ATL-DAQ-2001-002
06/03/2001

PC−based Event Filter Supervisor: Design and Implementation

Z. Qian, C. Bee, E. Fede, C. Meessen, F. Touchard

CPPM Marseille

Abstract

This document presents a PC−Based Event Filter Supervisor design and implementation, based on Java and Java Mobile Agent technology. The supervisor has been in use for two years and has been tested in various configurations and on different platforms. Full integration with the ATLAS DAQ/EF prototype has been performed and is described in detail

Keywords :	ATLAS, Event filter,Supervisor, Monitoring, Mobile Agent, Java
Document version :	1.3
Reference :	http://atddoc.cern.ch/Atlas/EventFilter/documents/spv/spv.html

Table of Contents

1 Introduction

The Event Filter (EF) is the last element in the DAQ chain before events are sent to permanent storage. The design of this element is described in a paper presented at RT99 conference [1]. The hardware configuration and the implementation of event data flow are shown in Figure 1 and Figure 2.

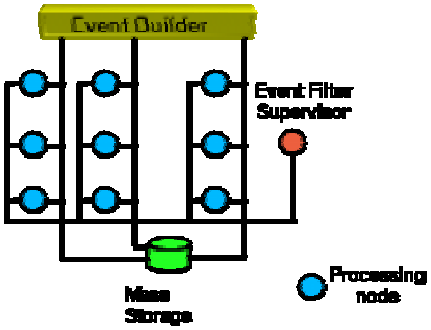


Figure 1: Hardware configuration

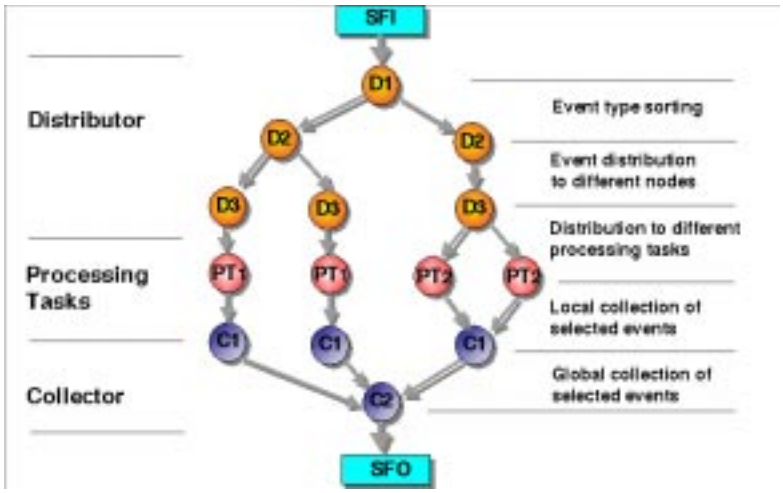


Figure 2: Event data flow implementation

This document describes the design and implementation of the supervision of a PC−based EF.

The Supervisor must fulfil the following requirements :

- Operation of the dataflow and the supervision must be totally independent, i.e. a crash of the supervisor should not affect the dataflow, and conversely.
- Platform, operating system independence, to cope with Event Filter's heterogeneity
- Event data flow independence for new technology and future development
- Scalability, which means that the Supervisor must be capable of handling different sized configurations, from one PC to several hundred PCs

- High degree of flexibility, to adapt to different architectures and implementation.
- Easy maintenance
- Robustness
- Remote control and monitoring
- Provide archiving means via access to a database.

The main functionality of EF Supervision is described in [2].

2 Technology Choices

The Java Mobile Agent technology has been chosen to implement the above requirements. The Supervisor is built on top of the ObjectSpace Voyager Core Technology, a Java Object Request Broker. All control functions and monitoring functions are performed by different types of Java Agents.

We summarise here some interesting features of Java Mobile Agents and of Voyager. A complete evaluation report can be found in [3].

2.1 What a Java Mobile Agent system looks like

The Java Virtual Machine and Java's class loading model, coupled with several of the Java features, among which serialisation, remote method invocation, multithreading, and reflection are the most pertinent, have made building mobile agent systems a fairly simple task.

Java Mobile Agent systems have a number of key characteristics :

- All Java Mobile Agent systems provide an agent server, which is a contact point on a given machine (see Figure 3). Those server objects act as warehouses, or workplaces into which agents move, and in which agents act. A server provides a means of hosting and managing its own agent in an environment that is secure from malicious agents.
- Agents can migrate from server to server, carrying their state with them. After moving into a server, an agent becomes a local user, and it can do everything that a local user can do, e.g. get system resource information (process, disk, memory, CPU, network...), create/ delete processes, make local communication with process ...
- Agents can load their code from a variety of sources. In general, since all the agent systems use a specialised version of the Java classloader, they can load Java class files from the local file system, the Web, and ftp servers.
- They are 100% pure Java. This means that they should run on any computer with a compatible Java runtime

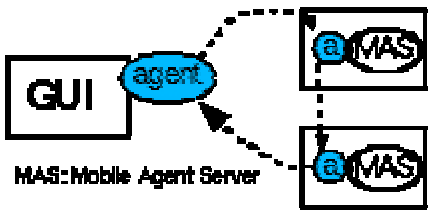


Figure 3: Mobile agent principle

2.2 What is Voyager and why Voyager was chosen

“ObjectSpace Voyager Core Technology” [4] is an advanced, 100% Java Object Request Broker (ORB), based on the Java language object model. Voyager 3.0 contains the following features : remote−enabling a class; remote create instance of any class and obtain his proxy; dynamic class loading; , remote message (one−way, sync, future); multicast; remote exception handling; distributed garbage collection; dynamic aggregation; support for IDL, IIOP and RMI; mobility − move any serializable object at runtime from one virtual machine to another; support autonomous mobile agents; activation of objects persisted in any kind of database; applets and servlets; universal naming service; publish−subscribe; thread pooling; enhanced security manager; possible for installing custom socket such as SSL; J2EE JMS compliance.

We decided to evaluate this framework for the following reasons:

- Three things in one, Voyager supports three types of communication : point−to−point, client/server, agent−based
- Easy of use
- Good performance
- Very good scalability
- Small (Voyager core classes are only 763k)
- Many facilities correspond to the requirement of the supervisor : space scalable group communication, publish/subscribe, event & listener, object persistence, integration with CORBA.
- possible use as internal message server

3 Supervisor design

3.1 Design scheme

The design scheme is shown in Figure 4. There are 3 levels of monitoring : PC tool for system level supervision, Mobile Agent for event data flow control and monitoring, offline analyse tools for monitoring data archived in a database

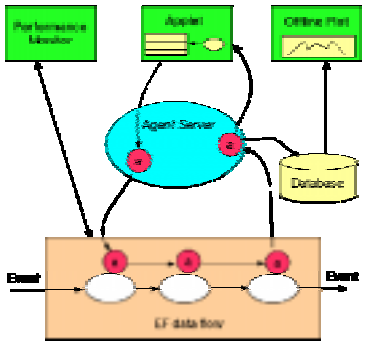


Figure 4 : Design scheme

3.2 User Interface

3.2.1 Online Graphical User Interface

A Graphical User Interface (GUI) was implemented in such a way that it can run in application mode (with a direct access to the agent server) or applet mode (via the Web); the same GUI can be used for monitoring and control and can run either in standalone mode or over the Web. Several copies of the interface can run at the same time, but only one is allowed to perform the control task. The agent server notifies all running interfaces when the EF status changes. Figure 5 illustrates how commands and status requests can be sent from the Control Interface to remote nodes by mobile agents and how status data is collected and returned to any monitoring interface.

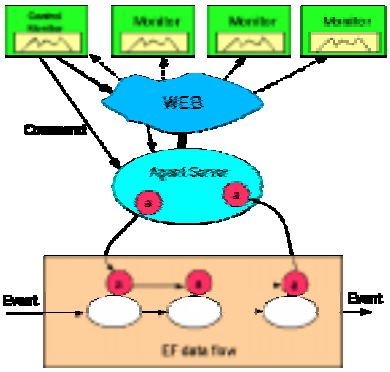


Figure 5 : Control flow

The GUI has multiple functionality, including : online histogramming and plotting; different viewers to display the entire EF hardware configuration and event data flow structure; control of usage of archiving database; possibility to switch on/off the database and change the rate of monitoring data collection; possibility of add/delete process at runtime; modification of some dataflow parameters; display an agent's travelling status via an agent itinerary window.

A working GUI is shown in Figure 6 which contains 3 basic windows : the Main Window (Supervisor), the Activity Viewer, and the Config Tree. The detailed functionality of each window is described below.

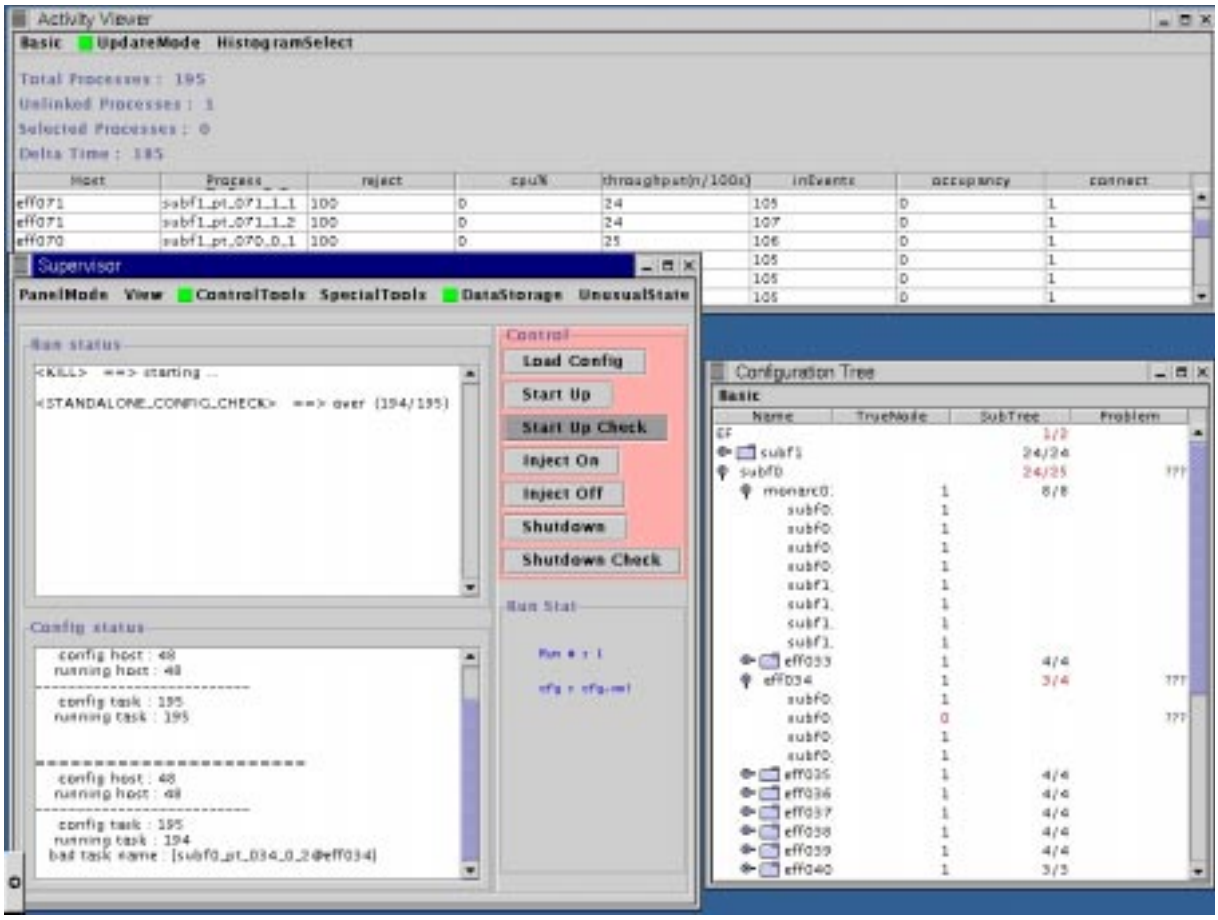


Figure 6 : Supervisor GUI

3.2.1.1 Main Window

The **Main Window** (Figure 7) consists of three parts:

- **Control panel** (top right): this panel allows to send commands to the Farm (partition) in standalone mode
- **Run status panel** (top left): the status of execution of a command is displayed in this window. As an example, when "Start Up Check" button is pressed, an agent goes into EF, checks all components and shows the result in this panel:
  "<STANDALONE_CONFIG_CHECK> ==> over (194/195)"
  This means that there are 195 processes in the config file, and 194 are presently running
- **Config status panel** (bottom left): this panel displays the summary of config status, including the number of hosts and tasks in the configuration, the number of running hosts and tasks, bad task name if any. In Figure 7 the panel shows one bad task named subf0_pt_034_0_2 at host eff034.

Menu bar of the window provides extra functionality, e.g.

- **PanelMode** allows to select the mode of the main window : Standalone, Combine, Combine be, Combine df. The differences between those modes are described in section 6.4.2.
- From **ControlTools**, the user can pop up secondary windows : Config file selector, Dataflow param viewer, Partial reset panel ... When the user activates "Cfg autotest", the Supervisor sends ping agents at an adjustable time intervals. A superclean command kills all

components of EF dataflow.

- **SpecialTools** used for supervisor functionality control, includes : internal message dump, native command panel, activate/deactivate supervisor debugging, reload supervisor init parameter, reconnect master ...
- **DataStorage** allows to connect/disconnect with persistent data storage
- **BeTools** allows to make test with Online Software, include : send status, send EF info, change partition, set shutdown option ... see section 6.4.2 for detail.
-

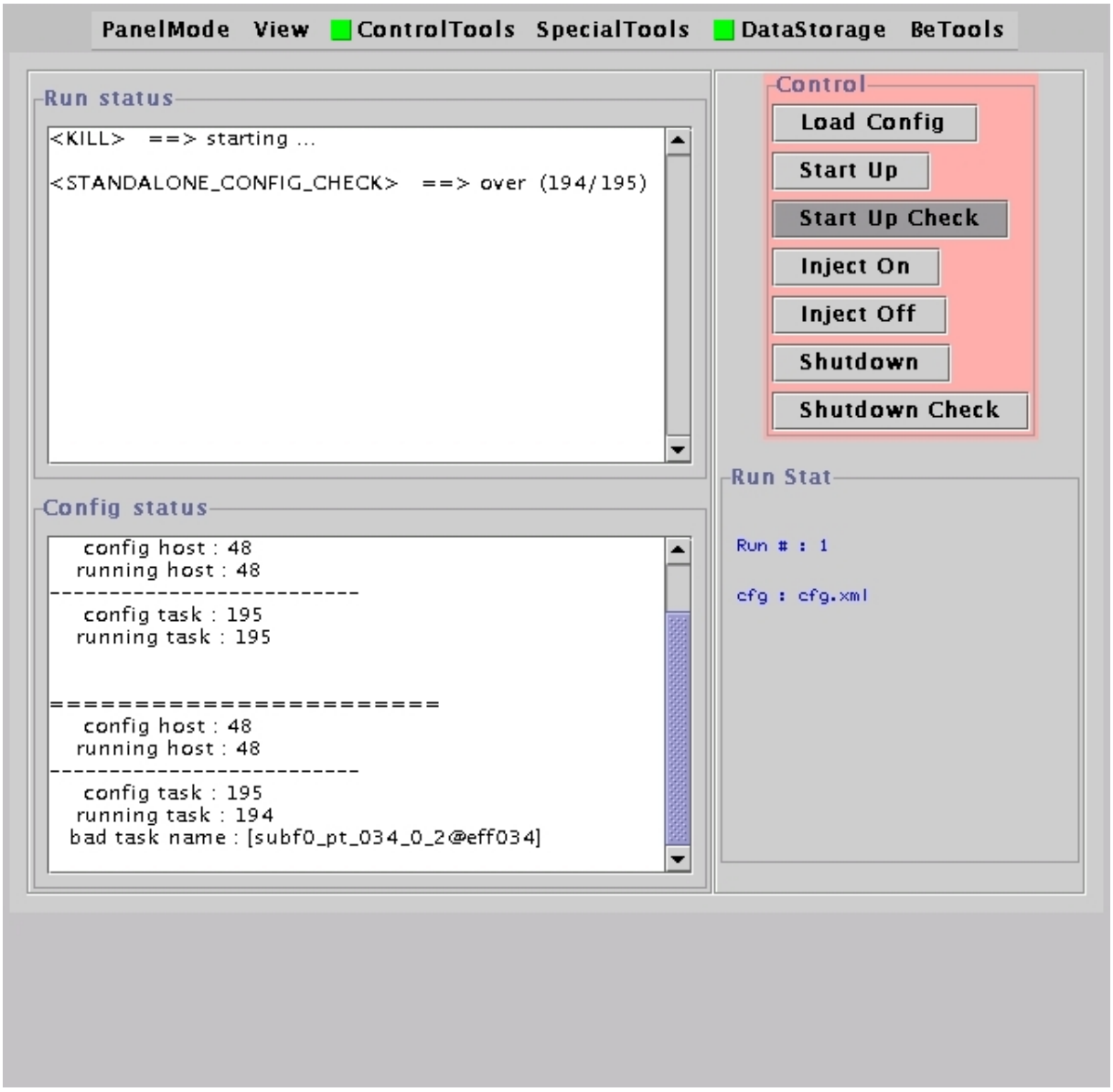


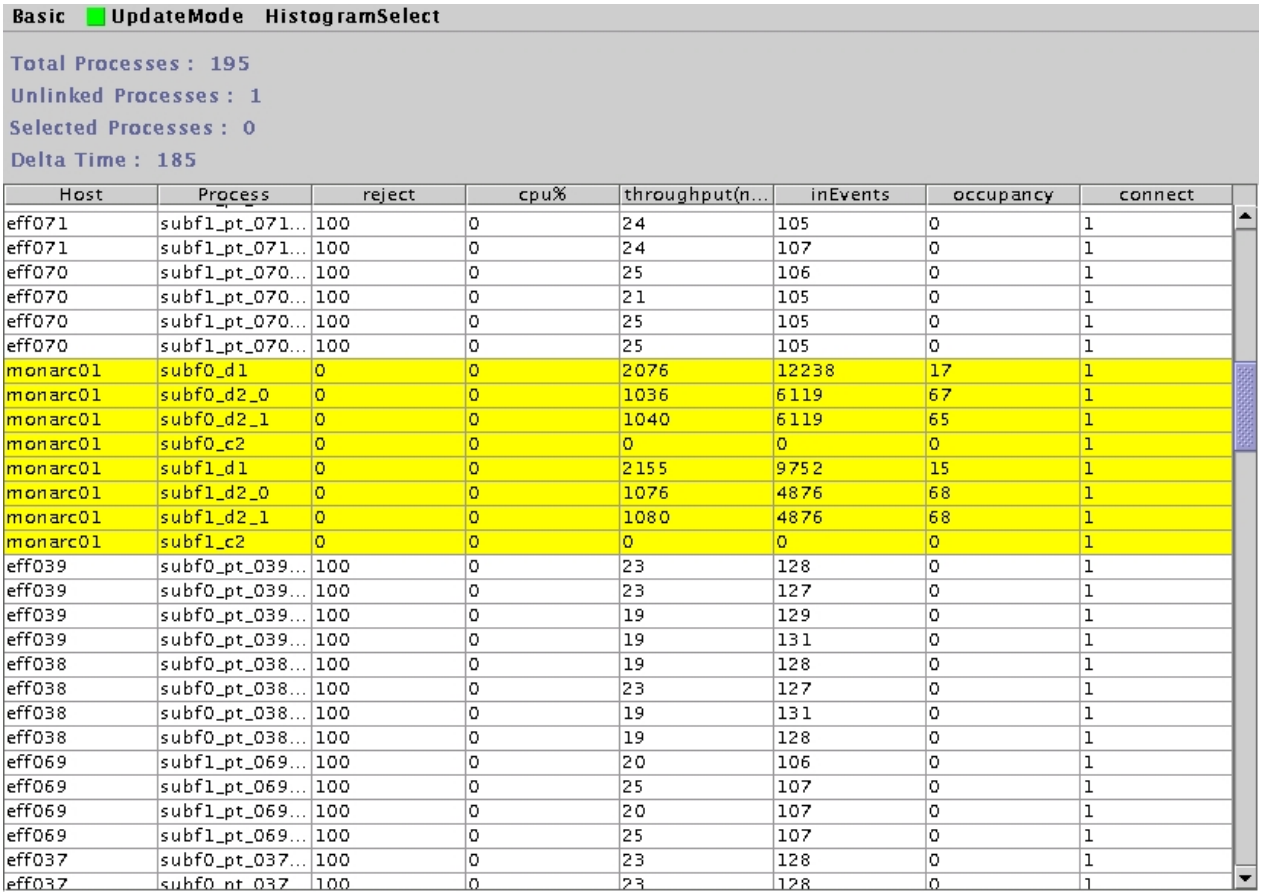
Figure 7 : Main Window

3.2.1.2 Activity Viewer

The Activity Viewer (Figure 8) displays monitoring data given by each dataflow component in

the system. Typical monitoring data is the total number of events having passed through the component, input/output FIFO occupancy, etc. ... Data is displayed in form of a swing table. The following facilities are available from the menu bar :

- Get all value (manual operation)
- Get all value (automatic operation with an adjustable time intervals)
- Get value for selected components (manual/automatic operation)
- Display histogram built from the current values.



Host	Process	reject	cpu%	throughput(n...	inEvents	occupancy	connect
eff071	subf1_pt_071...	100	0	24	105	0	1
eff071	subf1_pt_071...	100	0	24	107	0	1
eff070	subf1_pt_070...	100	0	25	106	0	1
eff070	subf1_pt_070...	100	0	21	105	0	1
eff070	subf1_pt_070...	100	0	25	105	0	1
eff070	subf1_pt_070...	100	0	25	105	0	1
monarc01	subf0_d1	0	0	2076	12238	17	1
monarc01	subf0_d2_0	0	0	1036	6119	67	1
monarc01	subf0_d2_1	0	0	1040	6119	65	1
monarc01	subf0_c2	0	0	0	0	0	1
monarc01	subf1_d1	0	0	2155	9752	15	1
monarc01	subf1_d2_0	0	0	1076	4876	68	1
monarc01	subf1_d2_1	0	0	1080	4876	68	1
monarc01	subf1_c2	0	0	0	0	0	1
eff039	subf0_pt_039...	100	0	23	128	0	1
eff039	subf0_pt_039...	100	0	23	127	0	1
eff039	subf0_pt_039...	100	0	19	129	0	1
eff039	subf0_pt_039...	100	0	19	131	0	1
eff038	subf0_pt_038...	100	0	19	128	0	1
eff038	subf0_pt_038...	100	0	23	127	0	1
eff038	subf0_pt_038...	100	0	19	131	0	1
eff038	subf0_pt_038...	100	0	19	128	0	1
eff069	subf1_pt_069...	100	0	20	106	0	1
eff069	subf1_pt_069...	100	0	25	107	0	1
eff069	subf1_pt_069...	100	0	20	107	0	1
eff069	subf1_pt_069...	100	0	25	107	0	1
eff037	subf0_pt_037...	100	0	23	128	0	1
eff037	subf0_pt_037...	100	0	23	128	0	1

Figure 8 : Activity viewer

3.2.1.3 Config Tree

The Config Tree (Figure 9) gives a global view of the farm configuration based on the config file. Configuration description is described more precisely below.

This window displays the tree structure of configuration (sub−farm → host → process) and their status. When the supervisor detects a dead process, the appearance of the corresponding line (the line subf0_pt_034_0_2 in fig 9) is modified, several parts of the line are changed in value and in colour. To highlight the problem, the parent lines of the dead process (eff034, subf0, EFroot) is also modified. the column “SubTree” shows that eff034 machine has 4 processes in config and 3

are still running, subf0 has 25 hosts in config and 24 have normal status, EF has 2 sub−farms in config and only one is working perfectly.

To each line is associated a pop−up menu which allows the user to start/stop any part of farm during run time.

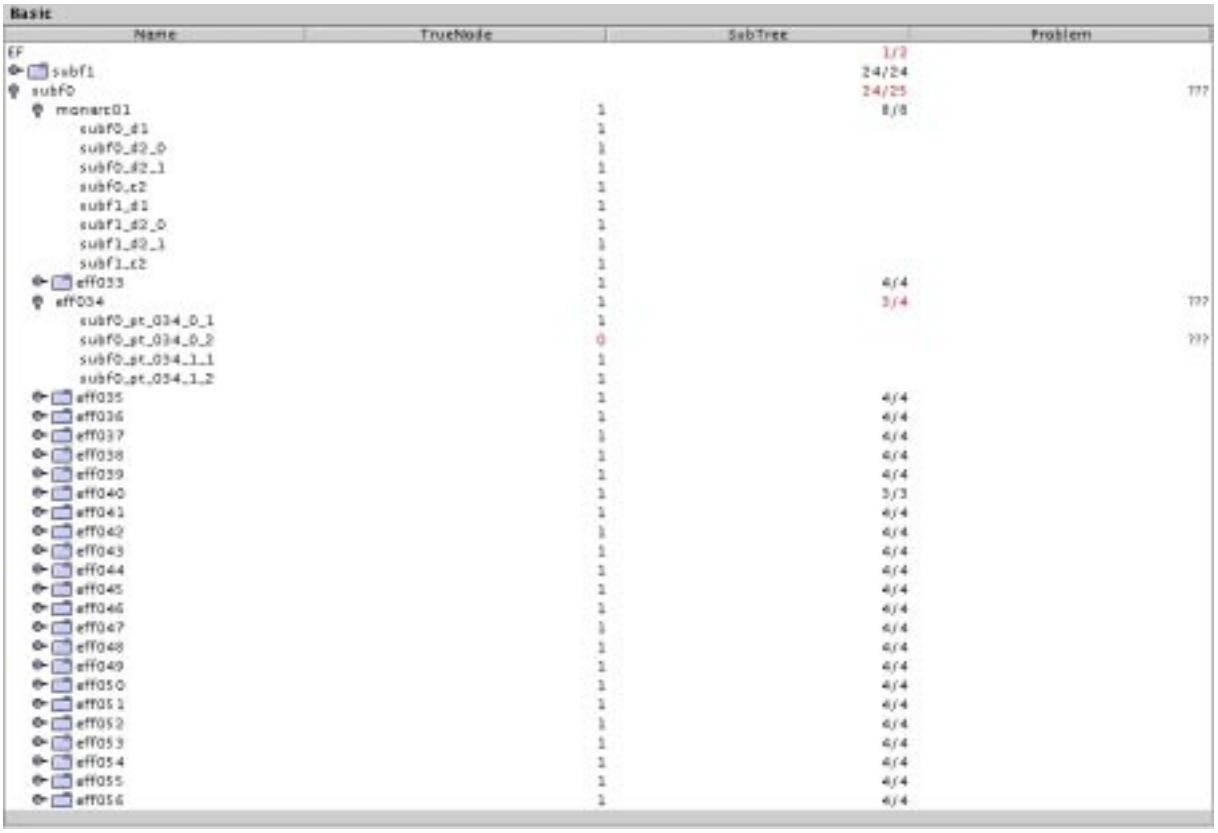


Figure 9:Configuration tree

3.2.2 GUI for configuration description

The configuration of the Farm is described in a XML file following the DTD (Document Type Definition) file displayed in Appendix 1

This XML file describes all the details of the farm, giving for every sub−farm its name, the hosts for the SFI and SFO components, the executable to be run by these components and the list of nodes involved for processing. For every node, the paths to the directories containing the binary files and the required information for the EF naming service are given. Finally, every component running on the node is described. The parameters of the different components are given in the “command line” given by the “Exec” attribute.

An user interface written in Tcl/Tk conveniently allows the generation of the XML file. It has been separated in two different processes. The first one (Figure 10 :)allows to modify all the basic parameters of the farm. It is intended to be an "expert" interface. The second one (Figure 11:), more dedicated to the end users, allows to modify only parameters relevant of the application, such as the full path of the processing tasks to be executed for the filtering operation.

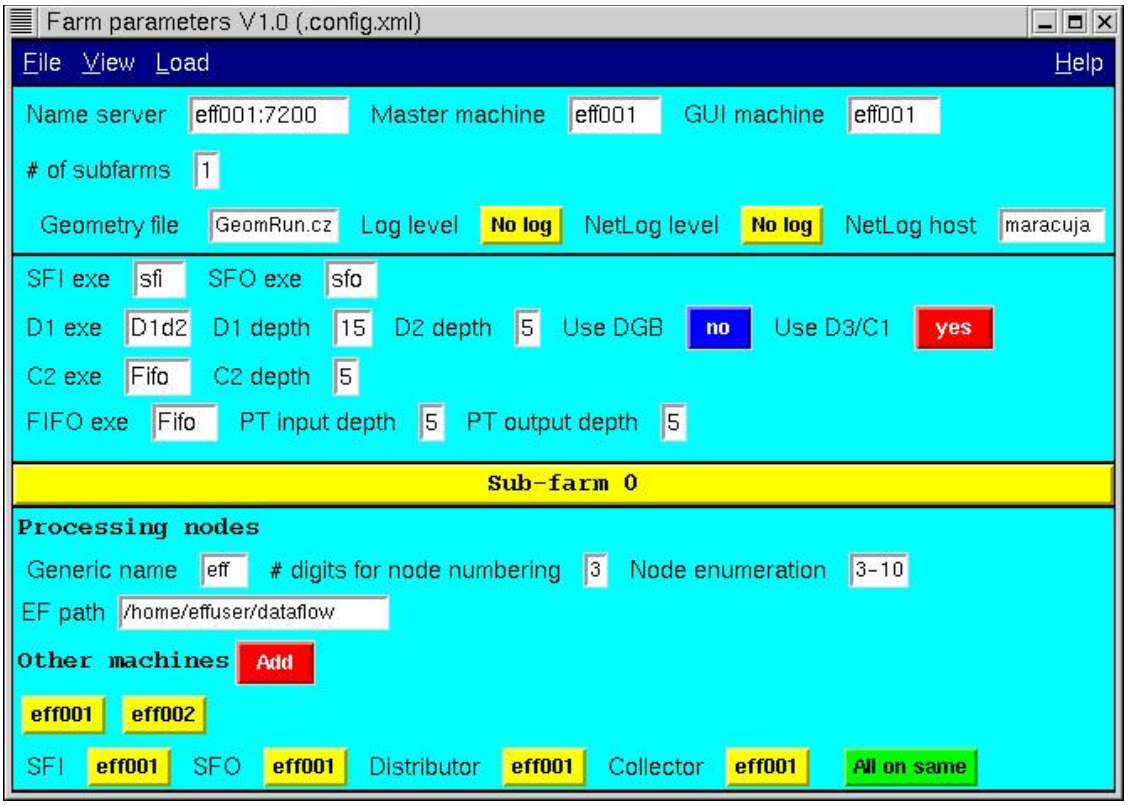


Figure 10 : User interface for basic farm parameters

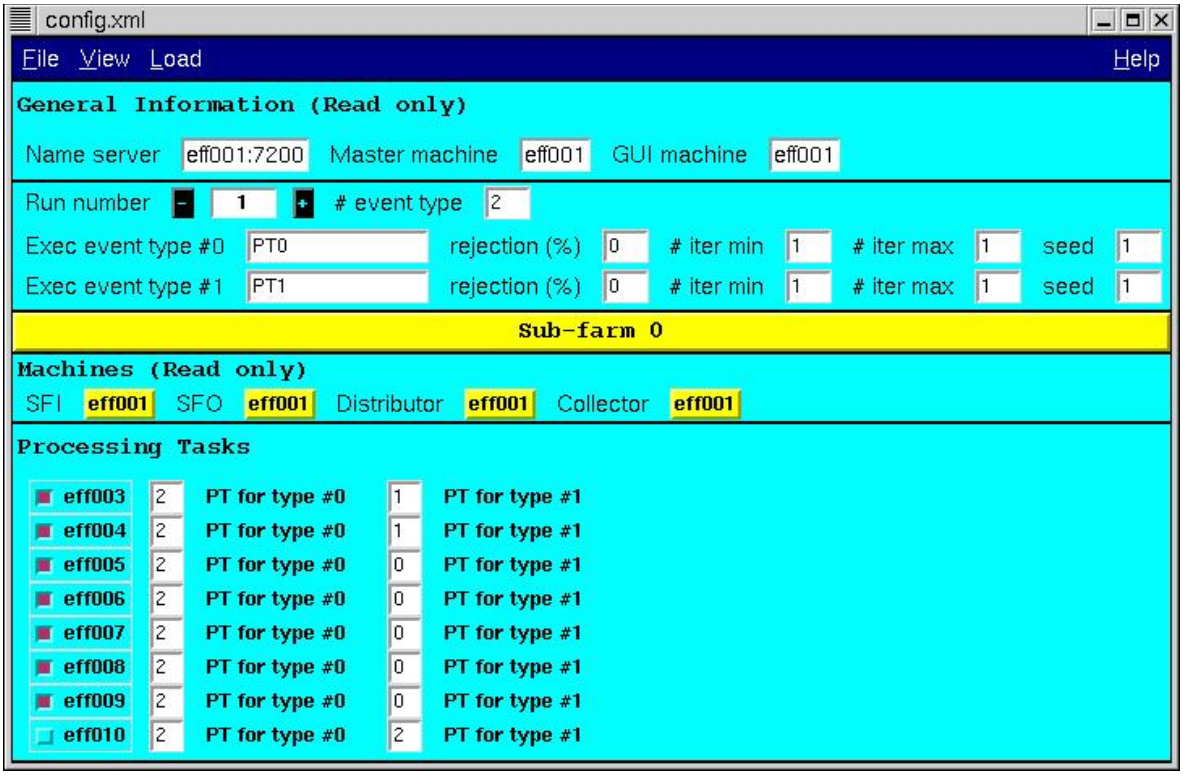


Figure 11: User interface for farm applications

3.2.3 Offline Analyse Tool

An histogram package and a plot facility have been developed in parallel for online plotting and

offline analysis. The histogram package contains 1D and 2D histogram creation and filling functions. The plot facility gives the possibility to display histograms in several manners : histogram with/ without statistics, runtime plot, slide. The data source can be an XML file, a flat file , a histogram saving file or a database (see Figure 12).

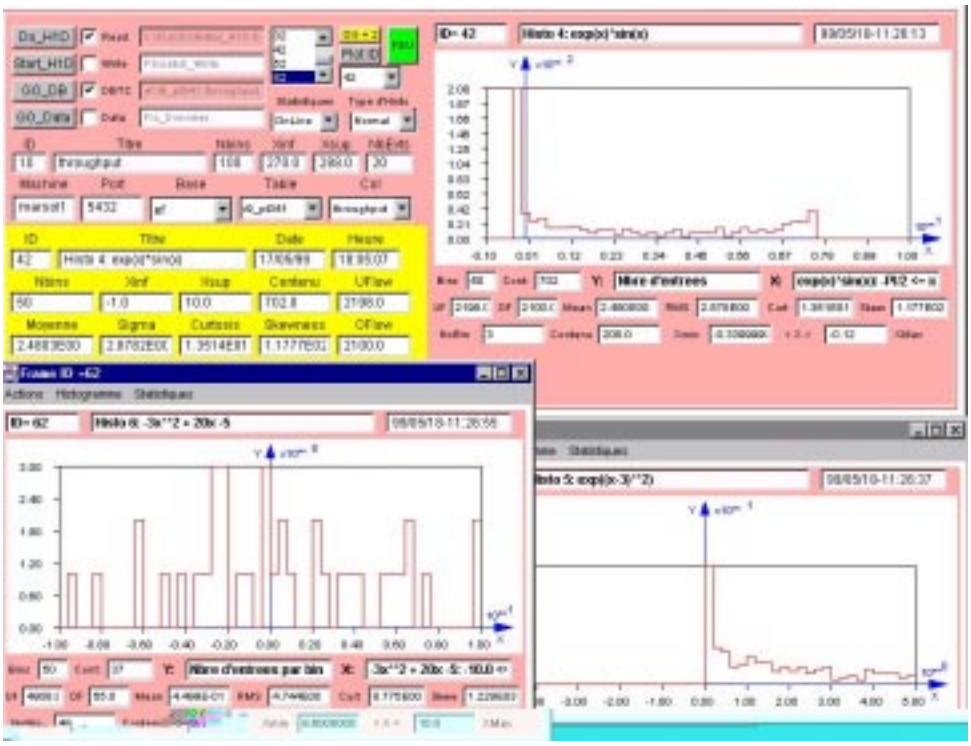


Figure 12 : Online analysis package

4 Using supervisor as end−user

For end−users, it is important that on one hand the agent framework should be as transparent as possible, and on the other hand that it is still possible to act upon it if desired. We try to follow this principle in the current implementation.

4.1 Start the whole system from scratch

Starting the system step by step :

- start **Voyager** for each host : done by script `StartVoyager`
- start **Supervisor Master** : done by script `SPVmaster` and `SPVstart`
- start **GUI** : done by script `SPVshow`

Typically, **GUI** and **Master** are running on different hosts.

4.2 Configure the SPV.ini

The Supervisor has a property file. Several parameters are predefined in this file, some of them can be changed at run−time, e.g. :

• `hostsPerAgent` : defines the degree of parallelism of the supervisor. It will play a role in farm start phase, and in dataflow data collection phase.

• `poolingTimer` : gives the pooling timeout

The most often used properties are described in Section 7.

4.3 Exercise the GUI

See above description about GUI.

5 Using the Supervisor as a developer

5.1 Package description

The system hierarchy is shown in Figure 13. The package is composed of 4 parts :

- bridge for Atlas online software (so−called ***be*** sub−package) : implemented, see below for detail.
- xml parser (so−called ***xml*** sub−package) : implemented, see below for description.
- core supervision (so−called ***spv*** sub−package) : implemented, see below for detail.
- Constant definition (so−called ***def*** sub−package)

Figure 14 shows the implementation view of the system.

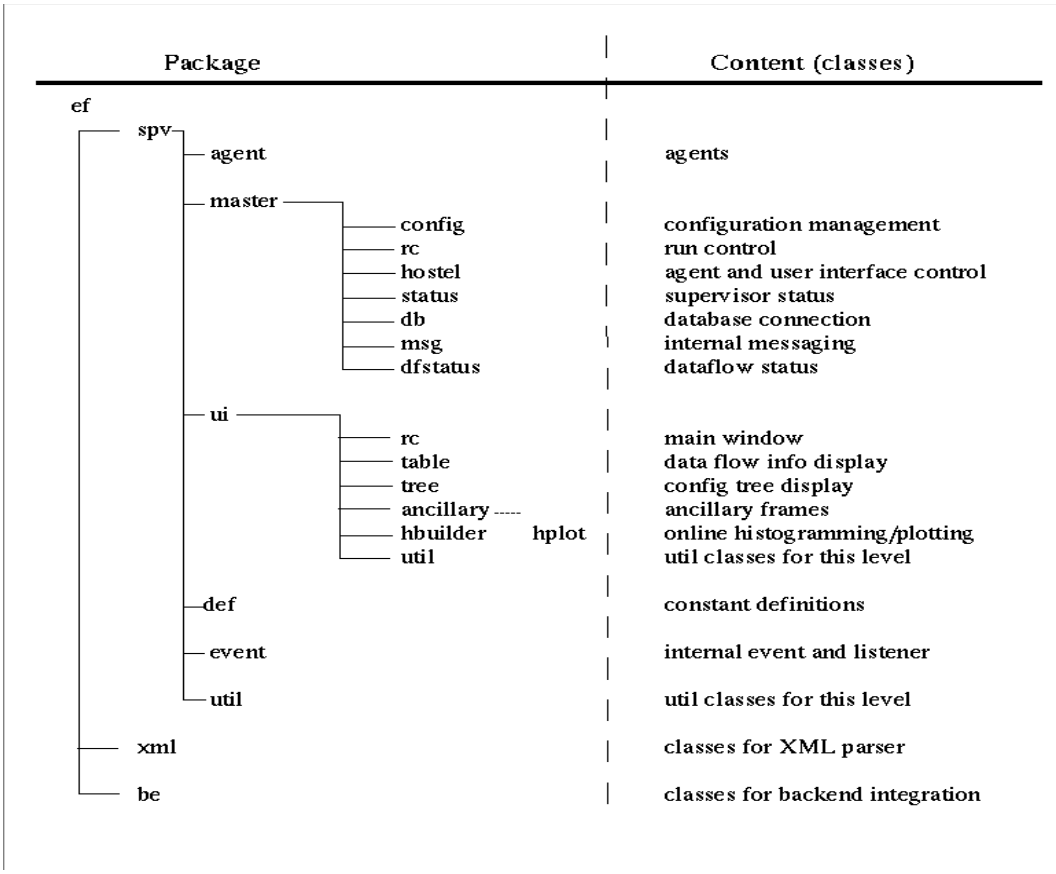


Figure 13: Package hierarchy

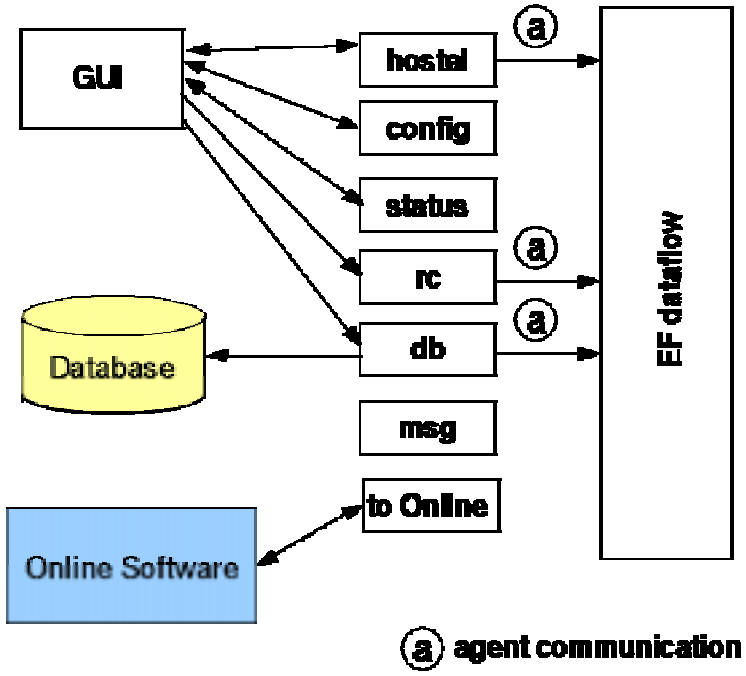


Figure 14: Implementation view

5.1.1 spv package

spv contains many components, each of them doing a well defined, limited job.

The package is also composed of 3 sub−packages :

- ***agent*** contains all agent classes
- ***master*** contains all master service classes
- ***ui*** contains all user interface classes

5.1.1.1 *master* package

master is the core component of the supervisor.

This package must provide the following functions : management of different kinds of agent; define the task and itinerary for each agent; retrieval of the results after having finished the job; management of abnormal behaviour of agent; management of different copies of GUI, each of them being able to perform separate monitoring or control; basic functionality of a control system (run control, process manager, message handling, monitoring ...); provision for future, possibly unplanned, system extensions.

In the current implementation, ***master*** is divided into 7 sub−packages :

- ***hostel*** : receives commands from various GUIs, sends dedicated agents to the Farm; retrieves information from agents when they have finished the job; sends information back to GUIs.
- ***config*** : for farm online configuration storage.
- ***status*** : is a set of classes for keeping various status of system
- ***rc*** : is the “run control” of farm. It receives the command from local control panel in case of

standalone operation and from higher level control process in case of integrated operation; sends dedicated agents to the farm; reports command execution status to command sender.

- ***dfstatus*** : performs acquisition of dataflow status
- ***db*** : an add−in package, run as bridge between hostel and a persistent database for monitoring data storage. Data connection is taken in charge by the agent, The persistent data will be used for further "offline" analysis.
- ***msg*** : receive message from various senders. The message can be then filtered and analysed by different tools.

5.1.1.2 *ui* package

All classes concerning graphic user interface are put into this package. The most often used classes are:

- `SVpanel` : main window
- `CfgSelector` : config file selection
- `ActivityViewer` : dataflow activity data display
- `ConfigTree` : farm configuration status
- `Reset` : start/stop any part of farm
- `DataflowParam` : dataflow component parameter display/update
- `NativeCmd` : use native command for different platform

5.1.1.3 *agent* package

This package contains all implemented agents classes in supervisor. Figure 15 shows the class hierarchy of the package.

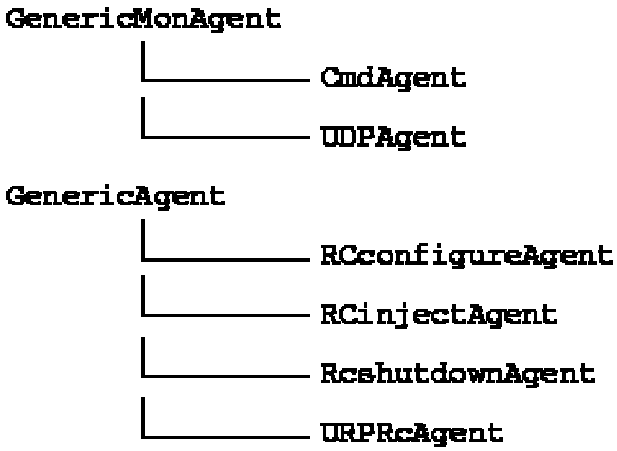


Figure 15: Class hierarchy

There are two distinct agents : monitor−agent inherited from `GenericMonAgent`, and control−agent inherited from `GenericRcAgent`. The function of each agent is :

- `CmdAgent` : carry users' command to target machine, execute the command, carry back the result if necessary.
- `UDPAgent` : go to target machine, send a UDP request message to dataflow component, go back with query result.
- `RCconfigureAgent` : start process on a remote machine.
- `RCinjectAgent` : perform injectOn, injectOff command, which corresponds to SFI start/stop.
- `RCshutdownAgent` : stop process on a remote machine.
- `UDPRcAgent` : go to target machine, send a UDP request to dataflow component, go back with query result.

In the Voyager toolkit, any object can become an agent at condition of

1. it implements `java.io.Serializable` interface;
2. it uses `Agent.of(this)` statement.

In our case, these two conditions are set by `GenericMonAgent` and `GenericRcAgent`. Only one method has to be written for the final classes : `atProgram()`. We take `CmdAgent.java` (see below) as an example to show how agents work.

The job of `CmdAgent` is to carry users' command to the remote location, perform the action (via `exec()`), get result of execution and finally go to next location. The `atProgram()` method defines what to do at each remote location when the agent moves into that node. The first thing the agent has to do is to determine what type of OS is actually running. The `next()` call at the end of the method lets agent go to the next location defined in the agent's itinerary list.

```
public void atProgram() {
      // get  local information
      try {
            Runtime runtime = Runtime.getRuntime();
            String osname = System.getProperty("os.name");
            Process proc = null;
            if ((osname.equals("Solaris")) || (osname.equals("Linux")))
                  proc = runtime.exec(unixCmd);
            if (osname.equals("Windows NT"))
                  proc = runtime.exec(winCmd);
            // waiting for result
            InputStream input = proc.getInputStream();
            BufferedReader in = new BufferedReader(
                                    new InputStreamReader(input));
            String s;
            while ((s = in.readLine()) != null) {
                  // do something
            }
      } catch(java.io.IOException e) {
```

```
            System.err.println(e);
        }
        next();
}
```

5.1.1.4 *event* package

Internal events are defined here. See Section 5.2 for event−driven description.

5.1.1.5 *util* package

All util classes used in this level are put in the package.

5.1.2 xml package

As we have described before, EF configuration file is in XML format. To read the file into the supervisor, we use **Voyager dxml** facility and **IBM xml4j parser**.

dxml is a toolkit used to create a set of Java classes based at a DTD file. The classes can then be used to get data from a XML file. This package actually contains the classes created from `cfg.dtd`.

5.1.3 def package

This package contains definitions of constant used by the Supervisor

5.1.4 be package

This package contains bridge−classes for Online Software / EF supervisor integration,. It is described in more details in section 6.

5.2 Supervisor component synchronisation : Event−Driven

Like other control and monitoring systems, we had to decide how to put components together. There are different methods to synchronise components in general : method call, messaging, event, ... On can choose one or another, or combine, depending on the system complexity. In the ***spv***, the relationship between components is complex. All relations need to be implemented correctly, and documented clearly for maintaining the current system and for future extension.

We decided to use "Java event" in most of case. **Event−driven** synchronisation gives the system an extra flexibility to arrange components' dependency, to ease unplanned extension which is a very important feature for a prototype with which one has little experience.

To make the components work together, we have defined several events. The components use the events to communicate with each other. Each component can be the sender of some events, the receiver of other events. We present the inter−component relationship by a matrix (Figure 16 and Figure 17) which determines the component dependency in the initial design. Because of the Java event−listener mechanism, adding new components and new events, re−arranging their dependencies becomes a simple game. The matrix gives the developer a global view of the system dependency, allowing to make more easily extensions.

Event / *Components*		*Config Stat*	*Rc Step*	*Db Stat*	*Moni Stat*	*Anomaly*	*Spv Info*	*Internal Msg*	*Ctrl Mode*	*Run Cmd*	*RunCmd Ack*
hostel	**S**				S	S	S		S		
	R	R	R	R				R			
config	**S**	S									
	R					R					
status	**S**										
	R	R		R	R		R				
rc	**S**		S			S					S
	R	R							R	R	
db	**S**			S		S					
	R										
msg	**S**							S			
	R										
be	**S**									S	
	R	R							R		R

*Figure 16: Component−Event matrix for **master** and **be***

Figure 16 describes Component−Event pairs for ***master*** package and ***be*** package, shows how components communicate with each other. As an example, component `config` **S**ends the **ConfigStat** event, which will be **R**eceived by components' `status`, `rc` and `hostel`.

The same mechanism is also implemented for ***ui*** package (see fig 16). As an other example , when user changes UI from control mode to monitoring mode, the `SPpanel` component **S**ends a **PanelModeChange** event which will be **R**eceived by `CfgSelector`, `DataflowParam` and `Reset`. When `SVpanel` **R**eceives a new report from an agent, it **S**ends **NewReport** event to the other windows : `Dataflow Param`, `ActivityViewer` and `ConfigTree`**.**

Components	*Events*	*New Report*	*PanelMode Change*	*ConfigTree Select*	*DataReady*
SV panel	**S**	S	S		
	R				
Cfg Selector	**S**				
	R		R		
Dataflow Param	**S**				
	R	R	R		
Reset	**S**				
	R		R		
Activity Viewer	**S**				S
	R	R		R	
ConfigTree	**S**			S	
	R	R			
Hbuilder	**S**				
	R				R

Figure 17: Component−Event matrix for GUI

5.3 Proxy generation

To build a distributed system, each class which will be contacted by a remote process has its representative in the remote system, called a *proxy*. Depending on the communication tools used by components (CORBA, rmi, Voyager ...), the way to construct proxies is different. Voyager provides two methods to construct a proxy :

- static : through **pgen** tool
- dynamic : through dynamic proxy generation system to generate proxy classes at runtime.

By default, Voyager creates proxy classes based on the interfaces the class implements, i.e. a remote deployment class `XXX.java` must have its interface class called `IXXX.java` which will be used for dynamic proxy generation. The advantage of interface−based proxies is that when using dynamic proxy class generation, Voyager will not require the implementation class to be present. This can be desirable for security reasons, to reduce remote classloading, or to be able to deploy a smaller `.jar` file on the client.

5.4 Implementation environment

Platforms : Sun Solaris 2.7, Windows NT 4.0, Linux RedHat 6.2, TruUnix64 4.0F (Alpha)

Packages :

Java : jdk 1.3

Mobile agent system : Voyager 3.3

6 Online software / EF Integration

The proposal of the integration has been published in [5].

6.1 Overview of organisation

A "2 component bridge" was inserted between Online Software (OS) and Event Filter Supervisor (EFS). Figure 18 shows the overall organisation in the case of a single farm.

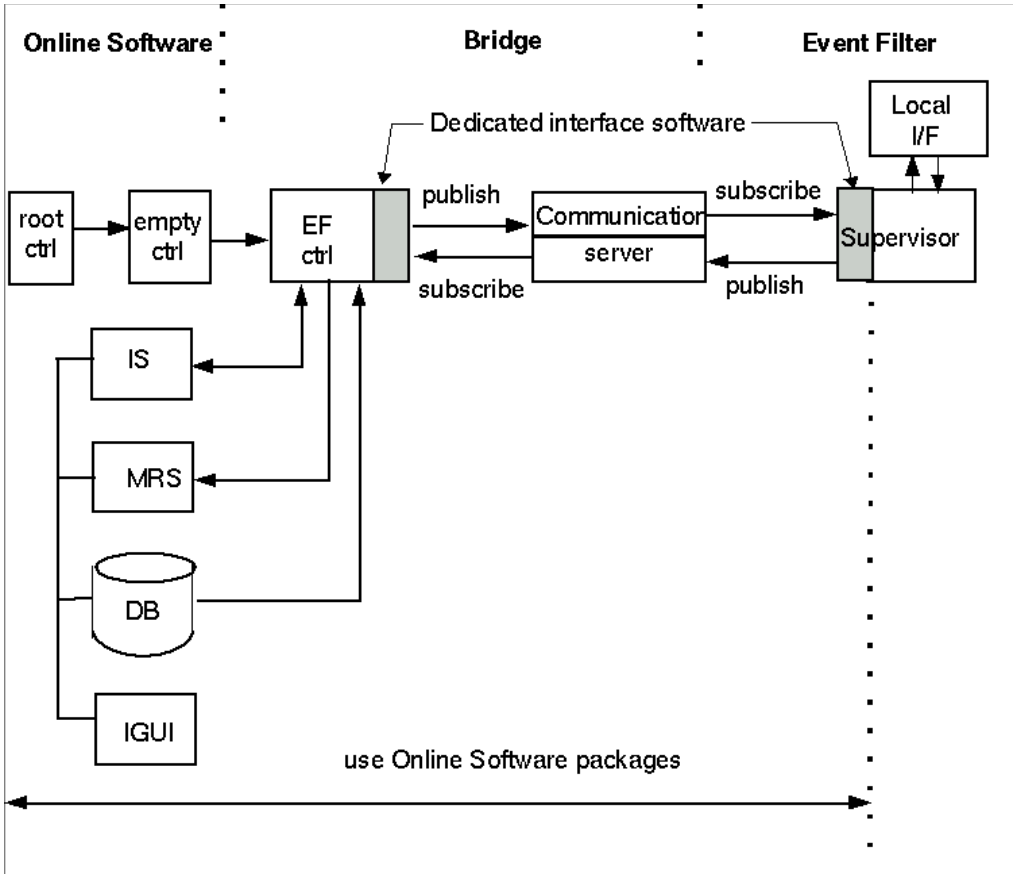


Figure 18 : Integration overall scheme

The first component is *EFctrl*, which uses the Controller skeleton of the Run Control. It provides the essential functionality of the Online Software: the standard RC Finite State Machine, Database access, the publish/subscribe mechanism of IS and MRS which are needed by the EF Supervisor to synchronise with the Online Software. It communicates with the *Root Controller* via the so−called *empty Controller* in charge of building the EF state.

The second component is a communication server which provides a bidirectional transfer mechanism for data exchange between OS and EFS via a publish/subscribe message model. It can be a CORBA server or a Message Oriented Middleware (MOM). The EF Supervisor uses this component to receive Online Software data (commands, IS information,
data (status, messages, ...). EF monitoring data, even in a complex form such as histograms, can also be sent to the the Online Software Integrated Graphical User Interface (IGUI) by this means, provided the latter is able to handle such data. For the phase I integration, we have chosen to use the Information Service package [6] in order to implement this server.

6.2 Items for exchange

Presently, the following items are exchanged between OS and EFS. Other items can be added at a later stage :

RunCtrl.EF_Ctrl1_command ---> contains DAQ rctrl command

RunCtrl.EF_Ctrl1_efstate ---> contains rctrl cmd execution result

RunCtrl.EF_Ctrl1_dbname ---> contains EF configuration file name

RunCtrl.EF_Ctrl1_info ---> contains EF general status

6.3 Implementation

As described in Figure 13, all integration classes are grouped into the ***be*** package. This piece of code listens to events sent by the other component and sends events to the related component.

Currently the following 5 events are used :

RunCmdAckEvent

ConfigStatEvent

CtrlModeEvent

InfoEvent

RunCmdEvent

The event relationship is described in Figure 16, except for InfoEvent which is sent by Online Software class.

Figure 19 shows the interaction sequence related to any of the events :

1. ***be*** package subscribes to RunCtrl.EF_Ctrl1_command and waits for an event
2. **Bridge** sends *InfoEvent*
3. ***be*** package gets *InfoEvent* then sends *RunCmdEvent* with parameter
4. ***rc*** package receives *RunCmdEvent*, performs the corresponding action then sends *RunCmdAckEvent*.
5. ***be*** package gets *RunCmdAckEvent and* publishes it to **Bridge**

...) and send EF

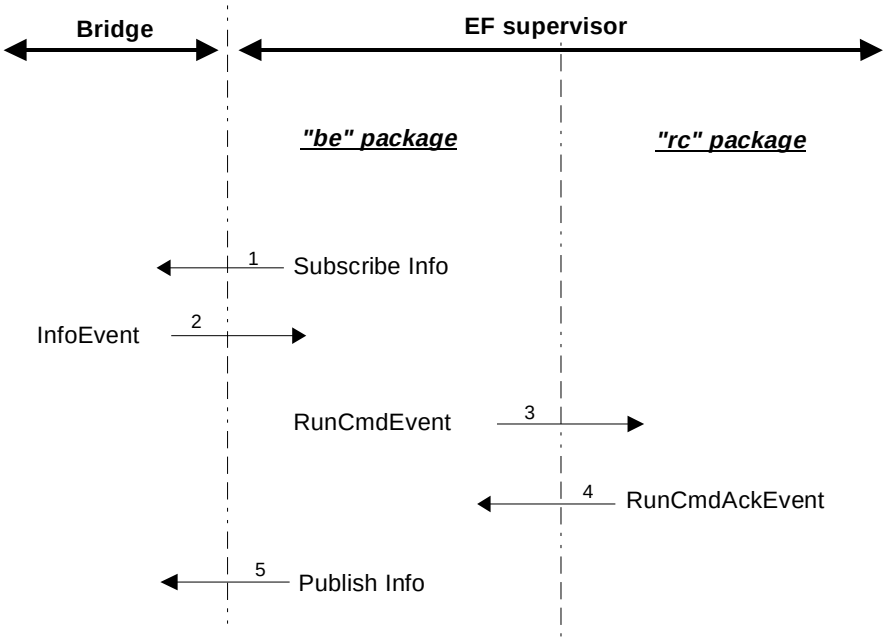


Figure 19 : Interaction sequence

6.4 Run EF with DAQ

6.4.1 Start EF from script

The EF processes can be started from outside of its context. The following steps show how to do it :

1) Install in each EF machine 3 packages : *dataflow*, *monitoring* and *script.* These files can be obtained from the tar file on http://atddoc.cern.ch/Atlas/EventFilter/activity.html

To avoid unnecessary path and classpath problems, it is better to install them in the user home directory. On the `lnxatd24` machine, currently used for EF tests at CERN, the files are installed in the directory `/home/effuser`. The password of the account `effuser` can be obtained on request to Z. Qian (`qian@cppm.in2p3.fr`)

2) Choose a host, later referred as the "EF Entry Point (EEP)" which is running *afs* and is reachable from the outside. Do the following two preparation steps in the *script* directory of this machine

- prepare the EF configuration file using the dedicated user interface

  `EFDB.tcl` ("expert" to prepare the configuration database)

  `EFConfig.tcl` (to finalise the configuration file)

- make EF setup files using the command :

  `makelist config-file-name`

  This tools creates a set of files which will be used by the different scripts :

  `ef_nshost` : EF Naming server running machine

`ef_nsport` : port number used by Naming server

`ef_master` : EF Master running machine

`ef_gui` : EF supervisor gui running machine

`ef_slave` : EF dataflow machine list

Note : `makelist` will be executed only if the EF is stopped (state `off`, see below).

3) From any outside machine, start and stop the EF processes using following rsh commands:

- `rsh EEP script/play_ef_start partition gui_display_address`
- `rsh EEP script/play_ef_stop`

Figure 20 shows EF start−up sequence chart. Figure 21 shows an example of play_ef_start log screen .

EF has only two states : `on` and `off`, this information is created by `play_ef_start` and `play_ef_stop,` and stored in the `ef_stat` file.

6.4.2 Dedicated tools for integration test

Some tools have been made available to ease the integration task. They can be used from the Main window menu (see Figure 7).

PanelMode allows to select the mode of the main window : "Standalone", "Combine", "Combine be" and "Combine df". The differences between these modes are the following :

- **Combine** : dismiss all local run control command, dismiss local Sfi/Sfo
- **Combine be** : dismiss all local run control command, enable local Sfi/Sfo, used for Back−end integration test.
- **Combine df** : enable local run control command, dismiss Sfi/Sfo start, used for dataflow integration test.

BeTools menu allows to :

- send individual EF status to Backend
- send EF information to Backend (to be defined)
- change partition without complete stop of EF, can be used when Backend restart or using different partition for test ...
- switch the shutdown command on or off. When it is switched off, EF ignores "unconfig−ure" commands sent by Backend and no process will be killed.

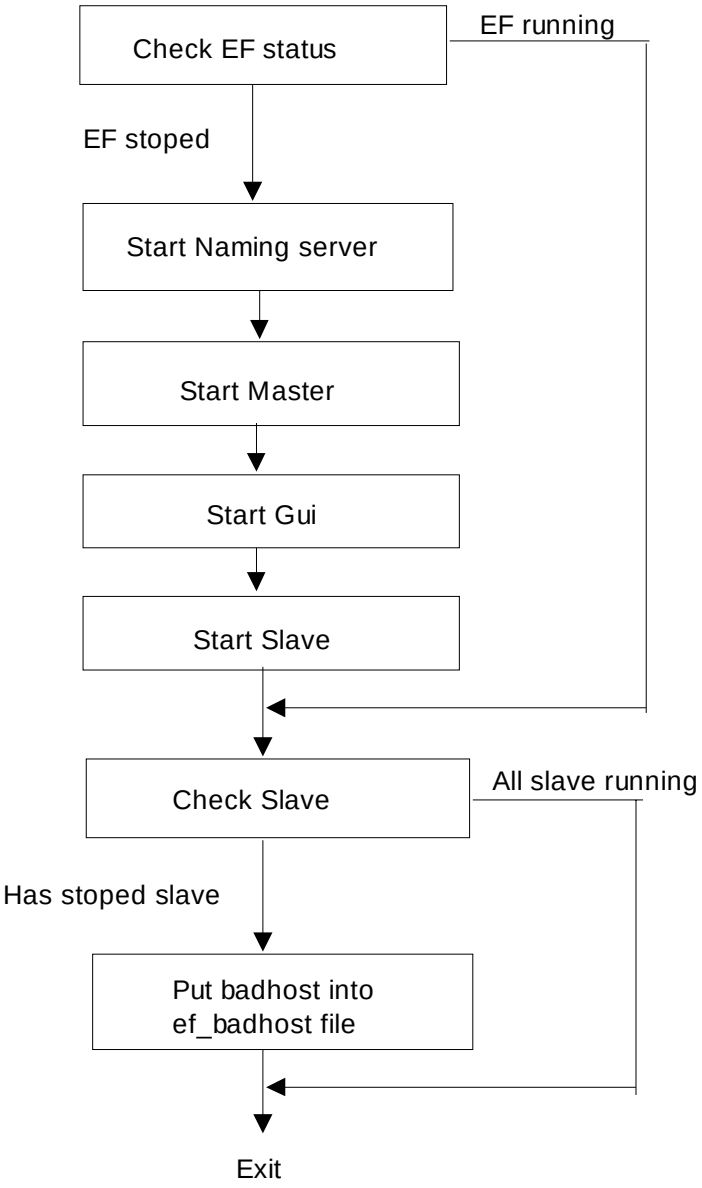


Figure 20 : Sequence chart of play_ef_start script

```
[marsol3] play_ef_start my_partition marsol3
   do EF start
   NServer: no process killed
   [1] 8215
   marntr2
   voyager: no process killed
   java: no process killed
   [1] 28262
   marntr1
   voyager: no process killed
   java: no process killed
   [1] 11506
   voyager: no process killed
   java: no process killed
   [1] 8231
   [2] 8272
   [2]  - Done         SPVstart my_partition >& ../script/log_start_master_SPVstart
   EF DISPLAY : marsol3:0.0
   [1] 8325

   start Voyager slave checking ...

      check address : marntr2:9100
      check address : marntr1:9100
         at tcp://marntr2:9100  Linux
         at tcp://marntr1:9100  Linux
      ---------------------------
      Running slave number : 2/2
```

Figure 21 : play_ef_start screen log

6.4.3 Critical points

The following points should be carefully considered :

1) DAQ cosnaming reference IPC_REF_FILE must be correct, this reference is defined in ~/monitoring/envm of the EFS master.

2) The full path to the EF configuration file must be stored into the DAQ configuration database.

7 System properties

Supervisor uses a set of system properties to control itself, including control panel presentation, agent behaviour, integration parameter, etc... All properties are stored in the `SPV.ini` file in the **~/monitoring** directory. The table below shows the list of properties used to set the configuration.

Property	***Possible value***	***Description***
SPV.master	−	SPV master address
SPV.defaultControlMode	STANDALONE COMBINE COMBINE_BE COMBINE_DF	See section 6.4.2 for mode description
SPV.defaultShutdown	true false	See section 6.4.2 for switch description
SPV.hostsPerAgent	Any positive number	This parameter defines the degree of parallelism of the supervisor. = 1 : fully parallel operation = 99999 : fully sequencial op. The value will play a role in farm start phase, and in dataflow data collection phase.
SPV.poolingTimer	Any positive number (in second)	When launch parallel agents for pooling dataflow info, supervisor set this timeout for waiting latest agent return
SPV.configureTimeout	Any positive number (in second)	Timeout for run control "configure" cmd execution. Supervisor wait the timeout before launch check−agent.
SPV.unconfigureTimeout	Any positive number (in second)	Timeout for run control "unconfigure" cmd execution.
SPV.stopTimeout	Any positive number (in second)	Timeout for run control "stop" cmd execution.
SPV.dfJavaImp	true false	Define which version of dataflow is to be started (Java or C++)
SPV.javaVM	−	Java VM path in dataflow machine. Only used when SPV.dfJavaImp=true.
SPV.classpath	−	Java classpath in dataflow machine. Only used when SPV.dfJavaImp=true.
BE.partition	−	Default value, can be changed in the run time
BE.server	−	Fixed value, can not be changed

8 Reference

[1] The ATLAS Event Filter, C.P. Bee *et al.*, Real Time 99, Santa Fe, USA, Conference proceedings http://atddoc.cern.ch/Atlas/EventFilter/documents/rt99−157−paper.pdf

[2] Event Handler Supervisor High Level Design, ATLAS DAQ−1 Note 92, http://atddoc.cern.ch/Atlas/Notes/092/Note092−1.html

[3] Java Mobile Agent for monitoring task, ATLAS DAQ−1 Note 78, http://atddoc.cern.ch/Atlas/Notes/078/Note078−1.html

[4] Voyager − http://www.objectspace.com/voyager

[5] Event Filter / Online software Integration, ATLAS COM−DAQ Note 2000

[6] Online Software Summary Document, ATLAS DAQ Note 2000−

Appendix 1 : Configuration file

DTD file

```
<!ELEMENT Cfg (GeneralInfo?,Subfarm+)>
<!ELEMENT GeneralInfo (NsAddress,MasterHost,GuiHost)>
<!ELEMENT NsAddress (#PCDATA)>
<!ELEMENT MasterHost (#PCDATA)>
<!ELEMENT GuiHost (#PCDATA)>
<!ELEMENT Subfarm (Sname,SfiHost?,SfoHost?,SfiCmd?,SfoCmd?,Node+)>
<!ELEMENT Sname (#PCDATA)>
<!ELEMENT SfiHost (#PCDATA)>
<!ELEMENT SfoHost (#PCDATA)>
<!ELEMENT SfiCmd (#PCDATA)>
<!ELEMENT SfoCmd (#PCDATA)>
<!ELEMENT Node (Name,BinDir,IorDir,Task+)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT BinDir (#PCDATA)>
<!ELEMENT IorDir (#PCDATA)>
<!ELEMENT Task EMPTY>
<!ATTLIST Task
          Id CDATA #REQUIRED
          Type CDATA #REQUIRED
          Sequence CDATA #REQUIRED
          Exec CDATA #REQUIRED>
```

XML configuration file (an example)

The XML file generated by the configuration GUI, which contains 2 sub−farms of 1 node each running 1 processing task.

```
<?xml version=”1.0” encoding=”ASCII”?>
<!DOCTYPE Cfg SYSTEM “http://atddoc.cern.ch/Atlas?EventFilter/xml/dtd/cfg.dtd”>
<Cfg>
    <GeneralInfo>
        <NsAddress>marntr4:7200</NsAddress>
        <MasterHost>marntr4</MasterHost>
        <GuiHost>marntr4</GuiHost>
    </GeneralInfo>
    <Subfarm>
        <Sname>sub1</Sname>
        <SfiHost>marntr1</SfiHost>
        <SfoHost>marntr1</SfoHost>
        <SfiCmd>/home/efuser/dataflow/bin/sfi sub1_d1 marntr4:7200
/home/efuser/dataflow/geo/GeomRun.cz</SfiCmd>
        <SfoCmd>/home/efuser/dataflow/bin/sfo sub1_c1 marntr4:7200</SfoCmd>
        <Node>
            <Name>marntr1</Name>
            <BinDir>/home/efuser/dataflow/bin/</BinDir>
            <IorDir>/home/efuser/dataflow/ior/</IorDir>
            <Task Id=”sub1_d1” Type=”fifo:d1d2” Sequence=”1” Exec=”d1d2 −id_d1
sub1_d1 −id_d2 sub1_d2 −nbr_d2 1 −type fifo −ns marntr4:7200 −path
/home/efuser/dataflow”/>
            <Task Id=”sub1_d2_0” Type=”fifo:d1d2” Sequence=”1” Exec=“ps”/>
            <Task Id=”sub1_c1” Type=”fifo:c1” Sequence=”2” Exec=”fifo −id sub1_c1
−depth 1 −type fifo −ns marntr4:7200 −path /home/efuser/dataflow”/>
            <Task Id=”sub1_pt1” Type=”pt:01” Sequence=”3” Exec=”pt −id sub1_pt1 −type
pt −eventtype 01 −ns marntr4:7200 −path /home/efuser/dataflow−geo GeomRun.cz −PTHi 10
−reject 50 −src sub1_d2_0 −dst sub1_c1”/>
        </Node>
    </Subfarm>
    <Subfarm>
        <Sname>sub2</Sname>
        <SfiHost>marntr2</SfiHost>
        <SfoHost>marntr2</SfoHost>
        <SfiCmd>/home/efuser/dataflow/bin/sfi sub2_d1 marntr4:7200
/home/efuser/dataflow/geo/GeomRun.cz</SfiCmd>
        <SfoCmd>/home/efuser/dataflow/bin/sfo sub2_c1 marntr4:7200</SfoCmd>
        <Node>
            <Name>marntr2</Name>
            <BinDir>/home/efuser/dataflow/bin/</BinDir>
            <IorDir>/home/efuser/dataflow/ior/</IorDir>
            <Task Id=”sub2_d1” Type=”fifo:d1d2” Sequence=”1” Exec=”d1d2 −id_d1
sub2_d1 −id_d2 sub2_d2 −nbr_d2 1 −type fifo −ns marntr4:7200 −path
/home/efuser/dataflow”/>
            <Task Id=”sub2_d2_0” Type=”fifo:d1d2” Sequence=”1” Exec=“ps”/>
```

```
            <Task Id=”sub2_c1” Type=”fifo:c1” Sequence=”2” Exec=”fifo −id sub2_c1
−depth 1 −type fifo −ns marntr4:7200 −path /home/efuser/dataflow”/>

            <Task Id=”sub2_pt1” Type=”pt:01” Sequence=”3” Exec=”pt −id sub2_pt1 −type
pt −eventtype 01 −ns marntr4:7200 −path /home/efuser/dataflow−geo GeomRun.cz −PTHi 10
−reject 50 −src sub2_d2_0 −dst sub2_c1”/>

        </Node>

    </Subfarm>

</Cfg>
```