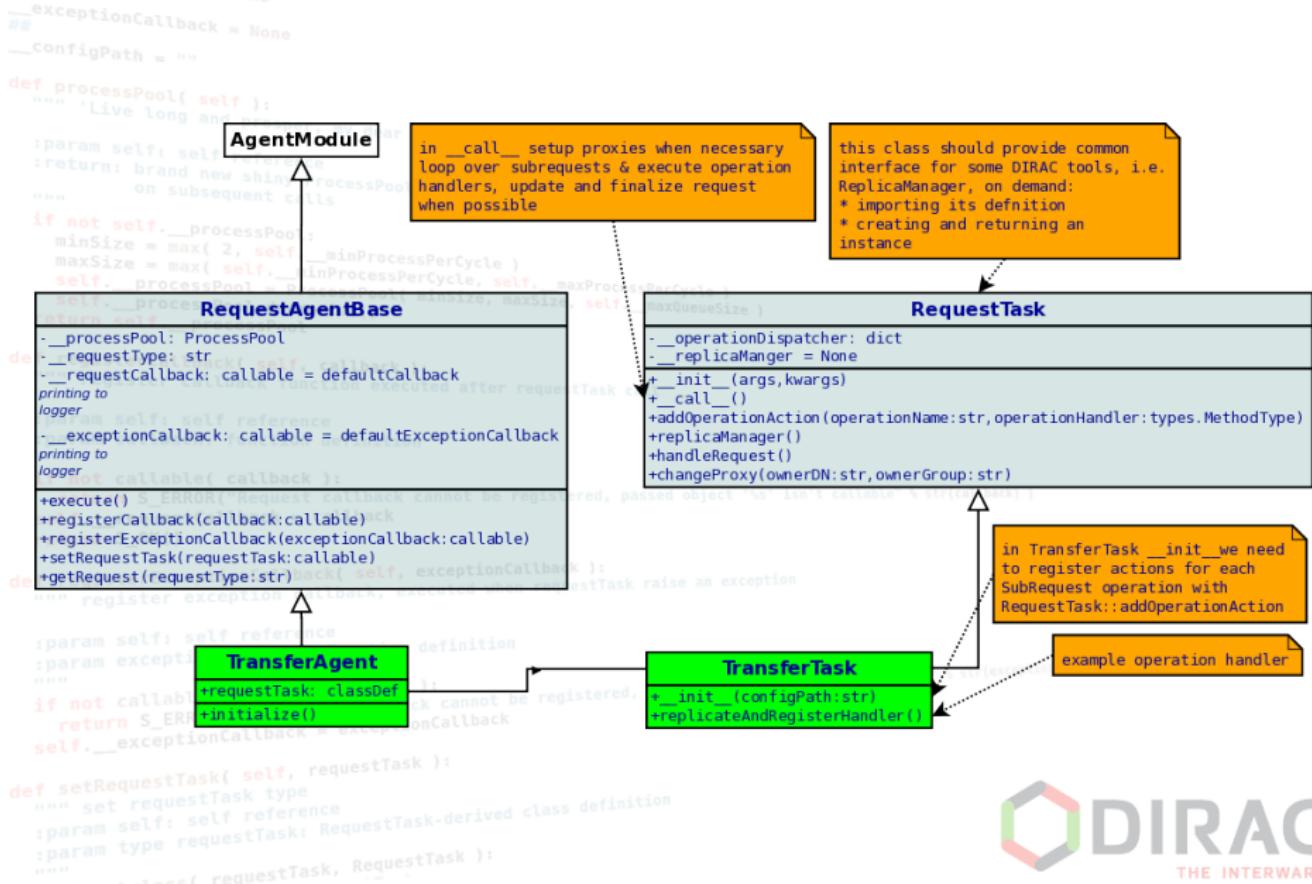
```

The diagram illustrates the class hierarchy and interactions between **RequestAgentMixin**, **TransferAgent**, and **AgentModule**.

- RequestAgentMixin** is a base class with a method `+finalizeRequest()`.
- TransferAgent** is a derived class that implements `+initialize()`, `+execute()`, and `+executeRequest()`.
- AgentModule** is another class that interacts with **TransferAgent**.
- RequestAgentMixin** has a relationship with **TransferAgent** labeled "request finalisation".
- RequestAgentMixin** has a relationship with **TransferAgent** labeled "request execution wrapped in ThreadJob".
- TransferAgent** has relationships with **AgentModule** labeled "call, the same instance", "read options from CS", "setup shifter proxy, ThreadPool, clients, database connections", and "push ThreadJobs into process all results".
- TransferAgent** also has a relationship with **AgentModule** labeled "an exception".



# PART I: new inheritance

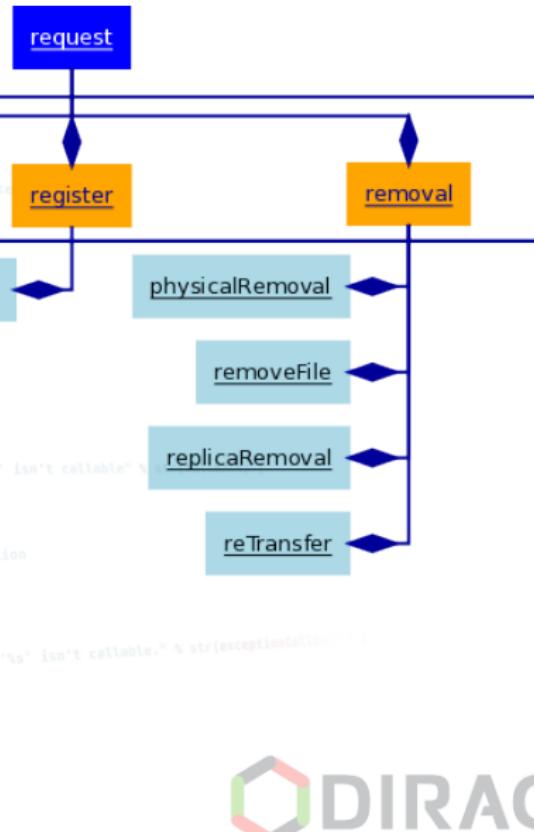


# PART I: old requests' types and operations

```
__exceptionCallback = None  
__configPath = ""
```

```
def processPool( self ):  
    """ Live long and prosper my dear ProcessPool - Mr. Spock  
    Requests' Types  
    Operations  
    class definition
```

```
    Requests' Types  
    Operations  
    class definition
```



## PART I: currently supported operations

The diagram illustrates the relationship between the `request` class and its various operations. The `request` class is at the top, connected to four main operations: `transfer`, `register`, `physicalRemoval`, and `removeFile`. Below these, there are several other operations listed, each with a red X over it, indicating they are not implemented:

- `registerFile`
- `replicaRemoval`
- `reTransfer`
- `putAndRegister`
- `putAndRemove`
- `replicateAndRegister`
- `replicateAndRemove`

The code snippets below the diagram correspond to the operations shown:

```

exceptionCallback = None
configPath = ""

def processPool( self ):
    """ live long and prosper, my dear ProcessPool """
    # and new shiny ProcessPool instances
    # on subsequent calls
    # not self._processPool:
    minSize = max( 2, self._minProcessPool )
    maxProcessPool = ProcessPool( self._maxProcessPool, self._maxProcessCycles,
        max( 2, self._maxProcessCycles ), self._maxProcessSize, self._maxProcessTime )
    self._processPool = max( 2, self._minProcessPool )
    self._processPool.daemonize()

def registerCallBack( self, callback ):
    """ register callback function definition """
    param self: self reference
    param callback: function definition

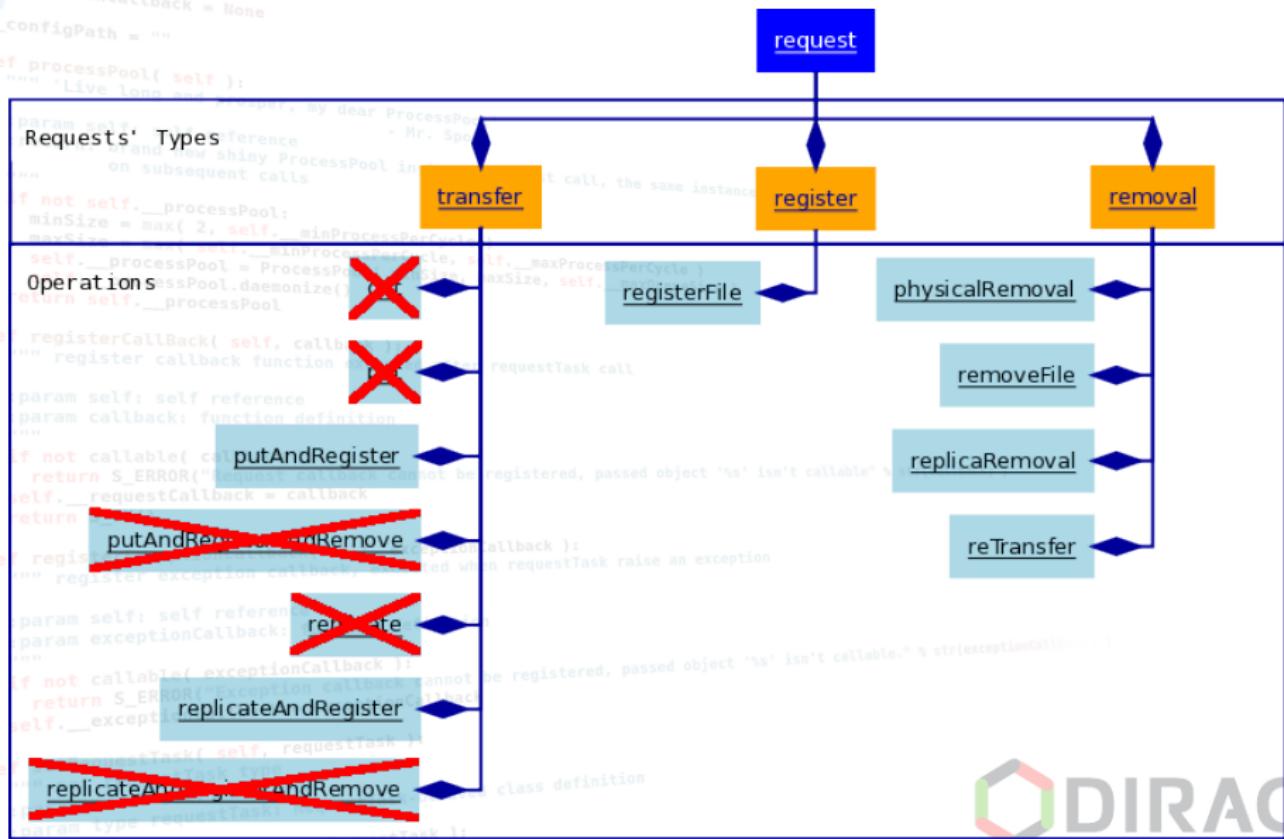
    if not callable( callback ):
        return S_ERROR("request callback cannot be registered, passed object '%s' isn't callable" % str(callback))
    self._requestCallback = callback
    return S_OK()

def registerExceptionCallBack( self, exceptionCallback ):
    """ register exception callback function definition """
    param self: self reference
    param exceptionCallback: function definition

    if not callable( exceptionCallback ):
        return S_ERROR("exception callback cannot be registered, passed object '%s' isn't callable." % str(exceptionCallback))
    self._exceptionCallback = exceptionCallback
    return S_OK()

def requestTask( self, requestTask ):
    """ class definition """
    pass

```



# PART I: base classes

```
__exceptionCallback = None
__configPath = ""
```

```
def processPool( self ):
    """ 'Live long and prosper, my dear ProcessPool'
    :param self: self reference
    :return: brand new ProcessPool instance
    """
    if not self:
        minSize = self._minSize
        maxSize = self._maxSize
        self._processPool = ProcessPool(minSize, maxSize, self)
        return self
```

## • RequestAgentBase – base class for agents

- no more threads, switch to multiprocessing (ProcessPool)
- common requests reading and dispatching to subprocesses
- callbacks from subprocesses used to monitoring/error reporting

## • RequestTask – base class for subprocesses

- one task – one request
- on-demand import & construction of DIRAC tools
- subrequests' dispatcher
- update & finalisation

## • KISS & DRY & TDD

### • documentation [click me!](#)

```
def registerExceptionCallback( self, exceptionCallback ):
    """ register exception callback to be called when requestTask raise an exception
    :param self: self reference
    :param exceptionCallback: function definition
    """
    if not callable(exceptionCallback):
        return S_ERROR("Exception callback cannot be registered, passed object '%s' isn't callable." % str(exceptionCallback))
    self.__exceptionCallback = exceptionCallback

def setRequestTask( self, requestTask ):
    """ set requestTask type
    :param self: self reference
    :param type requestTask: RequestTask-derived class definition
    """
    if not issubclass(requestTask, RequestTask):
        raise Exception("requestTask must be a subclass of RequestTask")
```



# PART I: ProcessPool and ProcessTask

- ProcessPool with two queues: waiting tasks and tasks' results
- ProcessTask execution: user def function then callback or exceptionCallback

- could be daemonized

- spawning a new python interpreter  $\forall$  queued tasks:

- no access to any global DIRAC tools, common modules, shared objects, nothing! ... except \_\_builtins\_\_
- the real parallel processing, but memory footprint!
- parent env inherited and could be modified **not disturbing the others**: can switch proxy (niiiice!) using gProxyManager when necessary

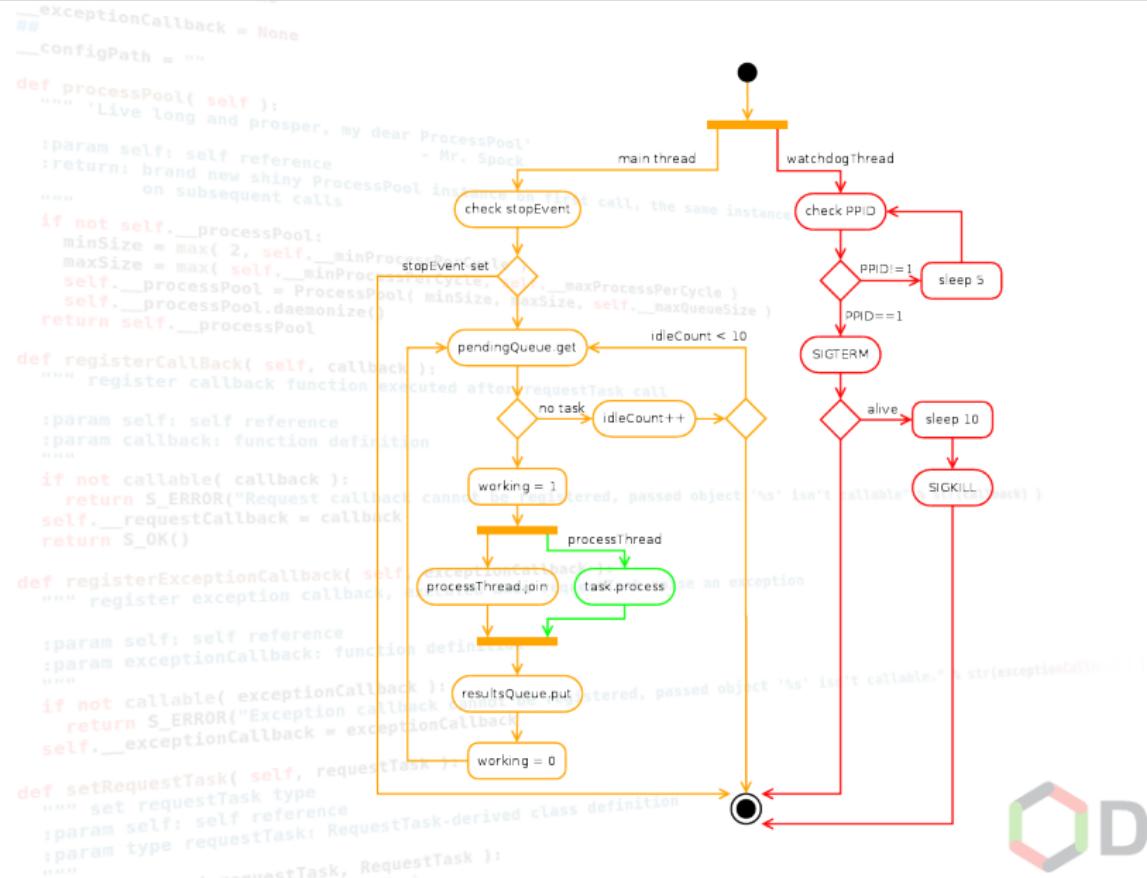
- execution of any py-callable: (**function, lambda, class** with defined \_\_call\_\_())

- dynamic subprocess management
- clean shutdown

- documentation [click me!](#)



# PART I: ProcessTask life time



# PART I: RequestTask - common modules, DIRAC tools & functions

```
def processPool( self ):
    """Live long and prosper, my dear ProcessPool"""
    :param self: self reference
    :return: None
    """Process pool class. First call, the same instance
    if not self:
        maxSize = max( 1, minProcessPerCycle )
        self = ProcessPool( maxSize )
        self._processPool = self._randomize()
    return self._processPool

    • globals objects: gLogger, gConfig, gMonitor, gProxyManager
    • common functions: S_OK, S_ERROR
    • py modules: os, sys, re, types, time
    • all loaded in RequestTask.__init__
    • ... but visible elsewhere!
```

**TRICK:** save obj ref into \_\_builtins\_\_, so py could look up its symbol elsewhere

```
## somewhere in RequestTask.__init__
from DIRAC import gMonitor
## __builtins__ type:: executed when requestTask raise an exception
## * a module, if __main__ == RequestTask
## * a dict, if RequestTask imported elsewhere
if type(__builtins__) == type(dict()):
    if __builtins__['gMonitor'] != gMonitor:
        __builtins__['gMonitor'] = gMonitor
else:
    setattr(__builtins__, 'gMonitor', gMonitor)

def setRequestTask():
    """set requestTask type
    :param self: self reference
    :param type requestTask: RequestTask-derived class definition
    """
    __builtins__.requestTask = requestTask
```



# PART I: RequestTask - specific tools

- DataLoggingClient, ReplicaManager, RequestClient, StorageFactory, TransferDB

- loaded on demand and proxied using public methods:

```
__exceptionCallback = None
__config = None
def processPool():
    """ Create my weak ProcessPool
    :param self: self reference
    :return: ProcessPool instance on first call, the same instance
    on subsequent calls
    """
    if not classmate:
        # Create placeholder for ProcessPool
        self.__processPool = ProcessPool(minSize, maxSize, self.__maxQueueSize)
        self.__processPool.daemonize()
    return self.__processPool

def __onDemandConstruction():
    """ register callback function executed after requestTask call
    """
    @classmethod
    def replicaManager( cls ):
        if not classmate:
            if not cls.__replicaManager:
                return S_ERROR("No ReplicaManager defined")
            self.__requestStatus = self.__replicaManager
            return S_OK()
        else:
            cls.__replicaManager = ReplicaManager()
        return cls.__replicaManager
    def registerExceptionCallback( self, exceptionCallback ):
        """ register exception callback for requestTask raise an exception
        """
        if not callable(exceptionCallback):
            return S_ERROR("%s is not callable." % str(exceptionCallback))
        self.__exceptionCallback = exceptionCallback
    def exampleMethod( self, lfn ):
        replicas = self.replicaManager().getReplicas( lfn )
        if not replicas['OK']:
            setRequestStatus( self, replicas )
    """ set requestStatus
    :param self:... if reference
    :param type requestTask: RequestTask-derived class definition
    """
    class( requestTask, RequestTask ):
```



## PART I: RequestTask - operation dispatcher

- a dict with operation names as keys and method to be executed as values

```
    __operationDispatcher = { 'operation1' : self.methodToRun1, ... }

    operation actions constrains:
        common signature

if not self.__processPool:
    minSize = max(1, len(self.__processes))
    maxSize = min(len(self.__processes), self.__maxProcessPerCycle)
    self.__processPool = ProcessPool(minSize, maxSize, self.__maxQueueSize)
    self.__processPool.daemonize()
return self.__processPool

def methodToRun( self, index, requestObj, subRequestAttrs,
                subRequestFiles ):
    :param self: self reference
    :param int index: execution order index for current subqueue
    :param RequestContainer requestObj: current request
    :param callback: param dict subRequestAttrs: current subrequest attributes
    :param dict subRequestFiles: current subrequest files
if not callable(callback):
    return S_ERROR("request callback cannot be registered, passed object '%s' isn't callable" % str(callback))
self.__requestCallback = callback
return S_OK()
```

- common return: S\_OK(**requestObj**) or S\_ERROR

- operation registration (in child task `__init__`)

- all funny things in `RequestTask.__call__`: switch proxy (when necessary), loop over subrequests (read & look up & execute operation actions), update request, finalize (if possible)

# PART I: installing new agents

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear Processpool'
        :param self: self reference
        :return: ProcessPool instance
    """
    if not self.__processPool:
        minSize = max( 2, self.__minProcessPerCycle )
        maxSize = max( self.__minProcessPerCycle, self.__maxProcessPerCycle )
        self.__processPool = ProcessPool( minSize, maxSize, self.__maxQueueSize )
        self.__processPool.daemonize()
    return self.__processPool

def __MinProcess = 1, MaxProcess = 4 - min and max number of subprocesses
    """ register callback function executed after RequestTask call
        :param callback: Function definition
    """
    • ProcessPoolQueueSize = 10 - waiting task queue size
    • ProcessPoolTimeout = 300, ProcessTaskTimeout = 300 - timeouts for
        pool and subprocess finalisation (in s)
    • RequestType = ... (transfer, removal, register)
    • TransferAgent/TransferTask/LogLevel=INFO,
    • RemovalAgent/RemovalTask/LogLevel=INFO,
    • RegistrationAgent/RegisterTask/LogLevel=INFO

def setRequestTask( self, requestTask ):
    """ set requestTask type
        :param self: self reference
        :param type requestTask: RequestTask-derived class definition
    """
    if not isinstance( requestTask, RequestTask ):
        raise Exception("requestTask must be registered as object '%s' is not available." % str(type(requestTask)))
    self.__exceptionCallback = requestTask
```



# PART I: special case: *transfer*

Farewell ReplicationScheduler, long live TransferAgent!

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ Live long and prosper, my dear Processpool """
    :param self: self reference
    :return: Processpool instance on first call, the same instance
             on subsequent calls
    """
    if not self.__processPool:
        minSize = self.__minProcessPerCycle
        maxSize = self.__maxProcessPerCycle
        self.__processPool = ProcessPool( minSize, maxSize, self )
    return self.__processPool

def registerCallBack( self, callback ):
    """ register callback """
    :param self: self reference
    :param callback: function definition
    """
    if not callable( callback ):
        return S_ERROR( "Request callback cannot be registered, passed object '%s' isn't callable" % str(callback) )
    self.__callBacks.append( callback )
    return S_OK()

def newStrategyHandler( self, graph ):
    """ natural mapping between SE (nodes) and FTS channels (edges) """
    :param self: self reference
    :param graph: graph definition
    """
    if not callable( graph ):
        return S_ERROR( "Request callback cannot be registered, passed object '%s' isn't callable" % str(graph) )
    self.__strategyHandler = graph
    return S_OK()

def setRequestTask( self, requestTask ):
    """ set requestTask type """
    :param self: self reference
    :param type requestTask: RequestTask-derived class definition
    """
    if not issubclass( requestTask, RequestTask ):
        raise Exception( "RequestTask must inherit from RequestTask" )

def registerExceptionCallback( self, exceptionType, requestTask, raiseOnException ):
    """ register exception callback """
    :param self: self reference
    :param exceptionType: exception type
    :param requestTask: RequestTask-derived class definition
    :param raiseOnException: boolean value
    """
    if not callable( exceptionType ):
        return S_ERROR( "Request exception type cannot be registered, passed object '%s' isn't callable" % str(exceptionType) )
    self.__exceptionCallbacks[ exceptionType ] = requestTask
    return S_OK()

def RssRWUpdatePeriod( self ):
    """ SE RW and FTS channels status refreshed every 5 minutes (CS: RssRWUpdatePeriod=300) """
    :param self: self reference
    """
    if not self.RssRWUpdatePeriod:
        return S_ERROR( "Request update period cannot be read" )
    return self.RssRWUpdatePeriod
```



## PART II: checksum check (DIRAC v6r4)

- critical for all data movement
- third party builtin mechanism in place
- FTS transfers: FTSSubmitAgent + FTSRequest
  - glite-transfer-submit --checksum-check ... SURL TURL  
**CKSMTYPE: CKSM**
  - new CS options: ChecksumTest = [True|False], ChecksumType = ... (ADLER32, MD5, SHA1, None)
    - default: None
  - outside FTS (i.e. staging, copying job's files)
    - SRM2Storage with lcg-cp5
    - CS options:
      - global: /Resources/StorageElements/ChecksumType = ... (ADLER32, CRC32, MD5, SHA1, None)
      - per SE: /Resources/StorageElements/SE/ChecksumType = ...
      - default: None (comparing only dst and src file size!)



# PART III: TransferDB clean up (DIRAC v6r5)

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear ProcessPool'
    :param self: self reference
    :return: None
    """
    if not FileToCat:
        minSize = 1
        maxSize = max( 1, self.__maxProcessPerCycle )
        self.__processPool.daemonize()
    return self.__processPool

def registerCallBack( self, callback ):
    """ register callback on executed after requestTask call
    :param self: self reference
    :param callback: callable object
    """
    if not callable( callback ):
        return S_ERROR("Callback cannot be registered, passed object '%s' isn't callable." % str(callback) )
    self.__requestTask.setCallBack( callback )
    return S_OK()

def registerExceptionCallback( self, exceptionCallback ):
    """ register exception callback
    :param self: self reference
    :param exceptionCallback: callable object
    """
    if not callable( exceptionCallback ):
        return S_ERROR("Exception callback cannot be registered, passed object '%s' isn't callable." % str(exceptionCallback) )
    self.__exceptionCallback = exceptionCallback
```

- FTS job filling in several tables in TransferDB: FTSReq, FileToFTS, FileToCat, ReplicationTree, Channel, FTSReqLogging

- do we need to keep whole FTS history? no!

- db size

- query time

- new agent: FTSCleaningAgent

- removing only successfully finished transfers

- CS options: back

- FTSCleaningAgent/GraceDaysPeriod = 180

- FTSCleaningAgent/FTSRequestsPerCycle = 50

- tested, certified but not documented yet (except inline in the code)



# PART IV: hot topics for the future

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear ProcessPool'
        - Mr. Spock
    """
    if not self._processPool:
        self._processPool = ProcessPool( self )
    return self._processPool

def registerCallPath( self, callback ):
    """ registers a call path
    """
    if not callable( callback ):
        return S_ERROR("Exception callback cannot be registered, passed object '%s' isn't callable." % str(callback) )
    self._exceptionCallback = callback
    return S_OK()

@FTS3 & GFAL2


- pure C++ and python bindings
- mysql + sqlite support
- internal auto-tuner and scheduler, session reuse
- dropping channels (easier to configure)
- backbone – GFAL2 with all available protocols (plugins)



working prototype Dec'12, release candidate Mar'13



- new CLI (fts-transfer-submit) + backward compatibility with FTS2

```

## XROOT SE

## DIRAC StorageElement POSIX-compliant

- review TransferDB schema

```
def setRequestTask( self, requestTask ):
    """ set requestTask type
    """
    if not isinstance( requestTask, RequestTask ):
        return S_ERROR("Exception callback cannot be registered, passed object '%s' isn't callable." % str(exceptionCallback) )
    self._requestTask = requestTask
```

and now for something completely different  
(not really...)



# RMS

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear ProcessPool'
        - Mr. Spock
    :param self: self reference
    :return: brand new shiny ProcessPool instance on first call, the same instance
            on subsequent calls
    """
    if not self.__processPool:
        minSize = max( 2, self.__minProcessPerCycle )
        maxSize = max( self.__minProcessPerCycle, self.__maxProcessPerCycle )
        self.__processPool = ProcessPool( minSize, maxSize, self.__maxQueueSize )
        self.__processPool.daemonize()
    return self.__processPool

def registerRequestCallback( exceptionCallback ):
    """ register callback function executed after requestTask all
    :param self: self reference
    :param callback: function definition
    """
    if not callable( callback ):
        return S_ERROR("Request callback cannot be registered, passed object '%s' isn't callable." % str(callback) )
    self.__requestCallback = callback
    return S_OK()

def registerExceptionCallback( self, exceptionCallback ):
    """ register exception callback, executed when requestTask raise an exception
    :param self: self reference
    :param exceptionCallback: function definition
    """
    if not callable( exceptionCallback ):
        return S_ERROR("Exception callback cannot be registered, passed object '%s' isn't callable." % str(exceptionCallback) )
    self.__exceptionCallback = exceptionCallback

def setRequestTask( self, requestTask ):
    """ set requestTask type
    :param self: self reference
    :param type requestTask: RequestTask-derived class definition
    """
    if not issubclass( requestTask, RequestTask ):
        raise ValueError("requestTask must be a RequestTask-derived class")
```



# PART I: current RequestClient.setRequest

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear ProcessPool' - Mr. Spock
    :param self: self reference
    :return: brand new shiny ProcessPool instance
    """
    if not self.__processPool:
        minSize = max( 2, self.__minProcessPerCycle )
        maxSize = max( self.__minProcessPerCycle, self.__maxProcessPerCycle )
        self.__processPool = ProcessPool( minSize, maxSize, self.__callback )
        self.__processPool.daemonize()
    return self.__processPool

RequestClient.setRequest

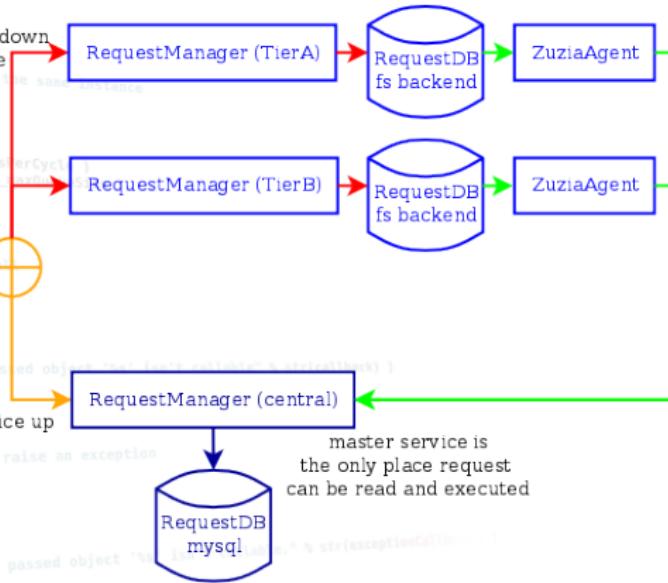
def setRequest( self, request, callback ):
    """ Set request
    :param self: self reference
    :param request: Request object
    :param callback: function definition
    """
    if not callable( callback ):
        return S_ERROR("Request callback cannot be registered, passed object is not callable")
    self.__requestCallback = callback
    return S_OK()

def registerExceptionCallback( self, exceptionCallback ):
    """ Register exception callback, executed when requests raise an exception
    :param self: self reference
    :param exceptionCallback: function definition
    """
    if not callable( exceptionCallback ):
        return S_ERROR("Exception callback cannot be registered, passed object is not callable")
    self.__exceptionCallback = exceptionCallback

def setRequestTask( self, requestTask ):
    """ Set requestTask type
    :param self: self reference
    :param type requestTask: RequestTask-derived class definition
    """
    if not issubclass( requestTask, RequestTask ):
        raise Exception("requestTask must be a subclass of RequestTask")

def saveRequest( self, request ):
    """ Save request
    :param self: self reference
    :param request: Request object
    """
    self.__requestDB.save( request )

def forwardRequest( self, request ):
    """ Forward request
    :param self: self reference
    :param request: Request object
    """
    if self.__exceptionCallback:
        self.__exceptionCallback( request )
    else:
        self.__requestDB.save( request )
```



# PART I: What's wrong?

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear ProcessPool'
    :param self: self reference
    :return: brand new shiny ProcessPool instance on first call, the same instance
             on subsequent calls
    """
    if o too much complexity, isn't it?
        minSize = max( 2, self._minProcessors )
        maxSize = min( 10, self._maxProcessors )
        self._processPool.daemonize()
    return self._processPool( self._minSize, maxSize, self._maxQueueSize )

def registerCallBack( self, callback ):
    """ register exception callback
    :param self: self reference
    :param callback: function definition
    """
    if not callable( callback ):
        o ZuziaAgent code was broken
        red, passed object '%s' isn't callable" % str(callback) )
    self._requestCallback = callback
    return S_OK()

def registerExceptionCallback( self, exceptionCallback ):
    """ register exception callback
    :param self: self reference
    :param exceptionCallback: function definition
    """
    if not callable( exceptionCallback ):
        red, passed object '%s' isn't callable." % str(exceptionCallback) )
        return S_ERROR("Exception callback cannot be registered, passed object '%s' isn't callable." % str(exceptionCallback) )
    self._exceptionCallback = exceptionCallback

def setRequestTask( self, requestTask ):
    """ set requestTask type
    :param self: self reference
    :param type requestTask: RequestTask-derived class definition
    """
    if not issubclass( requestTask, RequestTask ):
        red, passed object '%s' is not a subclass of RequestTask' % str(requestTask) )
```

The simpler is (almost always) the better.



# PART I: RequestClient.setRequest

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ Live long and prosper, my dear ProcessPool """
    param self: self reference
    :return: brand new RequestClient.setRequest
    """
    self._minSize = min( self._minProcessPerCycle,
                        self._maxProcessPerCycle )
    self._maxSize = max( self._minProcessPerCycle,
                        self._maxProcessPerCycle )
    self._processPool = ProcessPool( self._minSize, self._maxSize,
                                    self._maxQueueSize )
    self._processPool.daemonize()
    return self._processPool

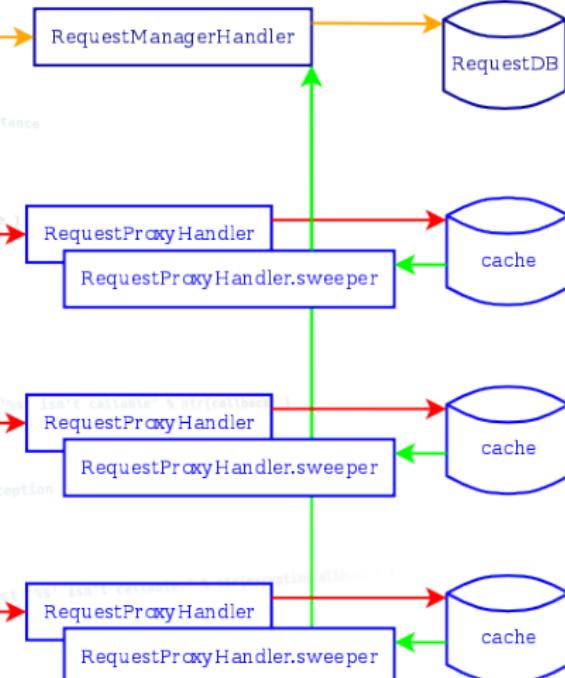
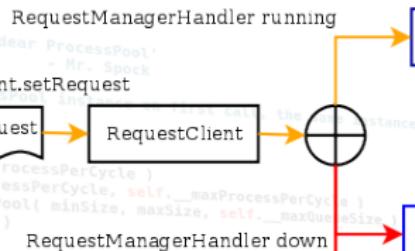
def registerCallBack( self, callback ):
    """ register callback function executed after requestTask call
    """
    param self: self reference
    param callback: function definition
    """
    if not callable( callback ):
        return S_ERROR("Request callback cannot be registered, passed object %s is not callable" % callback)
    self._requestCallback = callback
    return S_OK()

def registerExceptionCallback( self, exceptionCallback ):
    """ register exception callback, executed when requestTask raise an exception
    """
    param self: self reference
    param exceptionCallback: function definition
    """
    if not callable( exceptionCallback ):
        return S_ERROR("Exception callback cannot be registered, passed object %s is not callable" % exceptionCallback)
    self._exceptionCallback = exceptionCallback

def setRequestTask( self, requestTask ):
    """ set requestTask type
    """
    param self: self reference
    param type requestTask: RequestTask-derived class definition
    """
    if not isinstance( requestTask, RequestTask ):
        raise ValueError("requestTask must be a RequestTask instance")
    self._requestTask = requestTask
```

Legend

- > set request
- > save request
- > forward request



# PART I: RequestProxyHandler DIRAC v6r5

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear ProcessPool'
    :param self: self reference
    :return: brand new shiny ProcessPool instance on first call
    """
    if not self.__processPool:
        minSize = max( 2, self._minSize )
        maxSize = self._maxSize
        self.__processPool = ProcessPool( minSize, maxSize, self, maxQueueSize )
    return self.__processPool

def registerCallBack( self, callback ):
    """ request forwarding (sweeping) in a background thread
    :param self: self reference
    :param callback: function definition
    """
    if not callable( callback ):
        return S_ERROR("Passed object '%s' isn't callable." % str(callback) )
    self.__requestForwardingCallback = callback
    return S_OK()

def registerIX( self, exceptionCallback ):
    """ register exception callback, executed when requestTask raise an exception
    :param self: self reference
    :param exceptionCallback: function definition
    """
    if not callable( exceptionCallback ):
        return S_ERROR("Exception callback cannot be registered, passed object '%s' isn't callable." % str(exceptionCallback) )
    self.__exceptionCallback = exceptionCallback

def setRequestTask( self, requestTask ):
    """ set requestTask type
    :param self: self reference
    :param type requestTask: RequestTask-derived class definition
    """
    if not issubclass( requestTask, RequestTask ):
        return S_ERROR("Passed object '%s' is not a subclass of RequestTask" % str(requestTask))



- the only one "central" RequestManagerHandler
- + several RequestProxyHandlers installed at Tier1 boxes
- caching: XML serialisation + dump to file
- request forwarding (sweeping) in a background thread
- obsolete:
- RequestManager URI propagation
- ZuziaAgent
- RequestDBFile
- documentation here

```



## PART II: request's refactoring

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear ProcessPool'
    :param self: self reference
    :return: brand new shiny ProcessPool instance on first call, the same instance
            on subsequent calls
    """
    if not self.__processPool:
        minSize = max( 1, self.__minProcessPerCpu )
        self.__processPool = ProcessPool( minSize, maxSize = self.__maxProcessPerCpu )
```

Do we really need to do that?

- in mysql: 3 tables (Requests, SubRequests, Files)
- in py: only one class RequestContainer

```
def registerCall( self, callback ):
    """ register callback function executed after requestTask call
    :param self: self reference
    :param callback: callable object
    """
    if not callable( callback ):
        return S_ERROR( "callback must be a callable object (%s isn't callable)" % str(callback) )
    self.__requestCallback = callback
    return S_OK()
```

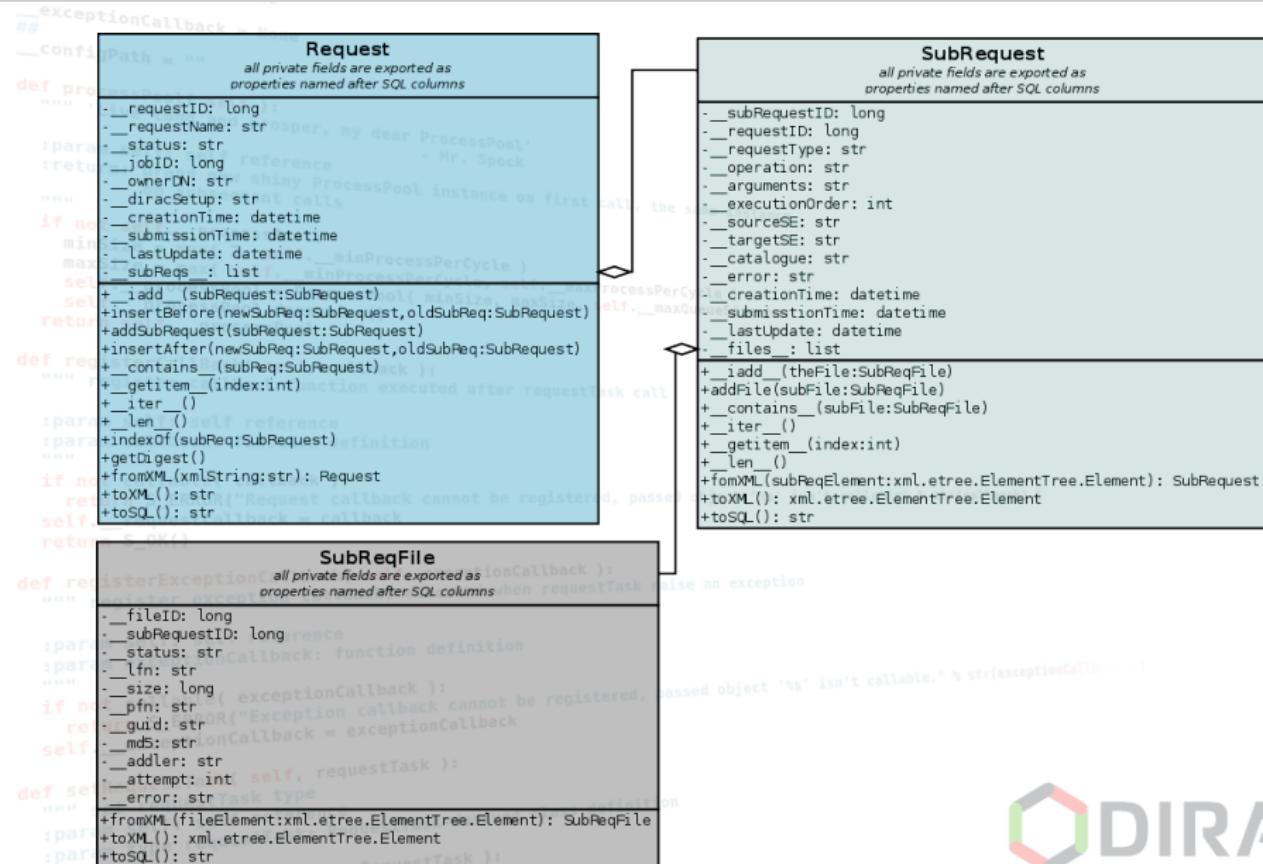
- too **heavy, obscure** and **unclear API**
  - ...70 functions in total!
- too **complicated** to maintain
  - no **validation** of attributes at all
- no internal **state machine**, processing spread over different agents and modules
  - irrelevant or missing indexes in SQL definition
    - ...see Zoltan Mathe PR993

```
def registerException( self, exceptionCallback ):
    """ register exception callback, executed when requestTask raise an exception
    :param self: self reference
    :param exceptionCallback: callable object
    """
    if not callable( exceptionCallback ):
        return S_ERROR( "exceptionCallback must be a callable object (%s isn't callable)" % str(exceptionCallback) )
    self.__exceptionCallback = exceptionCallback
```

```
def setRequestTask( self, requestTask ):
    """ set requestTask type
    :param self: self reference
    :param type requestTask: RequestTask-derived class definition
    """
    if not issubclass( requestTask, RequestTask ):
```



## PART II: new classes proposal



## PART II: new classes proposal

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear ProcessPool'
        :param self: self reference
        :return: None
    if not self._processPool:
        self._processPool = ProcessPool( self )
        self._processPool.daemonize()
    return self._processPool
```

- **bag object** - no clever operations over there!!!

- **ORM-like** – one py class per one sql table

- all sql fields exported as **properties**

- **lightweight** API:

```
def registerCallback( self, callback ):
    """ register a callable object as a request callback
        :param self: self reference
        :param callback: function definition
    if not callable( callback ):
        return S_ERROR("Request callback cannot be registered, passed object '%s' isn't callable." % str(callback) )
```

- **two step validation:**

```
sel
    sel._registerCallback( callback )
    return S_OK()
```

- **low-level** in object itself (i.e. Request.RequestID is a long or int?)

```
def registerExceptionCallback( self, exceptionCallback ):
    """ register an exception callback
        :param self: self reference
        :param exceptionCallback: function definition
    if not callable( exceptionCallback ):
        return S_ERROR("Exception callback cannot be registered, passed object '%s' isn't callable." % str(exceptionCallback) )
```

- **high-level** in RequestClient using RequestValidator (i.e. transfer

```
SubRequest has got some files defined? SubRequest's Operation matches
RequestType? )
```

```
self._exceptionCallback = exceptionCallback

def setRequestTask( self, requestTask ):
    """ set requestTask type
        :param self: self reference
        :param type requestTask: RequestTask-derived class definition
    """
```

- subclass( requestTask, RequestTask )



## PART II: lightweight API with properties

```
exceptionCallback = None
__configPath = ""

def process(self):
    """Process the queue, my dear ProcessPool"""
    - Mr. Spock
    if not self._queue:
        raise RuntimeError("No ProcessPool instance on first call, the
                           queue cannot be registered, passed to
                           ProcessPool constructor or to self._maxQueueSize")
    return self._queue.get()

class C(object):
    """A class with a property, my dear ProcessPool"""
    - Mr. Spock
    def __init__(self):
        self._x = None
    def x():
        """I'm the 'x' property."""
        doc = self.__doc__
        self._x = 10
        return self._x
    def fset(self, value):
        """ setter and low-level
            validation """
        if type(value) != int:
            raise TypeError("expecting int
                            , got %s" % type(value))
        self._x = value
    def fget(self):
        """ getter """
        return self._x
    def fdel(self):
        """ deleter """
        del self._x
    if not self._x:
        return locals()
    x = property(x, fset, fdel)
    def setRequestTask(self, requestTask):
        """ set requestTask type """
        self._requestTask = requestTask
    def __init__(self, requestTask):
        self._requestTask = requestTask
        self._queue = Queue(maxsize=10)
```

```
[cibak@localhost:uml/snippet]> python
Python 2.7.3 (default, Jul 24 2012,
10:05:38)
[GCC 4.7.0 20120507 (Red Hat 4.7.0-5)]
Type "help", "copyright", "credits" or
"license" for more information.
>>> c = C()
>>> C.x.__doc__ ## get the doc
"I'm the 'x' property."
>>> c.x = 10 ## setting x to 10
>>> c.x ## what's the value of x?
10
>>> c.x = "will throw TypeError"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "<stdin>", line 9, in fset
TypeError: expecting integer, got <
              type 'str'>
```



## PART II: new API by examples



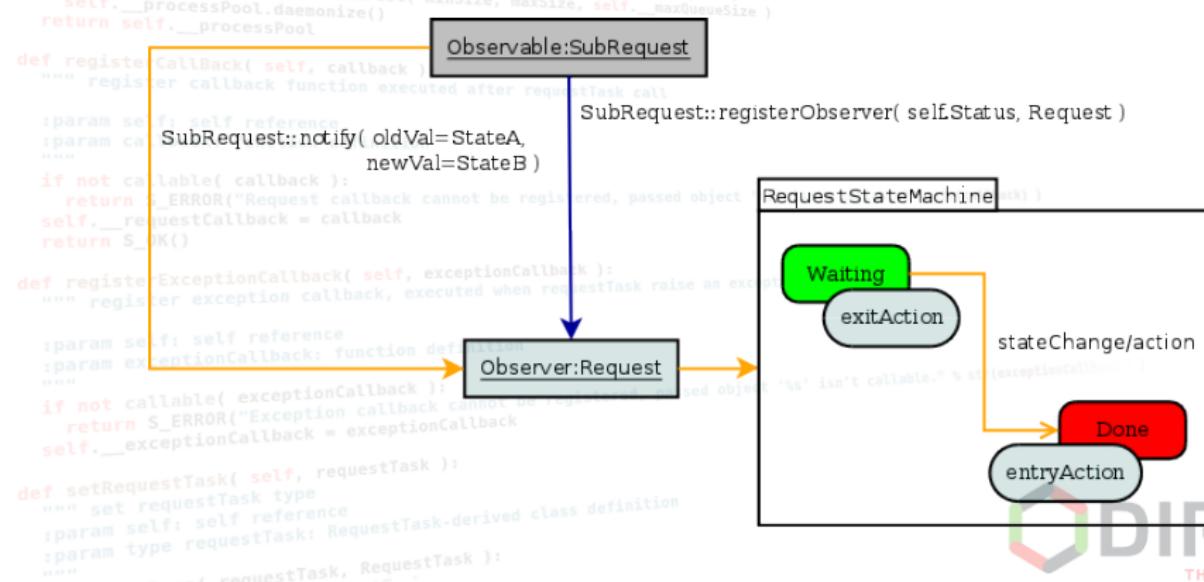
## PART II: state machine

- FMS using **directed graph** (again!):

- *node → state* (callable properties: `entryAction`, `exitAction`)
- *edge → state transition* (callable property: `action`)

- **Observer** pattern as a backbone of state machine:

- `Observable.registerObserver()`, `Observable.notify()`



# PART II: state machines

```

__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear ProcessPool'
    :param self: self reference
    :return: brand new shiny ProcessPool instance on first call, same instance
             on subsequent calls
    """
    Request state machine
    self._processPool = ProcessPool( minSize, maxSize, self, __maxProcessPerCycle )
    self._processPool.daemonize()
    return self.

def registerCallback( self, callback ):
    """ register callback Function executed after requestTask call
    :param self: self reference
    :param callback: function definition
    """
    if not callable( callback ):
        return S_ERROR("Request callback cannot be registered. %s isn't callable." % str(callback))
    self._requestCallback = callback
    return S_OK()

def registerExceptionCallback( self, exceptionCallback ):
    """ register exception callback, triggered when requestTask fails
    :param self: self reference
    :param exceptionCallback: function definition
    """
    if not callable( exceptionCallback ):
        return S_ERROR("Exception callback cannot be registered. %s isn't callable." % str(exceptionCallback))
    self._exceptionCallback = exceptionCallback

def setRequestTask( self, requestTask ):
    """ set requestTask type
    :param self: self reference
    :param type requestTask: RequestTask-derived class definition
    """
    if not issubclass( requestTask, RequestTask ):
        raise Exception("requestTask must inherit from RequestTask")

__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear ProcessPool'
    :param self: self reference
    :return: brand new shiny ProcessPool instance on first call, same instance
             on subsequent calls
    """
    - Mr. Spock
    SubRequest state machine
    self._processPool = ProcessPool( minSize, maxSize, self, __maxProcessPerCycle )
    self._processPool.daemonize()
    return self.

def registerCallback( self, callback ):
    """ register callback Function executed after requestTask call
    :param self: self reference
    :param callback: function definition
    """
    if not callable( callback ):
        return S_ERROR("Request callback cannot be registered. %s isn't callable." % str(callback))
    self._requestCallback = callback
    return S_OK()

def registerExceptionCallback( self, exceptionCallback ):
    """ register exception callback, triggered when requestTask fails
    :param self: self reference
    :param exceptionCallback: function definition
    """
    if not callable( exceptionCallback ):
        return S_ERROR("Exception callback cannot be registered. %s isn't callable." % str(exceptionCallback))
    self._exceptionCallback = exceptionCallback

def setRequestTask( self, requestTask ):
    """ set requestTask type
    :param self: self reference
    :param type requestTask: RequestTask-derived class definition
    """
    if not issubclass( requestTask, RequestTask ):
        raise Exception("requestTask must inherit from RequestTask")

```

**Request state machine**

- Initial state: Waiting
- Transitions:
  - onCreate: to Waiting
  - onUpdate (not Done): to Waiting
  - onUpdate (all Done): to Done/Failed
  - onProcessed: to Done/Failed
  - onReplicated: to Done/Failed
- Final state: Done/Failed

**SubRequest state machine**

- Initial state: Queued
- Transitions:
  - onPrevSubReqDone: to Waiting
  - onInsertBefore: to Waiting
  - onSelect: to Assigned
  - onProcessed: to Done/Failed
  - onReplicated: to Done/Failed
- Final state: Done/Failed

**SubReqFile state machine**

- Initial state: Waiting
- Transitions:
  - onSchedule: to Waiting
  - onProcessed: to Scheduled
  - onReplicated: to Done/Failed
- Final state: Done/Failed

*only one subRequest with 'Waiting' status at a time*

## PART II: a glimpse into the future

```
__exceptionCallback = None
__configPath = ""

def processPool( self ):
    """ 'Live long and prosper, my dear ProcessPool'
        :param self: self reference
        :return: brand new shiny ProcessPool instance on first call, the same instance
    """
    if not self._processPool:
        minSize = self.__processPoolMinSize
        maxSize = self.__processPoolMaxSize
        self._processPool = InProcessPool( self, minSize, maxSize )
        self._processPool.start()
    return self

def registerCallBack( self, callback ):
    """ ... and what's not?
        :param self: self reference
        :param callback: function definition
    """
    if not callable( callback ):
        return S_ERROR("Exception callback cannot be registered, passed object '%s' isn't callable." % str(callback) )
    self._exceptionCallback = exceptionCallback
    return S_OK()
```

- what's ready?

- Request, SubRequest, SubReqFile
- Graph classes, Observer, StateMachine
- RequestValidator

- ... and what's not?

- changes in the existing code: RequestClient, RequestManager, RequestDB, various agents...
- official doc + several tests

- time scale: release candidate by the end of this year

- questions? comments? [Krzysztof.Ciba@gmail.com](mailto:Krzysztof.Ciba@gmail.com)

