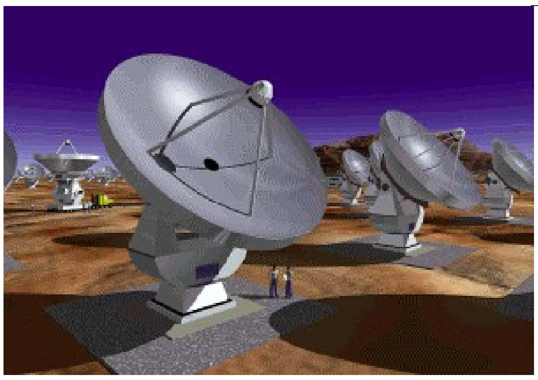


Atacama Large Millimeter Array	ALMA-SW-NNNN Revision: 1.8 2010-01-13

ACS Alarm System

Software Architecture and How-to manual

Alessandro Caproni (acaproni@eso.org),
ESO

Bogdan Jeram (bjeram@eso.org)
ESO

Keywords:
Author Signature:Date:
Approved by:Signature:Institute:Date:
Released by:Signature:Institute:Date:

Change Record

REVISION	DATE	AUTHOR	SECTIONS/PAGES AFFECTED
	REMARKS		
1.0	2006-07-25	Alessandro Caproni	All
	Created		
1.1	2006-09-20	Alessandro Caproni	All
	Revised after integrating the AS into ACS		
1.2	2006-09-26	Alessandro Caproni	CDB – Known problems
1.3	2006-10-30	Alessandro Caproni	All
	Revised for ACS 6.0		
1.3.1	2007-11-19	Alessandro Caproni	All
	Notes from Joe Schwarz		
1.4	2007-11-26	Alessandro Caproni	All
	ACS 7.0		
1.5	2008-03-17	Alessandro Caproni	CategoryClient, AlarmPanel
	ACS 7.0.1		
1.6	2009-05-31	Alessandro Caproni	All
	ACS 8.0		
1.7	2009-08-10	Alessandro Caproni	All
	GUI chaperts updated; a new chapter for sending alarms through IDL		
1.8	2010-01-12	Bogdan Jeram	BACI property
	Added description how FF/FM can be set. Values of FC. Alarm properties for BACI alarms. ACS-8.1		
1.9	2014-01-22	Alessandro Caproni	All
	Reviewed for the new API		

Table of Contents

1 Introduction

The alarm system (AS) is a messaging system: it collects, manages and distributes information about abnormal situations and shows what's happening to the user.

In a complex environment, a malfunctioning, either in the software or the hardware, might trigger a chain of malfunctions in other equipment. The main purpose of the alarm system is to help the user to identify the root cause of a problem in such a way that he can fix the problem in a short time. This is achieved comparing the alarms active at a given time against a knowledge base describing the correlation between the alarms.

The alarms are sent by the sources to the Alarm Service Component (ASC) whenever an abnormal situation is detected. The ASC listens for the alarms and when a new alarm is received it looks in its knowledge base to see whether a correlation exists between the new alarm and the other alarms already active. If such a correlation is found then the ASC could mask/hide some alarms that are not relevant for identifying the root cause of the malfunction. Finally, the ASC builds a new version of the alarms to send to the clients. This new version is more complete and human readable than the alarms generated by the sources.

The AS has a set of clients; one of them is the operator GUI that shows the alarms to the operator.

In the distribution there are two alarm systems:

1. the LASER Alarm System, the subject of this document.
2. an ACS implementation of the Alarm System that logs a message for each alarm published

   The ACS implementation is used by default: you can see the alarms with jlog or the loggingClient. To switch to the CERN implementation, the Alarms branch must be present in the CDB and the Implementation property must be explicitly set to CERN[1].

The development of the AS is an ongoing process. The interfaces for the developers should remain untouched. However the AS is not complete and some part of the system may not be fully functional or not implemented yet. In particular, the ACS LASER AS derives from the AS developed at CERN for the Large Hadron Collider (LHC). At the present, the porting of the original AS is not complete and some parts of the original AS are missing.

1.1 Glossary

ACS Alma Common Software

ALMA Atacama Large Millimeter Array

AS Alarm System

ASC Alarm Service CORBA servant

CDB Configuration DataBase

[1]You can find more about the CDB for the alarm system in paragraph 4.

CERN European Organization for Nuclear Research

FC Fault Code

FF Fault Family

FM Fault Member

FS Fault State

GUI Graphical User Interface

JMS Java Message Service

LASER LHC Alarm SERvice

LHC Large Hadron Collider

MR Multiplicity Reduction

NC Notification Channel

NR Node Reduction

RR Reduction Rule

See also:

http://www.alma.nrao.edu/development/computing/docs/joint/draft/Glossary.htm

1.2 References

- ACS documentation: http://www.eso.org/projects/alma/develop/acs/OnlineDocs/index.html
- LASER project: http://proj-laser.web.cern.ch/proj%2Dlaser/
- LHC control project: http://lhc-cp.web.cern.ch/lhc%2Dcp/

2 Installation and test

The AS is part of the ACS distribution. It is composed of several modules: the modules of the ACS implementation of the AS, that are built and installed before the containers, and the CERN implementation of the AS, whose modules are grouped into ACSLaser and compiled just after ACS.

The AS is in SVN `ACS/LGPL/CommonSoftware/ACSLaser`. The GUIs are in `ACS/LGPL/CommonSoftware/acsGUIs`.

There is a little demo that can be used to test the functioning of the AS. The demo shows the functioning of the Node Reduction and the Multiplicity Reduction. A complete CDB is available for this purpose in the `demo` module.

The NR demo consists of three modules capable of sending alarms: ALARM_SOURCE_PS, ALARM_SOURCE_MOUNT and ALARM_SOURCE_ANTENNA. When the power supply fails, it sends an alarm and triggers a failure of the mount component. In the same way a failure in the mount components causes the sending of an alarm and the failure of the antenna component. The purpose of this demo is to show how the alarms are reduced and only the root cause of the problem (the alarm in PS) is shown in the GUI.

The MR demo consists of a single component, MULTIPLE_FAILURES, sending 5 different alarms when the `multiFault` IDL method is executed. The threshold for the MR is set to 3. The purpose of this demo is to show how several alarms of the same kind are reduced when the number of such active alarms is greater then the threshold.

Follow these steps to check your installation and execute the NR demo:

1. set `ACS_CDB` to the CDB folder in `ACSLaser/demo/test`
2. start ACS: it will start the ASC as part of the services
3. start the java container, `frodoContainer`
4. start the java container `javaContainer`[2]
5. start `objexp`
6. start the ALARM_SOURCE_PS component
7. start the ALARM_SOURCE_MOUNT component (be careful because there are 3 mount components, one for java, one for C++, called ALARM_SOURCE_MOUNTCPP and one for python, called ALARM_SOURCE_MOUNTPY)
8. start the ALARM_SOURCE_ANTENNA
9. start the GUI with the command `alarmPanel`; the panel panel has a green icon in the right bottom side of the main window meaning that the application is connected to the AS
10. from the `objexp`, select the PS component and execute its fault function.

    The execution of the PS `fault_PS` IDL method, causes the three components to send a chain of alarms that appear in the main window of the GUI[3]. If the reductions are turned on, you will see only one the alarm generated by the power supply failure i.e. the root cause of the chain of failures. If the reductions are disabled, all the alarms will appear in the GUI. A bottom in the toolbar of the panel allows to enable/disable the alarm reductions in the table.

    To execute the MR demo instead of the NR, in step 8 activate MULTIPLE_FAILURES and execute its fault method.

    In both examples, the terminate fault method will set all the alarms from ACTIVE to TERMINATE. The color of the alarms in the GUI will change to green indicating that the abnormal situation has been fixed.

[2] There are 2 java containers. The power supply and the mount run in `frodoContainer`; the antenna and MULTIPLE_FAUILURES components run in `javaContainer.`

[3] To simulate a chain of failure, each component sends an alarm, waits 5 seconds and calls the fault method of the next component. This means that the fault method of PS needs about 15 secs to terminate. The sleep time of 5 secs between the sending of an alarm and the execution of the IDL method of the next component of the chain has been introduced to help the user to look at the GUI and see the changes when each new alarms arrive. Objexp has a timeout of 5sec for each IDL method executed on a component so it will show a dialog saying that the fault method of PS has caused a timeout: this is not an error. To avoid this notification you should extend the timeout setting properly the objexp.pool_timeout property as described in the Object Explorer user manual.

The component ALARM_SOURCE_MOUNTCPP is a C++ component running inside the C++ container `bilboContainer`. It is there to show how to send alarms using C++ API but it is not configured for reduction.

The component ALARM_SOURCE_MOUNTPY is the demo of an alarm generated by a python component that runs inside a python container named aragornContainer.

The ASC submits a great number of log messages when reading the CDB. These messages are easy to identify filtering for the source object AlarmService.

3 AS architecture

The AS is a distributed, layered application. Each layer depends on the layer below and provides a set of services for the layer above. The definition of a clear set of interfaces drives this hierarchy.

As shown in fig.1, The software is composed of three tiers: the *resource tier*, containing a set of sources, to detect malfunctions and send alarms; the *business tier*, that implements the system logic and its services, having a knowledge base of the specific domain; the *client tier* composed of clients consuming the business services.

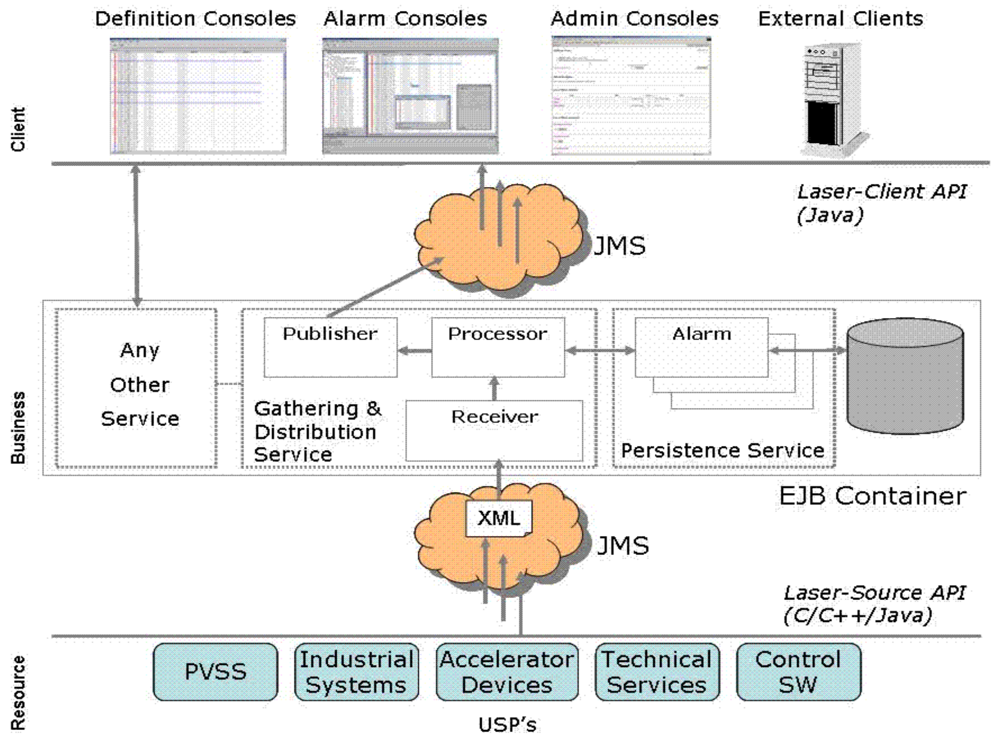


Figure 1: The architecture of the LASER alarm system

3.1 The resource tier

The resource tier is composed of the sources of alarms, i.e. applications that monitor the hardware and the software to detect malfunctioning. The sources can be written using different programming languages and run on different platforms.

Each alarm is identified by a Fault State (FS). A FS is composed by a triplet and an actiavation state. The triplet is composed of the Fault Family (FF), the Fault Member (FM) and the Fault Code (FC).

```
FS = <FF, FM, FC>
```

The FF is a string and identifies a set of elements of the same kind, for example the set of all the power supplies. The Fault Member, again a string, identifies a particular instance of the elements of the FF, for example PS3 i.e. a power supply, called PS3. The FC is an integer describing the specific type fault, for example 100 for over current.

Conversely speaking, a triplet is a unique identifier for an alarm. It means that for each possible alarm that can be generated by each possible source there is one and only one triplet. And vice versa, to each alarm corresponds one and only one triplet. The triplet does not say *where* an alarm happens but it says *which* alarm is occurring. The information *where* must be inserted in the database and we'll talk about that later.

These are examples of valid triplets:

- `<MOUNT, MOUNT_1, 100>` (ACS component)
- `<ANTENNA, ANTENNA7, 200>` (Hardware equipment)
- <kernelModule, pcnet32@host, 1> `(kernel module)`

In the examples, `MOUNT_1` is the name of a component. Each component can easily retrieve its name by calling the `getName` method of the `ContainerServices`. pcnet32 is the name of a kernel module, and host is the name of the host where the module is running.

The triplet identifies an alarm sent by a source to the ASC. The triplet is also used internally by the ASC to apply the reduction rules or retrieve more information about a specific alarm from the database. We can think at each triplet as a unique identifier used in low level computation. The information sent by the ACS to the clients, the operator GUI for example, is not a triplet as it has been received by a source but it is a human readable description of the alarm represented by that triplet.

There are no guidelines yet for the strings and numbers to use to define the triplets but common sense suggests to use meaningful strings for FFs and FMs and possibly coherent numbers for failures of the same kind. For example, for a component the FF could be its IDL interface and the FM could be its name. For a kernel module, the FM could contain the name of the module and the name of the host where the module is executed.

An FS, i.e. an alarm, has a activation state, *active* or *inactive*. When an abnormal situation occurs, a source sends an *active* alarm. When the problem has been fixed, the source sends the same alarm with status *inactive* to the ASC. When a source is created, there is no need to send an alarm, even an inactive one; but when an abnormal situation has been fixed and the alarm does not exist anymore, the source must send an inactive alarm.

An heartbeat mechanism is associated with a source so that in case of a problem so bad that the source itself crashed before sending an alarm, the ASC is able to detect that something happened to the source because of the missing receipt of the heartbeat.

3.2 Alarm source API

ACS provides a API for java, C++ and python[4] to connect the sources to the business tier and is very small in order to be as simple as possible for the user. the following shows a java example to send an alarm.

The following snipped shows hot set and clear an alarm from a java component:

```
// Set the alarm with FF=PS, FM=PS1, FC=1
m_containerServices.getAlarmSource().raiseAlarm("PS","PS1",1);

// Clear the alarm with FF=PS, FM=PS1, FC=1
m_containerServices.getAlarmSource().clearAlarm("PS",PS1",1);
```

Setting and clearing a alarm is done getting a `alma.acs.alarmsystem.source.AlarmSource` from the `ContainerServices` and calling its `raiseAlarm` and `clearAlarm` methods. The `AlarmSource` does more then that as we will see later.

The following shows how to set and clear the same alarm in C++:

```
// Set the alarm with FF=PS, FM=PS1, FC=1
getContainerServices()->getAlarmSource()->raiseAlarm("PS","PS1",1);

// Clear the alarm with FF=PS, FM=PS1, FC=1
getContainerServices()->getAlarmSource()->clearAlarm("PS","PS1",1);
```

The alarms produced by the sources are sent to the ASC (through JMS for java) with a dedicated notification channel. It is possible to configure the AS in order to use more channels and it is also possible to define which channel is used by each source to send an alarm to the ASC[5]. The ASC receives all the existing alarms listening at all the source notification channels.

There are two implementations of the alarm system. One uses the CERN implementation i.e. the sources send alarms to the ASC by means of notification channels. Alternatively, it is possible to use the ACS implementation i.e. the sources log a message for each alarm they send. The implementation to use can be chosen by changing a property in the CDB[6]. The API is independent of the type of alarm system in use so you can freely switch between the CERN and the ACS implementation without changing the code. CERN implementation is meant to be used in production. ACS implementation can be used for testing.

3.2.1 Extended features of `AlarmSource`

We showed how to send alarms with the help of the AlarmSource returned by the ContainerServices. This is the easiest way to set and raise alarms. But the AlarmSource

[4] Python API is not as advanced as that of the other languages. You can find a example for setting and clearing alarms with python in the next chapter.

[5]In the present version of the AS, all the alarms are published in the same channel named `ALARM_SYSTEM_SOURCES`.

[6]See chapter 6.

does much more then that. The C++ and Java sources of AlarmSource have a detailed documentation that you should carefully read. There is no AlarmSource available for python.

To reduce the traffic on the source NC,AlarmSource does not send twice the same alarm if its activation state did not change. It means that if a piece of code sets (clears) the same alarm n-tmes then the activation (termination) is published only once in the alarm source notification channel. In this way there are less useless alarms published in the source NC and less useless alarms processed by the alarm source. This functionality is embedded in the raiseAlarm and clearAlarm methods and transparent to the developer.

To damp the effect of oscillation each active alarm is immediately forwarded to the ASC but the clearing of a alarm is delayed of about one second to catch the case of a reactivation in that time interval. Also this feature is embedded in the raiseAlarm and clearAlarm methods.

It happens quite often that devices send spurious alarms during their initialization; this is usually due to the fact that the values retrieved from the hardware do not reflect the real state of the device before its initialization completes.
At the end of the initialization the alarms activated by spurious values are cleared by reading the values from the device. If some alarm is still active after initialization it means that a real problem has been detected.

The AlarmSource allows to queue all the alarms generated during the initialization phase of a device[7] by calling queueAlarms[8] before starting the initialization. The flushAlarm method must be called when the initialization terminates.
After calling flushAlarms, the AlarmSource does not forward the alarms generated with raiseAlarm or clearAlarm to the ASC, it, instead, saves them in a internal map together with their activation state. When flushAlarm runs, then the AlarmSource flushes the alarm in the queue with their last activation state.

Three more methods are provided by the AlarmSource:

- `terminateAllAlarms`: terminates all the active alarms
- `disableAlarms/enableAlarms`: alarms generated after calling disableAlarms are immediately discarded until enableAlarms has been executed. Those alarms are not queued and will never arrives to the ASC.

3.3 Legacy API

The API presented in the previous chapter has been written to better integrate the alarm system into ACS and offers to the developers improved features with a easier syntax hiding all the details of the alarm system.
This chapter shows the legacy API for historical reasons and because it can still be found in some of the sources of the ALMA software. Unless you have a compelling reason, this API should not be used.

The following shows a java example to send an alarm.

```
import alma.alarmsystem.source.ACSAlarmSystemInterfaceFactory;
```

[7] Actually the initialization of a device is a particular case of the usage of the queuing. The general use case is whatever phase of a computation that temporarily generates a bounce of alarms that do not reflect a real problem.
[8] Two flavors of queueAlarms are available one of which accepts a time interval: when the time is elapsed then the alarms are flushed without the need of calling flushAlarms. This implementation is useful when the time to initialize a device in known in advance.

```
import alma.alarmsystem.source.ACSAlarmSystemInterface;
import alma.alarmsystem.source.ACSFaultState;
...
public void send_alarm(
       String faultFamily,
       String faultMember,
       int faultCode,
       boolean active)
{
ACSAlarmSystemInterface alarmSource =
       ACSAlarmSystemInterfaceFactory.createSource(this.name());
ACSFaultState fs =
       ACSAlarmSystemInterfaceFactory.createFaultState(
              faultFamily, faultMember, faultCode);
if (active) {
       fs.setDescriptor(ACSFaultState.ACTIVE);
} else {
       fs.setDescriptor(ACSFaultState.TERMINATE);
}
fs.setUserTimestamp(new Timestamp(System.currentTimeMillis()));
Properties props = new Properties();
...
props.setProperty(...);
fs.setUserProperties(props);
alarmSource.push(fs);
}
```

The following shows a C++ example to send an alarm:

```
#include "ACSAlarmSystemInterfaceFactory.h"
#include "AlarmSystemInterface.h"
#include "FaultState.h"
#include "faultStateConstants.h"
#include "Timestamp.h"
#include "Properties.h"

using namespace acsalarm;
...

// constants we will use when creating the fault
string family = "AlarmSource"; // FF
string member = getComponent()->getName(); // FM, the name of the component
int code = 1; // FC

// Create the AlarmSystemInterface using the factory
auto_ptr<AlarmSystemInterface> alarmSource =
       ACSAlarmSystemInterfaceFactory::createSource();

// Create the FaultState using the factory
auto_ptr<acsalarm::ACSFaultState> fltstate =
       AlarmSystemInterfaceFactory::createFaultState(family, member, code);

// Set the fault state's descriptor
string stateString = faultState::ACTIVE_STRING;
fltstate->setDescriptor(stateString);

// Create a Timestamp and use it to configure the FaultState
Timestamp * tstampPtr = new Timestamp();
auto_ptr<Timestamp> tstampAutoPtr(tstampPtr);
fltstate->setUserTimestamp(tstampAutoPtr);

// Create a Properties object and configure it, then assign to the FaultState
Properties * propsPtr = new Properties();
propsPtr->setProperty(faultState::ASI_PREFIX_PROPERTY_STRING, "prefix");
propsPtr->setProperty(faultState::ASI_SUFFIX_PROPERTY_STRING, "suffix");
```

```
propsPtr->setProperty("TEST_PROPERTY", "TEST_VALUE");
auto_ptr<Properties> propsAutoPtr(propsPtr);
fltstate->setUserProperties(propsAutoPtr);

// Push the FaultState to the alarm server
alarmSource->push(*fltstate);
```

The following shows a python example to send an alarm.

```
import Acsalarmpy
import Acsalarmpy.FaultState as FaultState
import Acsalarmpy.Timestamp as Timestamp

Acsalarmpy.AlarmSystemInterfaceFactory.init()

alarmSource=Acsalarmpy.AlarmSystemInterfaceFactory.createSource("ALARM_SYSTEM_S
OURCES")
fltstate=Acsalarmpy.AlarmSystemInterfaceFactory.createFaultState(family,member,
code)

fltstate.descriptor = FaultState.ACTIVE_STRING

fltstate.userTimestamp = Timestamp.Timestamp()
fltstate.userProperties[FaultState.ASI_PREFIX_PROPERTY_STRING] = "prefix"
fltstate.userProperties[FaultState.ASI_SUFFIX_PROPERTY_STRING] = "suffix"
fltstate.userProperties["TEST_PROPERTY"] = "TEST_VALUE"

alarmSource.push(fltstate)


Acsalarmpy.AlarmSystemInterfaceFactory.done()
```

The sources build a message containing the triplet and an action, like active or terminate. The API embeds the message in a structure and publishes the message in a notification channel to the business tier.

To define an alarm as active or inactive, the `setDescriptor` method must be used passing the right string. It is important to remember that the descriptor of an alarm is the status of the alarm itself and not a generic string to describe what's happening at a certain instant. In fact, the description of the alarm is stored in the database.
In java, the descriptor of the alarm must be one of the constant String of `alma.alarmsystem.source.ACSFaultState` (for example `ACSFaultState.ACTIVE`). In C++, the descriptors are defined in `faultStateConstants.h` under the `faultState` namespace (like for example `faultState::ACTIVE_STRING`).

It is a common mistake to set the descriptor of an alarm to a user defined string in the `setDescriptor` method. In that case the alarm system will ignore the alarm (i.e. the alarm will be discarded as unrecognized and it will not be processed neither sent to the clients).

3.4 The business tier

The business tier is the core of the alarm service:

- listens for FS changes and heartbeats from the sources

- reads the further data of a received alarm from the database
- reduces or masks the FS depending on the knowledge of the environment and the current status of the system
- persists the FS
- traces and archives the changes of the FS
- allows management of the alarm system without stopping the alarm service component
- authenticate users on the client GUIs

All these services are realized by the ASC and the communications between the upper and the bottom layers happen through a definite API.
The persistence, trace and archiving of FS are not implemented in this version of the AS.

In order to keep the Laser-source API simple, a source sends to the business layer only the triplet describing the alarm with the time of its creation and its activation state. For each alarm received, the ASC reads its complete definition from the CDB in order to present a complete snapshot of the situation, its possible solution and consequences to the operators. The following table shows some of the information stored in the database for each alarm[9].

Property name	***Description***
system-name	The name of the system
identifier	The identifier of the system
problem-description	The description of the problem
cause	he cause of the problem
action	The action the operator must follow to fix the problem
consequence	A description of the consequences of the alarm
priority	The priority of the alarm: there are four priority levels
contact name	The name of the responsible person
contact gsm	A GSM number to call to notify about the problem
contact email	The email address of the responsible person for the problem
help-url	An url that point to the documentation of the problem
source-name	The name of the source
location	The location of the source

One of the most relevant parts of the business tier is the reduction of the alarms. In a complex environment where a failure can cause a cascade of secondary alarms, it is very important to show to the operators the root cause of a problem. Operators are also confused when the operator GUI shows a great number of repeated alarms of the same type. The alarm reduction mechanism addresses both these problems.

To perform the reduction, the alarm system reads from the database a set of dependency rules between alarms describing their correlation. Whenever the service receives a FS change, it applies that set of rules and eventually marks some alarms as reduced.

[9] The triplet is the unique identifier to get the informations shown in the table from the database. The data in the table should be more interesting then the triplet for the operators.

All the alarms, both reduced and not reduced, will be sent to the client because some clients may be interested in receiving all the alarms regardless of their reduction status: it is the GUI that hides the reduced alarms to the operators depending on the specific configuration.

There are two types of reduction rules:

- *node reduction*: when it is known that a failure in an equipment A triggers a failure also in the equipment B then the latter alarm is reduced, with the effect that only A, the root cause of the FS, is shown;
- *multiplicity reduction* when there is a great number of alarms of the same type[10] then these alarms are reduced and a new alarm is shown effectively reducing the number of alarms shown in the client GUI.

Four priority levels (0 to 3) are foreseen for the gravity of the alarms. Different priority levels are shown with different colors in the operator GUI.

3.5 The client tier

The client tier consists of java applications that consume the data published by the business tier. The client connects to the business tier by means of the Laser-console API . The business tier supports both login and configuration facilities[11].

Once connected, the clients can access services of the business tier by means of the laser-client API .The communication between the alarm service and the client applications happens through JMS whose implementation is based on Notification Channels (NC).

The alarms may be subdivided in categories and to each category corresponds one and only one NC. When the alarm service receives an alarm, it retrieves the name of its category together with other information for the operator from the database. The alarm service checks the alarm against its knowledge base and the state of other alarms applying the reduction rules and finally sends the alarm to the application of the client tier.

Three GUIs developed at CERN with Netbeans are part of the client tier: the definition console, the alarm console and the admin console. The definition console and the admin console allow the user to define alarms, sources and categories as well as create accounts and configurations for the operators of the alarm console. In this version of the AS the only available GUI is the alarm console but it is not used since ACS is providing a swing based alarmPanel. The source of the original CERN GUI, ported to ACS, is still available in SVN.

The CERN Netbeans Alarm console shows the alarms to the operators: when the operator starts the GUI, he has to log into the system, the GUI then loads his specific configuration and connects to the alarm service. In the configuration it is possible to select the categories of the alarms to show to the operator, showing him only the alarms relevant for his area of interest. In the configuration it is also possible to define whether the operator is interested in receiving all the alarms or only the reduced alarms. An apposite configuration panel allows the user to change its configuration at run-time.

The alarmPanel is an operator GUI developed in java swing for ACS to replace the Alarm Console. It shows alarms from all the categories and can be run as a stand alone java

[10] The MR is not activated by sending several times the same ACTIVE alarm. The MR is activated when a set of *different* alarms are all active at a certain instant of time and their number is greater then a given threshold. The alarms that must be counted as part of a MR is defined in the CDB.

[11] In the actual implementation the only existing user is test and the login is done automatically.

application or as a OMC plugin. After connecting to all the categories, the alarmPanel presents a view of the active and inactive alarms by means of a table.

4 CDB configuration

Each alarm must be defined in the CDB in order to be recognized and managed by the ASC. It is possible to define default alarms and categories to shorten the effort of setting up the CDB. However you should keep in mind that the main purpose of the alarm system is to present to the operators detailed information abut an alarm in order he/she can fix the problem very soon. Using default definitions allows to have a shorter CDB but generic informations are probably not enough to present the operators all they need to fix a problem. Therefore we suggest to use default values as less as possible and paying attention that the informations presented by default alarms fit with the real situation.

The definition of each alarm is very important because it contains both the information for the alarm system engine and those for the client tier applications that should be complete and clear enough to show to the operators all the information they would need to fix an abnormal situation.

A misconfiguration of the CDB does not inhibit a source from publishing an alarm as can be easily verified running the alarmSourcePanel or looking at the logs. A misconfiguration instead does not allow the ASC to send alarms to operators because there is no way for the service to know on which category an undefined alarm must be published.

The definition of the alarms is entirely contained in the `Alarms` folder in the CDB having the following structure:

```
CDB
      ...
      Alarms
            Administrative
                  AlarmSystemConfiguration
                  Categories
                  ReductionDefinitions
            AlarmDefinitions
                  FaultFamily_1
                  FaultFamily_2
                  ...
                  FaultFamily_n
```

The schemes defining the content of XML files are in the `acsalarmidl` module. In `ws/config/CDB/schemas` you can find the following files:

- `acsalarm-categories.xsd`: is the schema for the XML file into `Categories` folder
- `acsalarm-fault-family.xsd`: contains the definition for XML files called `FaultFamily_x` in the example upon

- `AcsAlarmSystem.xsd`: is used by `AlarmSystemConfiguration` and `ReductionDefinitions`

We describe the content of the `CDB/Alarms` folder in the same order the developers should follow to populate the database.

The `Alarms` folder must exist in order to use the CERN alarm system. `Alarm/Administrative/AlarmSystemConfiguration/AlarmSystemConfiguration.xml` must also exist and have the "implementation" property set to CERN. If one of the previous conditions is missing, ACS will not use the CERN AS. Note that ACS log a message with the alarm system in use to help debugging.

During activation, the ASC logs a large number of debug messages for every piece of information it reads from the CDB: `jlog` can be of a great help while setting up the CDB for the alarms[12].

`Alarms` is composed of two folders:

- `Administrative` that contains all the administrative informations like the definition of the reduction rules and the categories to group the alarms;
- `AlarmDefinitions` with the definition of all the alarms.

Each time a new alarm is created, its definition must be added in `AlarmDefinitions`. The alarm system administrator deals with the `Administrative` part of the CDB defining how to group alarms into categories and how reduce/mask alarms. The administrator defines how and to whom the alarms will be shown.

4.1 AlarmDefinitions

The first step while creating alarms is the definition of the triplets. As we said before the FF is usually the IDL of the components while the FM is the name of a specific component implementing such interface. The FC represents the specific alarm sent by such a component. This definition includes dynamic components whose name (and therefore the FM) is known only at run time.

Referring to the `demo` example we have PS, Antenna, Mount and MF idl interfaces and a bounce of components implementing such interfaces.

While creating alarms for a component, the developer must create a folder with the name of the fault family (i.e. the IDL of the component), having an XML file with the same name inside:

```
CDB
    Alarms
        AlarmDefinitions
            Mount
                Mount.xml
```

It means that we have only one folder for each FF, i.e. one folder for each IDL definition. Inside such folder, there is only one XML file containing the definition of all the fault

[12] To use jlog for this purpose you should remember to set the log level to *trace* and discard level to *none*. Filtering by source name (AlarmService) is also suggested.

codes (i.e. the type of alarms sent by the component implementing such interface) and the definition of all the fault members (i.e. the component names, including dynamic components).

The definition of the alarms sent by components implementing the Mount IDL interface is described by `Mount.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<fault-family name="Mount" ...>
  <alarm-source>ALARM_SYSTEM_SOURCES</alarm-source>
  <help-url>http://tempuri.org</help-url>
  <contact name="Ale"/>
  <fault-code value="1">
    <priority>1</priority>
    <problem-description>Mount test</problem-description>
  </fault-code>
  <fault-member-default>
  </fault-member-default>
  <fault-member name="ALARM_SOURCE_MOUNT">
  </fault-member>
</fault-family>
```

The document type is `fault-family` having the attribute `name` with the name of the fault family defined by the file[13], Mount in our example.

The following table describes the tags of the XML.

XML tag name	Tag meaning
`alarm-source`	The name of the NC in use by the sources while sending alarms to the ACS for the Mount components. This field is fixed and therefore can't be changed by the developer since in current version of the alarm system only one channel is supported for all the sources[14].
`help-url`	The URL where the operators find detailed information about the alarms published by the components of the family. The web page should contain at least a detailed description of all the alarms and the way to fix problem.
`contact`	Defines the person to contact in case the operator needs more information to fix the situation. When an alarm is raised for this family, the contact person should always be informed of the malfunction. This tag contains three other tags ● `name` ● `email` ● `gsm`
`fault-code`	The XML contains a non-empty array of fault codes. They are the FCs of the triplets of the family. All the possible FCs must have an entry of this type. The attribute `value`, is the number of this FC and is mandatory. fault-code contains the following tags: ● `priority` is an integer in the interval [0,3]

[13] It is the FF of the triplet.
[14] In future versions of the alarm system, it will be possible to have more then one channel for different sources. We therefore preferred to force the definition of the source channel in the XML.

	● cause: the cause of the problem ● action: the action to fix the problem ● consequence: possible consequence of the problem ● problem-description: a short description of the problem
fault-member-default	The definition of the fault memmebr to use when a fault-member is not defined for a given triplet. fault-member-default contains an optional tag location. When this tag is present and the alarm service receives a triplet for this FF whose FM is not defined then it generates automatically the data for such a triplet.
fault-member	The file contains an array of fault-member each of which represents a FM of the triplet. The tag has a mandatory attribute name that is the FM of the triplet (i.e. the name of the component sending the alarm). It is possible to define a location as element of this tag.

fault-member-default is the default member definition used by the ASC when it receives a triplet whose FM is not describet by any fault-member entry of the family. When a triplet is received, the ASC reads the default definition to fill the fields of the alarm description, leaving the FM name as it is in the triplet. It is possible to define a location for a fault-member-default but it is optional because sometimes it is not possible to define a location (like for example for dynamic components).

The fault-member-default is normally used for dynamic components whose names (and therefore their FMs) are unknown before run-time. Another situation where the definition of a deafult member can be very useful is the definition of BACI properties as described in chapter 7.6.

The demo example shows the usage of fault-member-default for the component ALARM_SOURCE_MOUNT_CPP.

A location is composed of four string elements:

- building: the building of the failing equipment
- floor: the floor in he building
- room: the room number or name
- mnemonic: a mnemonic to quickly recall the position of the equipment
- position: the exact position in the room

Reassuming, the definition of alarms is done having the fault family in mind. The developers must define one folder for each fault family containing one XML file with the definitions of all the members and the codes of the alarms sent by the members of the family. The fault codes describe the type of the alarms and there must be at least one fault code. Fault members represent the entities that can produce alarms but in this case it is possible to define a default member avoiding the repetition of the same definition several time in the file.

4.2 Administrative

This part of the CDB defines how alarms are presented to the operators resembling the process of creating views by a database administrator.

As we said before, the developer describes the meaning of each alarm by filling the `AlarmDefinitions` part of the CDB. Whenever the ASC receives a triplet from a source, it reads the definition of the alarm from this part of the CDB.

After reading the alarm definition, the ASC starts processing the alarm checking if some the reduction rules is applicable and eventually publishes the alarm in one or more categories where the clients listen to alarms. If an alarm is not associated to any of the existing categories then it will not be published and therefore will not be visible by ASC clients like the operator GUIs..

Categories represent the mechanism used by the administrator to create different views of the alarms: he/she decides in which categories an alarm will be published. Users listens to alarms from the categories they are allowed to connect[15].

4.3 AlarmSystemConfiguration

`AlarmSystemConfiguration` contains `AlarmSystemConfiguration.xml`:

```
AlarmSystemConfiguration
        AlarmSystemConfiguration.xml
```

This XML file contains a list of properties common to the whole AS. At the present there is only one property that can be defined: Implementation.

Property name	*Property values*
`Implementation`	● CERN if the AS uses the CERN implementation ● ACS: if the AS uses the ACS simple implementation i.e. the sources log a message instead of sending messages to the ASC If this property is not found or it has a wrong value, then the ACS implementation of the alarm system is used.

ACS comes with two implementations of the AS: ACS and CERN. The ACS implementation logs a message for each published alarm. The message basically consists of the triplet and the status of the alarm. It is possible to browse the alarms with jlog , or the loggingClient.

CERN implementation is the subject of this document. The following steps are needed to activate the CERN implementation of the AS:

- define the `Alarms` branch of the CDB as described in this chapter
- set the `Implementation` property of `AlarmSystemConfiguration.xml` to CERN

You can switch from one implementation to the other simply changing the `Implementation` property of the CDB i.e. you do not need to remove the Alarms branch if it is present.

[15] The CERN operator GUI allows the user to select the categories they want to listen to alarms from. The alarmPanel instead automatically connects to all available categories.

The format of the XML file is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<alarm-system-configuration
  xmlns="urn:schemas-cosylab-com:acsalarm-alarmservice:1.0
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
       <configuration-property        name="Implementation">CERN</configuration-
property>
</alarm-system-configuration>
```

Each property is defined with the tag `configuration_property` and the name of the property is in the attribute `name`.

4.4 Categories

The `Categories` folder contains `Categories.xml`:

```
Categories
       Categories.xml
```

Here are defined all the categories (and therefore all the notification channels) into which the alarms are grouped for publishing by the ASC. The categories are used by the applications of the client tier. For example each user of the operator GUI has a configuration with the categories he/she is authorized/interested in.

This is the configuration of categories from the demo example:

```
<categories
       xmlns="urn:schemas-cosylab-com:acsalarm-categories:1.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <category is-default="true" path="CATEGORY1">
    <description>Test category 1</description>
    <alarms>
      <FaultFamily>PIPELINE</FaultFamily>
    </alarms>
  </category>
</categories>
```

`Categories.xml` must have `categories` has document tag. the document is composed of an array of `category`. Each category is as described in the following table.

***`category`* elements**	***Values***
`description`	The description of the category
`alarms`	The fault families of the alarms that must be associated to this category. It is composed of an array of `FaultFamily` elements each of which contains the FF of the triplets of the alarms to be published in the category

`category` has 2 mandatory attributes: `path` and `is-default`. `path` represents the name of the NC used by the category. In the example upon, you can see that CATEGORY1 is used. Meaningful names should be used for the categories because the users selects the categories understanding the types of the alarms that the categories will publish mainly by reading their names.

The boolean attribute `is-default` is also mandatory. In the definitions of the categories one category can be set as default. It means that whenever an alarm is received and it is not associated to any category (i.e. its FF is not present in any of the `alarms` element of the categories) then the default category will be used to publish that alarm. This avoid duplicating information and keeping the definition of categories short.

As we said before, if the ASC can't associate an alarm to one category then the alarm will not be published to any category and therefore will not by visible by client applications like the operator GUIs. Forgetting to associate an alarm to a category is a common mistake: the alarm is sent by a source but lost inside the ASC. By checking the logs you can easily understand what's going on because the ASC will warn that a log will not be published in any category[16]. The definition of a default category avoids losing alarms not associated to any categories, of course.

As we said for `fault-member-default`, the usage of a default category is very helpful avoiding mistakes and taking the definition of the CDB short. But on the other hand it should avoided because it breaks the concept of categories and the logical association of alarms to categories that at the end of the process means the creation of views for different types of alarm system users.

In the example upon, you can see that alarms for the FF PIPELINE are published in CATEGORY1. But CATEGORY1 is also the default category where all the alarms with a unknown FF are published.

There are no rules to associate alarms to categories. As a guideline we suggest that each ALMA subsystem publishes alarms in one category with the name resembling the name of the subsystem. It will make easier identifying the entity generating the alarm and the responsible subsystem.

4.5 ReductionDefinitions

The `ReductionDefinitions` folder contains `ReductionDefinitions.xml`:

```
ReductionDefinitions
        ReductionDefinitions.xml
```

The document type of the XML is r`eduction-definitions` and contains 2 sections, `links-to-create` with the reduction-link and `thresholds` with the thresholds for the multiplicity reduction rules:

```
<reduction-definitions ...>
        <links-to-create>
                <reduction link...>
```

[16] The ASC can only log a messages because the lack of association of an alarm to one category can be a desired behavior. On the other hand it can also be a mistake of the administrator and a presence of a log messages helps fixing the problem.

```
                ...
                </reduction-link>
                ....
        </links-to-create>
        <thresholds>
                <threshold>
                ...
                </threshold>
                ...
        </thresholds>
```

The configuration file describes the reduction rules (RR) representing the knowledge base used by the ASC to mask/hide alarms to the operators.

As we said before, there are two kinds of reduction rules, node and multiplicity. Each rule joins two alarms identified by their triplet. Each RR has a type, `NODE` or `MULTIPLICITY`.

The multiplicity reduction (MR) is used to reduce the number of alarms of the same kind presented to the operator. When the ASC receives n alarms of the same kind and n is greater then a given threshold then the ASC masks all the alarms and produce a new alarm for the user. It is important to remember that the alarm shown to the operator is a completely new alarm generated by the ASC and not by a source. The new alarm generated by the ASC must be defined in the CDB in order to present to the operator the right information.

As we said before, the MR is not activated by sending the same active alarm n times. It is instead activated when n alarms are active at the same time and such alarms are all part of the MR described in the database.

The node reduction (NR) aims to mask an alarm when it is known that a failure in a equipment is always triggered by the failure of another one. In this case the ASC shows the root cause of the problem, masking the other alarm.

Each RR joins 2 alarms defining their correlation. The ASC in turns joins more RR building a chain of reduction that culminates showing only the root cause of a cascade of alarms.

This is an example of an NR:

```
<reduction-link type="NODE">
        <parent>
                <alarm-definition fault-family="PS"
                fault-member="ALARM_SOURCE_PS" fault-code="1"/>
        </parent>
        <child>
                <alarm-definition fault-family="Mount"
                fault-member="ALARM_SOURCE_MOUNT" fault-code="1"/>
        </child>
</reduction-link>
```

As you can see, the type of the reduction is represented by the `type` attribute of the `reduction-link` tag. The RR is composed of a parent alarm and a child alarm. The meaning of the RR is that every time there is an alarm of type `<PS, ALARM_SOURCE_PS,1>` then there is also an alarm of type `<Mount,`

ALARM_SOURCE_MOUNT,1> (i.e. whenever there is a failure of type 1 of the ALARM_SOURCE_PS then it triggers a failure of ALARM_SOURCE_MOUNT).

If the PS fails, then the MOUNT fails but the ASC knows that the failure of the MOUNT is triggered by the PS failure so it masks the second alarm: the operator sees only the alarm of the PS that is the root cause of this chain of failure.

Let's suppose that the failure of the PS, triggers the failure of the MOUNT that in turn triggers the failure of the ANTENNA. In this case the root cause of the chain of events is again the failure of the PS and we have to model the knowledge base of the ASC with this new correlation. In this case we only need to add the following NR:

```
<reduction-link type="NODE">
       <parent>
              <alarm-definition fault-family="Mount"
              fault-member="ALARM_SOURCE_MOUNT" fault-code="1"/>
       </parent>
       <child>
              <alarm-definition fault-family="Antenna"
              fault-member="ALARM_SOURCE_ANTENNA" fault-code="1"/>
       </child>
</reduction-link>
```

The case of the MR is analogous: the type of the RR must be set to MULTIPLICITY instead of NODE. The parent alarm is the alarm generated by the ASC when the number of active alarms is greater then threshold. The child alarm is the alarm generated by a component.

The thresholds section of the XML defines the thresholds for the MR. Each threshold is defined by an integer, the threshold, and the triplet of the alarm generated by the ASC (i.e. the triplet of the parent in the MR).

The alarms that are counted as part of a MR are all the alarms defined in a MR entry having the same alarm (generated by the ASC) as parent.

In the following example, you can see two alarms that are part of the same MR: <MF, MULTIPLE_MF_FAILURES, 0> and <MF, MULTIPLE_MF_FAILURES, 1>. When both of them are active (remember that the threshold is 2), the ASC generates an alarm of type <MF, ALARM_MULTIPLE_MF_FAILURES, 5>.

The following is an example of a definition of such a MR:

```
...
<reduction-link type="MULTIPLICITY">
       <parent>
              <alarm-definition fault-family="MF"
         fault-member="MULTIPLE_MF_FAILURES" fault-code="5"/>
       </parent>
       <child>
              <alarm-definition fault-family="MF"
          fault-member="ALARM_SOURCE_MF" fault-code="0"/>
       </child>
</reduction-link>
<reduction-link type="MULTIPLICITY">
       <parent>
              <alarm-definition fault-family="MF"
        fault-member="MULTIPLE_MF_FAILURES" fault-code="5"/>
       </parent>
```

```
	<child>
		<alarm-definition fault-family="AlarmSource"
	 fault-member="ALARM_SOURCE_MF" fault-code="1"/>
	</child>
</reduction-link>
....
<thresholds>
	<threshold value="2">
		<alarm-definition
			fault-family="MF"
			fault-member="MULTIPLE_MF_FAILURES"
			fault-code="1"/>
	</threshold>
	...
</thresholds>
```

The make the MR example short and readable, we have published alarms having the same FFs and FMs but with different FCs. While defining MR rules you can define triplets of any kind in the child elements.

The triplet generated by the ASC can also be of any kind but it needs to be defined in the CDB in order to be visible to the operators (i.e. its definition must be present in `AlarmDefinition`, and its family must be associated to a category in `Categories.xml` as you can see browsing the CDB of the demo example).

The alarms that are not part of any RR are not masked by the ASC and they will appear in the operator GUI.

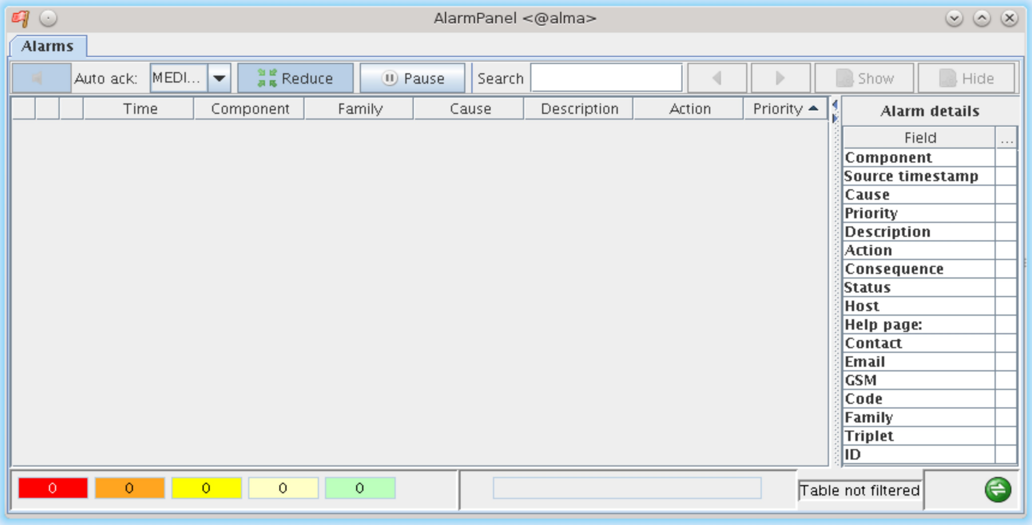
There are no defaults available for the reduction rules definitions: the application of such rules make some alarms invisible in the operator GUIs and we do not want this happens without the explicit control of the administrator.

5 The alarm GUI

ACS provide a java SWING GUI showing the alarms in a table. Such a GUI runs as OMC plugin and as a stand alone application by launching the `alarmPanel` command.

When the panel connects to the ACS it gets the list of the active alarms from the ASC and shows them in the table. You can then start the panel at any time and have immediately visible the active alarms regardless if they have been issued before launching the panel. Each alarrm has a color depending on the priority of the alarm and its state. If the alarm is inactive, it is shown in green. If it is active, the color depends on its category: red for very high priority (priority 0), orange for high (priority 1), yellow for medium (priority 2) and light yellow for low (priority 3). Inactive alarms are displayed in green but not removed from the table by default. To remove an alarm the user has to press the right mouse button over an alarm and select the Acknowledge popup menu item. It is important to remark that active alarms can't be removed by the table.

The actual version of the panel, automatically connects to all the existing categories without asking the user for a confirmation.



Auto ack: MEDI...

Field	...
Component	
Source timestamp	
Cause	
Priority	
Description	
Action	
Consequence	
Status	
Host	
Help page:	
Contact	
Email	
GSM	
Code	
Family	
Triplet	
ID	

Alarm details

Table not filtered

Figure 1: The alarmPanel at startup

The panel is divided in four main areas:

- the toolbar in the upper side
- the alarm table in the left center side
- the details table in the right center side
- the status bar at the bottom

The left side of the status area shows 5 colored widgets with a number in the middle. Each widget shows a summary of the alarms in the table of a given priority, the number is the number of active alarms of that priority. An alarm has one of the four priorities and therefore there are four colored widgets from red, top priority to light yellow, lowest priority. The last widget, the green one, reports the number of inactive alarms in the table. Those widgets are refreshed approximately every 2 seconds independently of the flow of the alarms i.e. it may happen that the numbers shown in the widgets and the number of alarms in the table are misaligned for a short time.

The right side of the status bar shows an icon that is green when the panel is connected to the ASC and gray when it is trying to connect to the ACS.

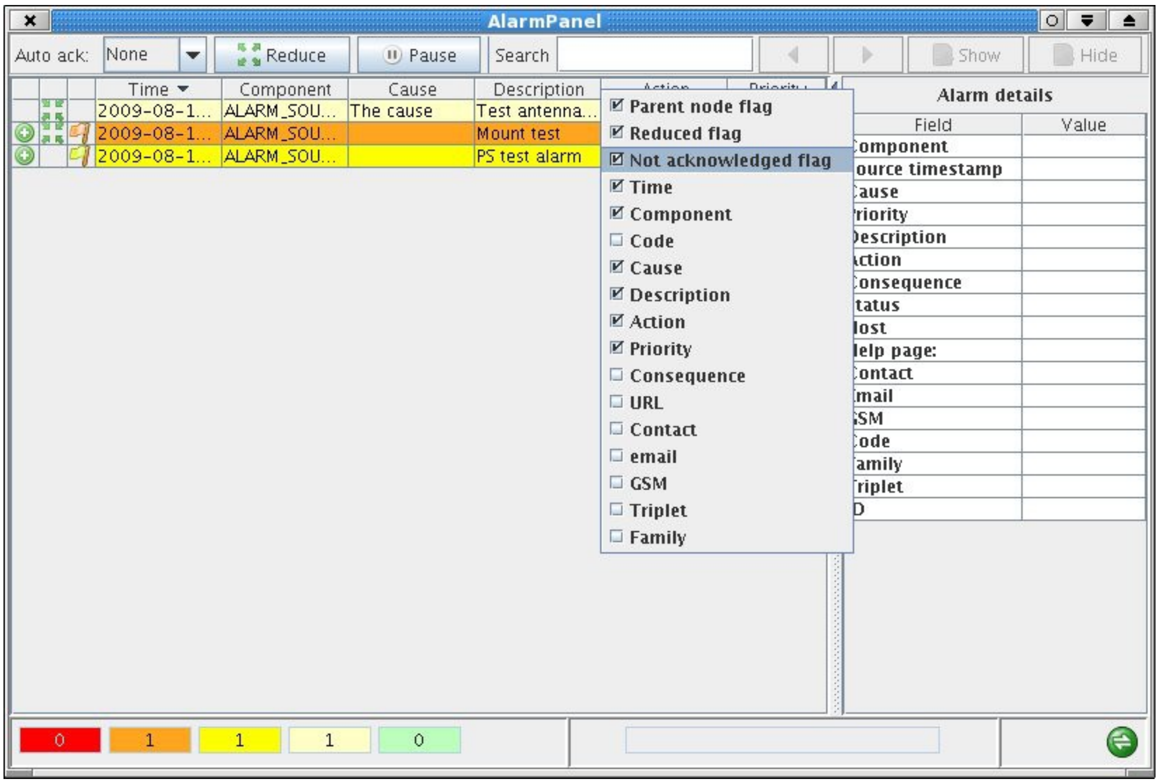
The table in the center left side of the GUI shows the alarms, one for each line of the table. The alarms are colored depending on their priority. Initially the alarms are sorted by putting a new alarm on top. If an alarm becomes inactive, it appears colored in green but does not change its position in the table. Vice versa, if an alarm is inactive and becomes active again, it is moved on top of the table.

The user can change the sorting of the table by pressing the column headers. The table sorts the alarm by 2 level remembering the two last selected columns. For example if you want to sort the table by component and then by priority you have to press the priority header and then the component header. Table columns can be moved by selecting their header and dragging to the desired position.

A popup menu is shown by pressing over a row with three items:

- Acknowledge: as said before it is used to remove inactive alarms from the table; this item is disabled if the alarm below the mouse pointer is active
- Save: save a text file containing the alarm below the mouse pointer
- To clipboard: copy the alarm in the clipboard

By default the time, the component, the priority, the description and the cause of each alarm are displayed. But it is possible to customize the fields of the alarm displayed by pressing the right mouse button over the table header and selecting the switches to activate/deactivate as shown by the following picture.



Time	Component	Cause	Description
2009-08-1...	ALARM_SOU...	The cause	Test antenna...
2009-08-1...	ALARM_SOU...		Mount test
2009-08-1...	ALARM_SOU...		PS test alarm

- [✓] Parent node flag
- [✓] Reduced flag
- [✓] Not acknowledged flag
- [✓] Time
- [✓] Component
- [ ] Code
- [✓] Cause
- [✓] Description
- [✓] Action
- [✓] Priority
- [ ] Consequence
- [ ] URL
- [ ] Contact
- [ ] email
- [ ] GSM
- [ ] Triplet
- [ ] Family

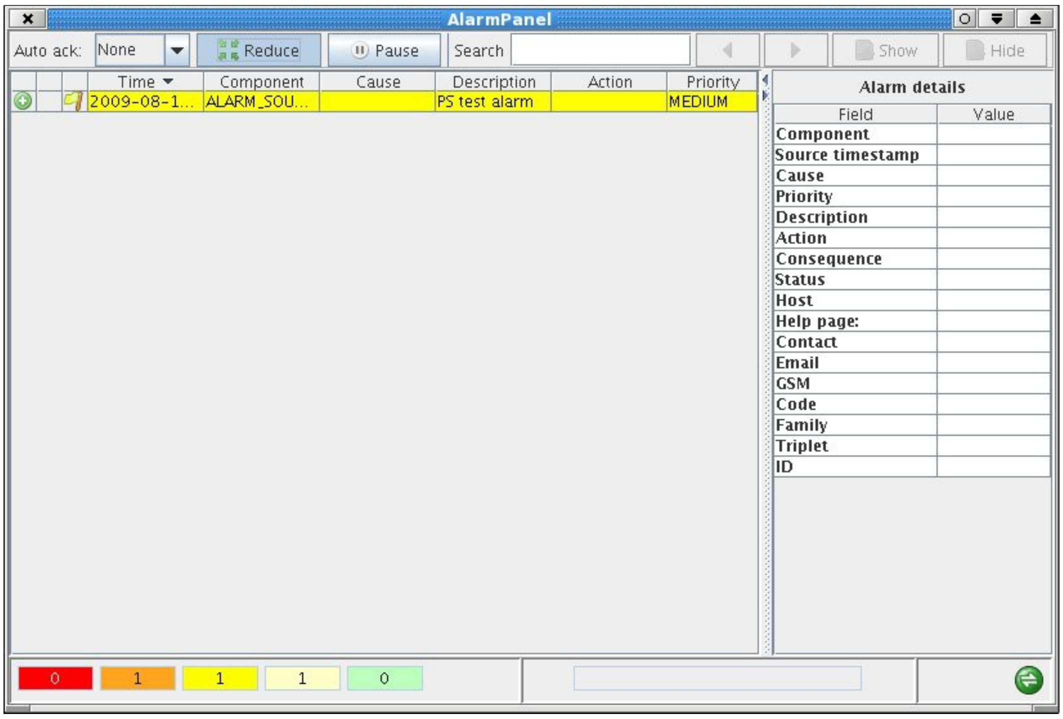
Figure 2: the table of alarms without reduction rules

The figure upon shows the alarms generated by the demo example without activating the reductuion rules. The first two columns shows is an alarm is involved in a reduction rule. The first column has a green icon with a plus if a reduction rul that alarm hides another alarm. The second columns shows a green icon with four little arrows pointing to the center if the alarm is hided by a reduction rule.

Note that an alarm can cause the hiding of another alarm and been hided at the same time.

If the reduction rules have been configured in the CDB, the alarm system marks the alarms involved in the reduction rules with a special set of flags. The GUI reads those flags and is able to hide the uneeded alarms showing only the root cause of a problem. This is very important because can drammatically reduce the number of alarms in the table and enhance the readability of the table as well as the understanding of the problem by the operators.

In the previous example in fact, if we activate the reduction rules we can immediatly see that the cause of the chain of alarms was a failure in the power supply so we know where to look to fix the failure.



Auto ack: None

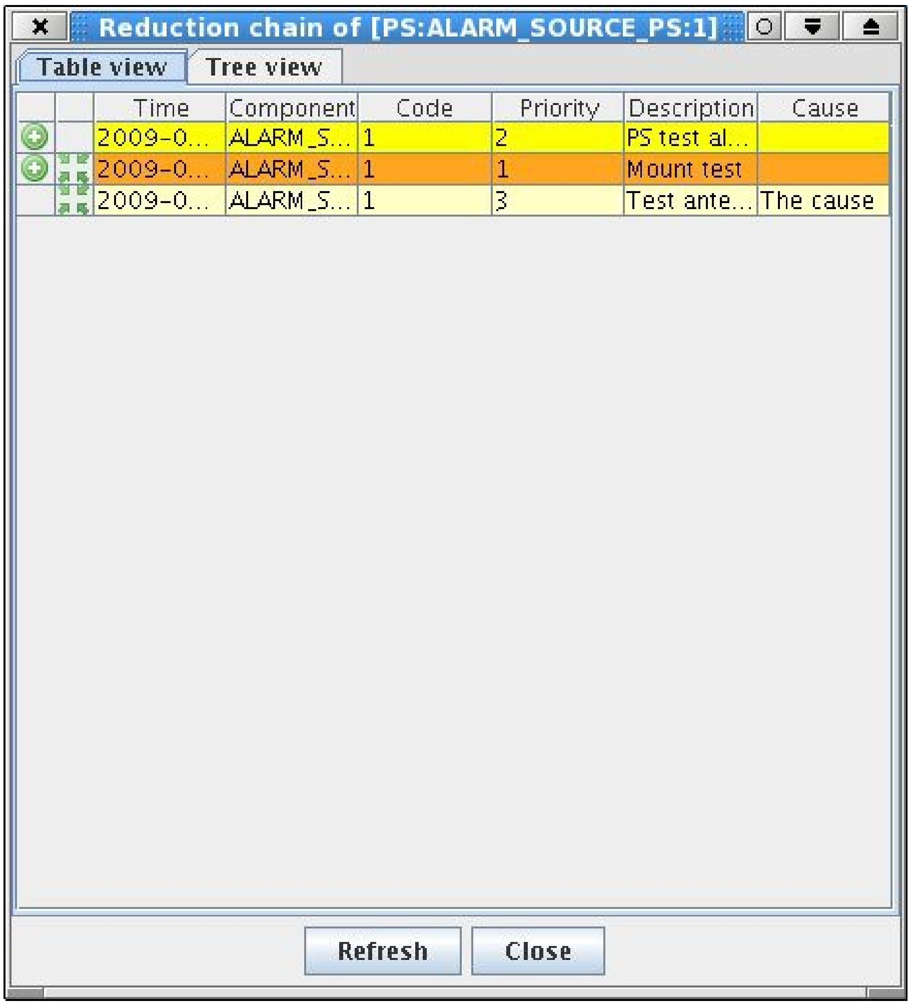
Time	Component	Cause	Description	Action	Priority
2009-08-1…	ALARM_SOU…		PS test alarm		MEDIUM

Field	Value
Component	
Source timestamp	
Cause	
Priority	
Description	
Action	
Consequence	
Status	
Host	
Help page:	
Contact	
Email	
GSM	
Code	
Family	
Triplet	
ID	

Figure 3: the panel with the reduction rules activated

It is often very useful to investigate which alarms are involved in a reduction rule. For example if we have only one alarm visible like in the previous figure it could be useful to see the full chain of alarms involved by that reduction rule. It can be done by moving the mouse pointer over the alarm of the table, pressing the right mouse button and select the *show reduction chain* item of the popup menu.

The reduction chain of an alarm is shown by means of a dialog that shows the alarm in a flat view or in a tree view better representing the relationship between involved alarms.



		Time	Component	Code	Priority	Description	Cause
⊕		2009-0...	ALARM_S...	1	2	PS test al...	
⊕		2009-0...	ALARM_S...	1	1	Mount test	
		2009-0...	ALARM_S...	1	3	Test ante...	The cause

Figure 4: the reduction chain of the PS alarm

The table with the reduction chain is aimed to show the relationship between alarm and is simpler than the table of alarms in the main window. For example it is not possible to select the columns to show but it is possible to sort the table by pressing over the column headers.

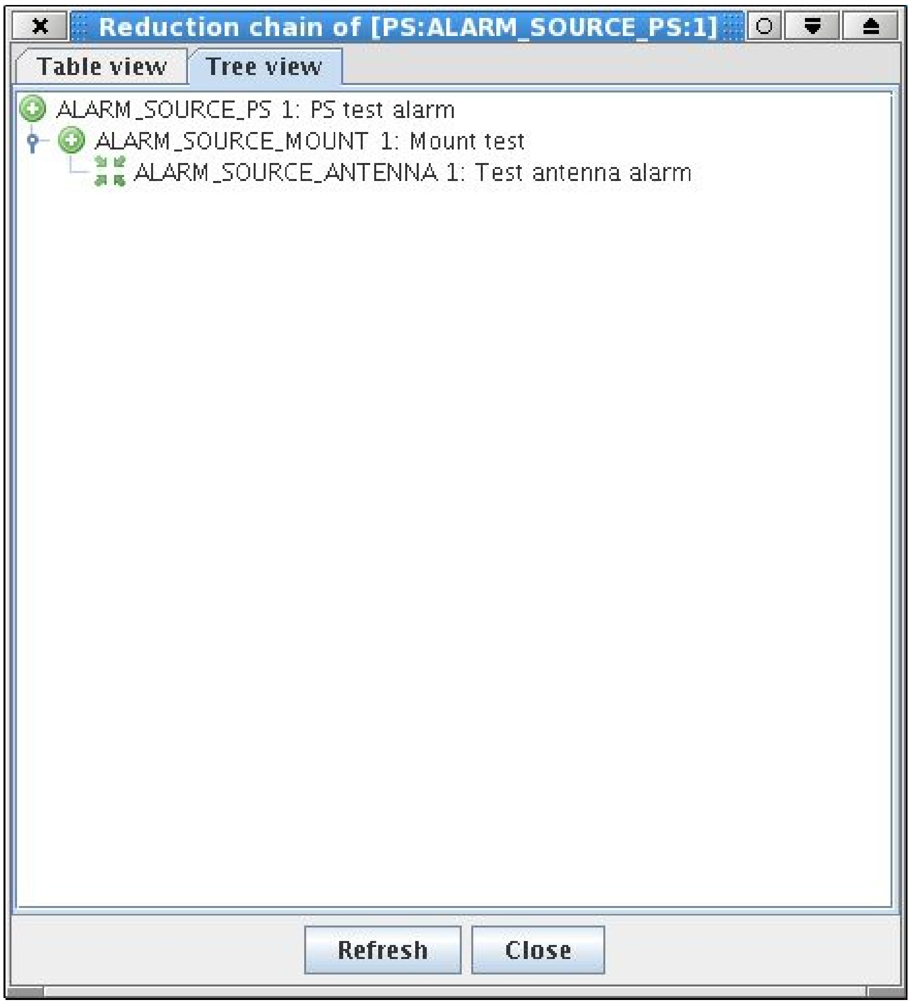

Reduction chain of [PS:ALARM_SOURCE_PS:1]
Table view
Tree view
ALARM_SOURCE_PS 1: PS test alarm
ALARM_SOURCE_MOUNT 1: Mount test
ALARM_SOURCE_ANTENNA 1: Test antenna alarm
Refresh
Close

Figure 5: the reduction chain of the PS alarm as a tree

The tree view is another way of viewing the same reduction chain that better shows who is hiding what. As a general suggestion, the tree view fits better for NR and the table view for MR.

When an alarm becomes inactive, i.e. the failure has been fixed, its color turns to green but the alarm is not removed from the main table of alarms until the user acknowledge the alarm buy pressing the right mouse button and selecting the apposite menu item.

The toolbar has a menu to select the auto-acknowledge level (disabled as default). By setting the auto-acknowledge level all the alarms having the selected priority, or a lower one, are transparently removed by the table when they become inactive. It is not possible to set the auto-acknowledge level for priority 0 alarms that must be explicitly acknowledged by the user by pressing the right mouse button. Auto acknowledgement allows to reduce the number of visible alarms but does not deletes the hiostory of what happened to the system and therefore should be avoided or used very carefully.

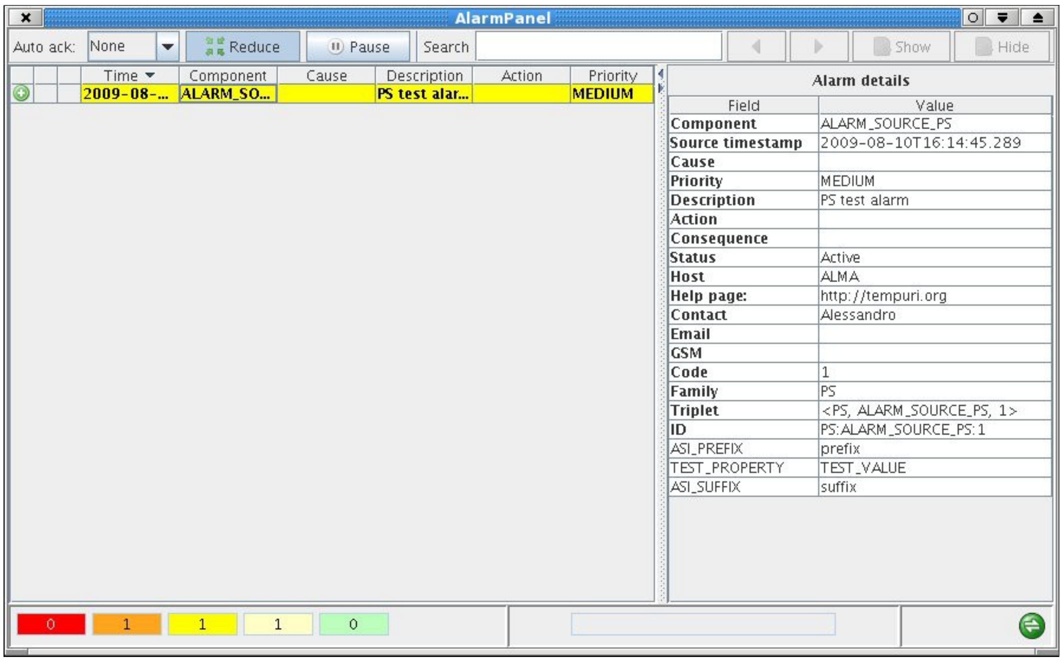
Note that at the present, the acknowledgemnt of an alarm is done locallly simply removing the entry from the table. In a furture release the acknowledgment will be propagated to all the opened panels.

The max number of alarms displayed by the table is limited to 20000. If the table is full and a new alarm arrives, the the oldest is removed.

The pause button freezes the refresh of the table of alarms enhancing the readability of the table.

5.1 Alarm details

Selecting an alarm in the table, causes the alarm details table in the right side to be filled with a detailed description of such an alarm.



			Time	Component	Cause	Description	Action	Priority
			2009-08-...	ALARM_SO...		PS test alar...		MEDIUM

Alarm details

Field	Value
Component	ALARM_SOURCE_PS
Source timestamp	2009-08-10T16:14:45.289
Cause	
Priority	MEDIUM
Description	PS test alarm
Action	
Consequence	
Status	Active
Host	ALMA
Help page:	http://tempuri.org
Contact	Alessandro
Email	
GSM	
Code	1
Family	PS
Triplet	<PS, ALARM_SOURCE_PS, 1>
ID	PS:ALARM_SOURCE_PS:1
ASI_PREFIX	prefix
TEST_PROPERTY	TEST_VALUE
ASI_SUFFIX	suffix

🟥 0	🟧 1	🟨 1	1	🟩 0

Figure 6: The details of an alarm

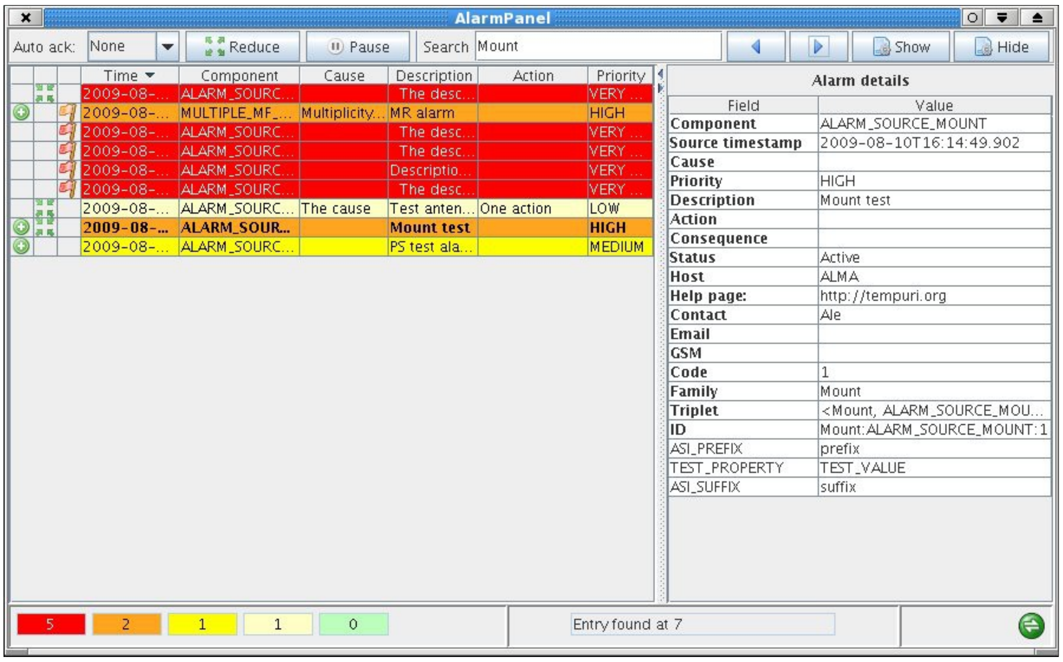
The alarm details table shows all the details of an alarm. The table has one entry for each field of the alarm. Those fileds are the same fields shown in the table of the alarms but the details table shows also the columns hidden in the alarms table.

As we said before, it is possibble to associate properties to an alarm: each properties has the form of a name and a value.
The details table shows such couple at the bottom of the details table like for example [TEST_PROPERTY, TEST_VALUE] in the picture above.

5.2 Quick search and filtering

The toolbar has a widget allowing to quickly search for alarms. It is composed of a text field and a set of buttons. The buttons are disabled unless the user inserts a string in the text field.

If the user writes a string in the text field, for example Mount, then all the buttons are enabled and such a string can be used to scan the table and search for an alarm: the button with the left arrow look for an alarm before the current select alarm while the right arrow look for an alarm after the current select entry. When an alarm is found, the roew of the entry appears in bold and the details of the alarm are shown in the righ table.


AlarmPanel

Auto ack: None | Reduce | Pause | Search Mount | Show | Hide

Time	Component	Cause	Description	Action	Priority
2009-08-…	ALARM_SOURC…		The desc…		VERY …
2009-08-…	MULTIPLE_MF_…	Multiplicity…	MR alarm		HIGH
2009-08-…	ALARM_SOURC…		The desc…		VERY …
2009-08-…	ALARM_SOURC…		The desc…		VERY …
2009-08-…	ALARM_SOURC…		Descriptio…		VERY …
2009-08-…	ALARM_SOURC…		The desc…		VERY …
2009-08-…	ALARM_SOURC…	The cause	Test anten…	One action	LOW
2009-08-…	**ALARM_SOUR…**		**Mount test**		**HIGH**
2009-08-…	ALARM_SOURC…		PS test ala…		MEDIUM

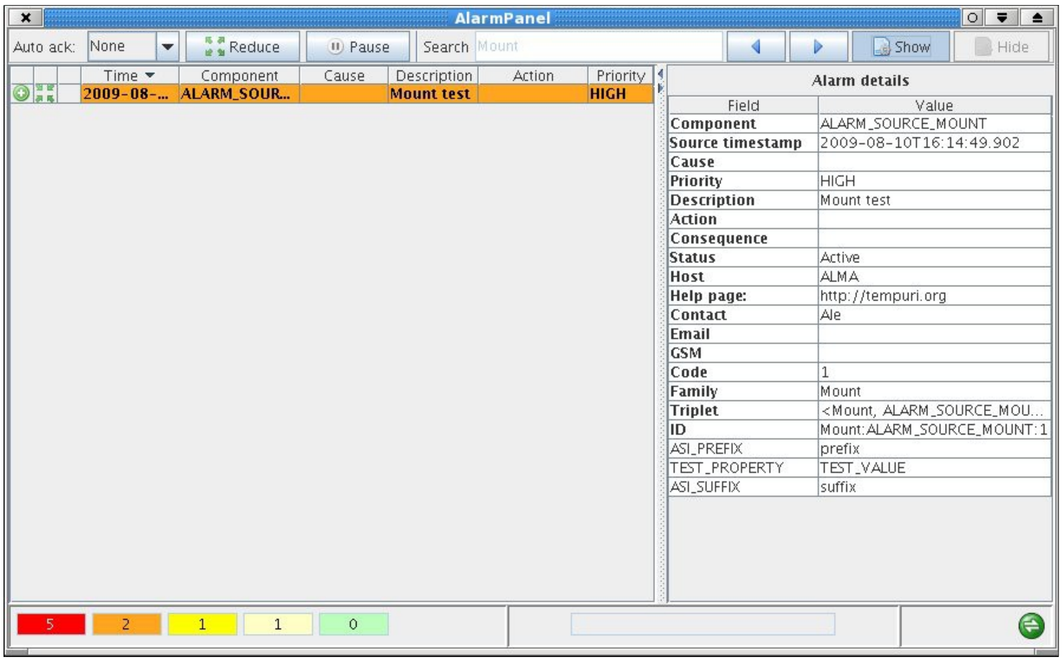
Alarm details

Field	Value
Component	ALARM_SOURCE_MOUNT
Source timestamp	2009-08-10T16:14:49.902
Cause	
Priority	HIGH
Description	Mount test
Action	
Consequence	
Status	Active
Host	ALMA
Help page:	http://tempuri.org
Contact	Ale
Email	
GSM	
Code	1
Family	Mount
Triplet	<Mount, ALARM_SOURCE_MOU…
ID	Mount:ALARM_SOURCE_MOUNT:1
ASI_PREFIX	prefix
TEST_PROPERTY	TEST_VALUE
ASI_SUFFIX	suffix

5 2 1 1 0 | Entry found at 7

Figure 7: The search for an alarm containing "Mount".

The two button, Show and Hide allow to quickly filter the table for the given string. By pressing the show button, the alarm table is filtered in order to show all and only the alarm containing Mount.


AlarmPanel

Auto ack: None | Reduce | Pause | Search Mount | Show | Hide

Time	Component	Cause	Description	Action	Priority
2009-08-…	**ALARM_SOUR…**		**Mount test**		**HIGH**

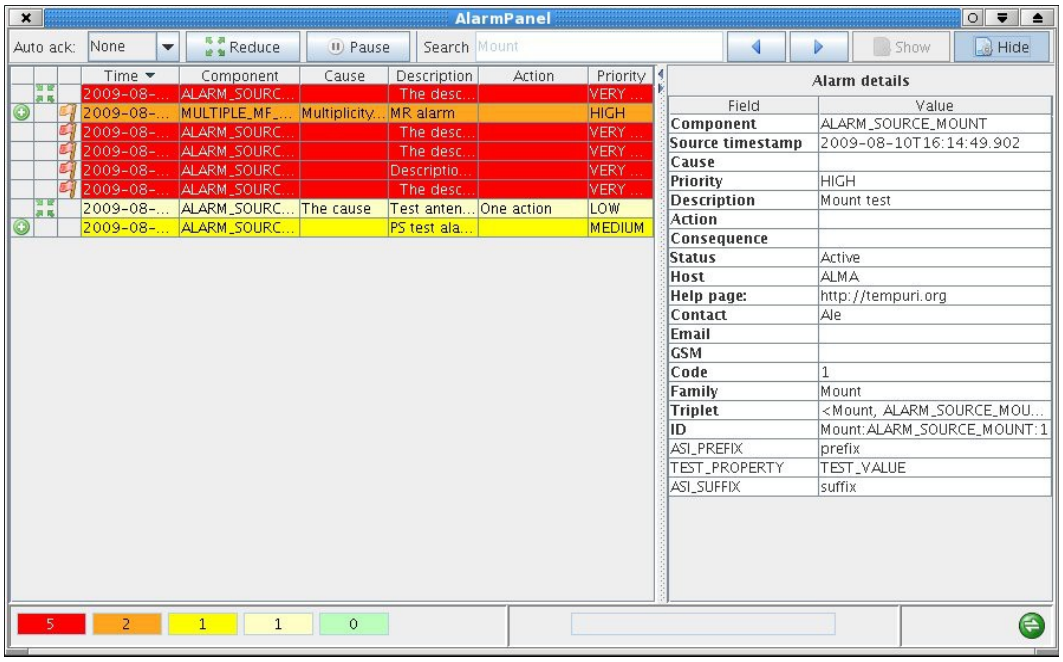
Alarm details

Field	Value
Component	ALARM_SOURCE_MOUNT
Source timestamp	2009-08-10T16:14:49.902
Cause	
Priority	HIGH
Description	Mount test
Action	
Consequence	
Status	Active
Host	ALMA
Help page:	http://tempuri.org
Contact	Ale
Email	
GSM	
Code	1
Family	Mount
Triplet	<Mount, ALARM_SOURCE_MOU…
ID	Mount:ALARM_SOURCE_MOUNT:1
ASI_PREFIX	prefix
TEST_PROPERTY	TEST_VALUE
ASI_SUFFIX	suffix

5 2 1 1 0

Figure 8: Filter in all the alarms of the table containing "Mount".

The Hide button works in the other way around: it filter the table by hiding all the alarms containing “Mount”.



Time	Component	Cause	Description	Action	Priority
2009-08-...	ALARM_SOURC...		The desc...		VERY ...
2009-08-...	MULTIPLE_MF_...	Multiplicity...	MR alarm		HIGH
2009-08-...	ALARM_SOURC...		The desc...		VERY ...
2009-08-...	ALARM_SOURC...		The desc...		VERY ...
2009-08-...	ALARM_SOURC...		Descriptio...		VERY ...
2009-08-...	ALARM_SOURC...		The desc...		VERY ...
2009-08-...	ALARM_SOURC...	The cause	Test anten...	One action	LOW
2009-08-...	ALARM_SOURC...		PS test ala...		MEDIUM

Alarm details

Field	Value
Component	ALARM_SOURCE_MOUNT
Source timestamp	2009-08-10T16:14:49.902
Cause	
Priority	HIGH
Description	Mount test
Action	
Consequence	
Status	Active
Host	ALMA
Help page:	http://tempuri.org
Contact	Ale
Email	
GSM	
Code	1
Family	Mount
Triplet	<Mount, ALARM_SOURCE_MOU...
ID	Mount:ALARM_SOURCE_MOUNT:1
ASI_PREFIX	prefix
TEST_PROPERTY	TEST_VALUE
ASI_SUFFIX	suffix

Figure 9: Filter out all the alarms containing "Mount".

Note that when filtering the table, it is not possible to change the string in the text filed: to change filtering or search for alarms the Hide or Show button must be pressed again.

6 The CERN operator GUI

The CERN alarm system provides a netbeans GUI to show the alarms. Unfortunately such a GUI does not run inside the OMC and therefore we implemented a SWING GUI, the alarmPanel. The netbeans GUI is still part of ACS distribution and can be run as a stand alone application.

The ACS swing panel is the default operator panel and is automaticcaly built by ACS. If you wish to try the CERN GUI, it must be manually built: it is in the module `ACS/LGPL/CommonSoftware/NetbeansACS`.

The CERN operator GUI is started with the `acsalarmgui` command: a script that reads the actual configuration and starts the netbeans application. For it to work well you need ACS correctly installed, and the ASC running.

`acsalarmgui` is a python script that accepts only one parameter, the corbaloc of the manager; it sets up all the parameters needed by the GUI before starting it. It looks for the manager reference from the command line, the MANAGER_REFERENCE environment variable and the local host. The ACS_INSTANCE is read from the environment variable or set to 0 if missing.

The script takes care of preparing the classpath, copying and installing the last version of the jars in the current directories as expected by netbeans. The packages are searched in the INTROOT, INTLIST and ACSROOT as usual.

The GUI is written in netbeans but a new SWING version is available at CERN and will be ported as soon as possible into ACS.

At the present, the user configurations are not stored on the database so each time the GUI starts, the user needs to setup the configuration. After login as test (the only available user), a configuration dialog is shown. The user should select the categories he wants to listen to. This is done by selecting the ROOT category and the adding all its subcategories. Another important setting is the activation of the Reduction in the behavior tab.

When finished setting up the configuration, the user has to approve the changes and finally close the configuration window.

At this point the GUI is connected to the AS, as shown by the green icon in the right bottom side of the window.

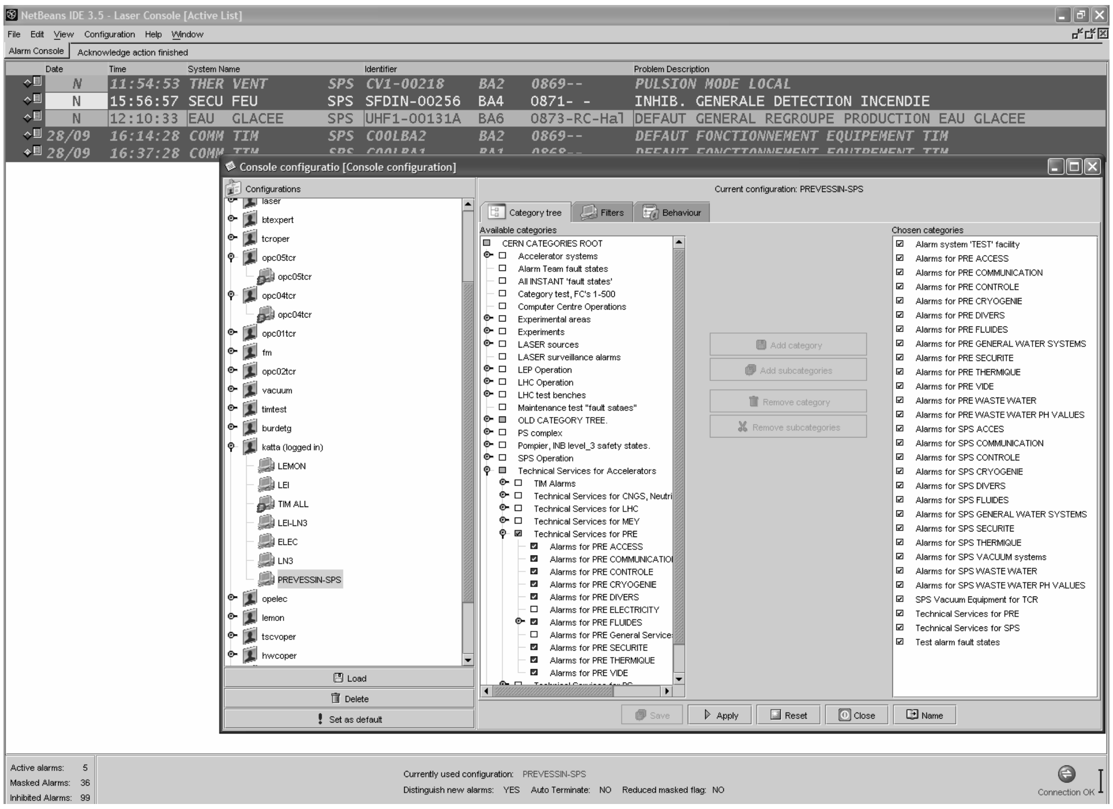
The main table in the center of the windows shows the incoming alarm.

Each new alarm is shown with a “N” in the left side of the line. When the user selects the alarm, the N disappear and is replaced by the date when the alarm was received. The purpose of the N is to show the user that an alarm is new and has not yet been seen (selected).

If the reduction is disabled, the table will show all the alarms published by the ASC. If the reduction is active, then only the root cause of a chain of alarms is shown and all the other alarms are hidden reducing drastically the amount of alarms shown in the table.

The alarms do not disappeared from the GUI even if the abnormal situation has been fixed. This is to force the user to explicitly acknowledge each alarm by pressing a right mouse button over the alarm.

We suggest to play a little bit with the GUI to understand its functioning and options. However, not all the functionalities of the original LASER GUI are available in this version. No further work is foreseen in this interface because all the efforts will be to adopt the new SWING version.



```
Date   Time      System Name      Identifier                     Problem Description
N      11:54:53  THER VENT   SPS  CV1-00218     BA2  0869--      PULSION MODE LOCAL
N      15:56:57  SECU FEU    SPS  SFDIN-00256   BA4  0871- -     INHIB. GENERALE DETECTION INCENDIE
N      12:10:33  EAU  GLACEE SPS  UHF1-00131A   BA6  0873-RC-Hal DEFAUT GENERAL REGROUPE PRODUCTION EAU GLACEE
28/09  16:14:28  COMM TIM    SPS  COOLBA2       BA2  0869--      DEFAUT FONCTIONNEMENT EQUIPEMENT TIM
```

Current configuration: PREVESSIN-SPS

Available categories:

- CERN CATEGORIES ROOT
  - ☐ Accelerator systems
  - ☐ Alarm Team fault states
  - ☐ All INSTANT 'fault states'
  - ☐ Category test, FC's 1-500
  - ☐ Computer Centre Operations
  - ☐ Experimental areas
  - ☐ Experiments
  - ☐ LASER sources
  - ☐ LASER surveillance alarms
  - ☐ LEP Operation
  - ☐ LHC Operation
  - ☐ LHC test benches
  - ☐ Maintenance test "fault sataes"
  - OLD CATEGORY TREE.
  - ☐ PS complex
  - ☐ Pompier, INB level_3 safety states.
  - ☐ SPS Operation
  - Technical Services for Accelerators
    - ☐ TIM Alarms
    - ☐ Technical Services for CNGS, Neutri
    - ☐ Technical Services for LHC
    - ☐ Technical Services for MEY
    - ☑ Technical Services for PRE
      - ☑ Alarms for PRE ACCESS
      - ☑ Alarms for PRE COMMUNICATIO
      - ☑ Alarms for PRE CONTROLE
      - ☑ Alarms for PRE CRYOGENIE
      - ☑ Alarms for PRE DIVERS
      - ☐ Alarms for PRE ELECTRICITY
      - ☑ Alarms for PRE FLUIDES
      - ☐ Alarms for PRE General Service
      - ☑ Alarms for PRE SECURITE
      - ☑ Alarms for PRE THERMIQUE
      - ☑ Alarms for PRE VIDE

Chosen categories:

- ☑ Alarm system 'TEST' facility
- ☑ Alarms for PRE ACCESS
- ☑ Alarms for PRE COMMUNICATION
- ☑ Alarms for PRE CONTROLE
- ☑ Alarms for PRE CRYOGENIE
- ☑ Alarms for PRE DIVERS
- ☑ Alarms for PRE FLUIDES
- ☑ Alarms for PRE GENERAL WATER SYSTEMS
- ☑ Alarms for PRE SECURITE
- ☑ Alarms for PRE THERMIQUE
- ☑ Alarms for PRE VIDE
- ☑ Alarms for PRE WASTE WATER
- ☑ Alarms for PRE WASTE WATER PH VALUES
- ☑ Alarms for SPS ACCES
- ☑ Alarms for SPS COMMUNICATION
- ☑ Alarms for SPS CONTROLE
- ☑ Alarms for SPS CRYOGENIE
- ☑ Alarms for SPS DIVERS
- ☑ Alarms for SPS FLUIDES
- ☑ Alarms for SPS GENERAL WATER SYSTEMS
- ☑ Alarms for SPS SECURITE
- ☑ Alarms for SPS THERMIQUE
- ☑ Alarms for SPS VACUUM systems
- ☑ Alarms for SPS WASTE WATER
- ☑ Alarms for SPS WASTE WATER PH VALUES
- ☑ SPS Vacuum Equipment for TCR
- ☑ Technical Services for PRE
- ☑ Technical Services for SPS
- ☑ Test alarm fault states

Active alarms: 5
Masked Alarms: 36
Inhibited Alarms: 99

Currently used configuration: PREVESSIN-SPS
Distinguish new alarms: YES Auto Terminate: NO Reduced masked flag: NO

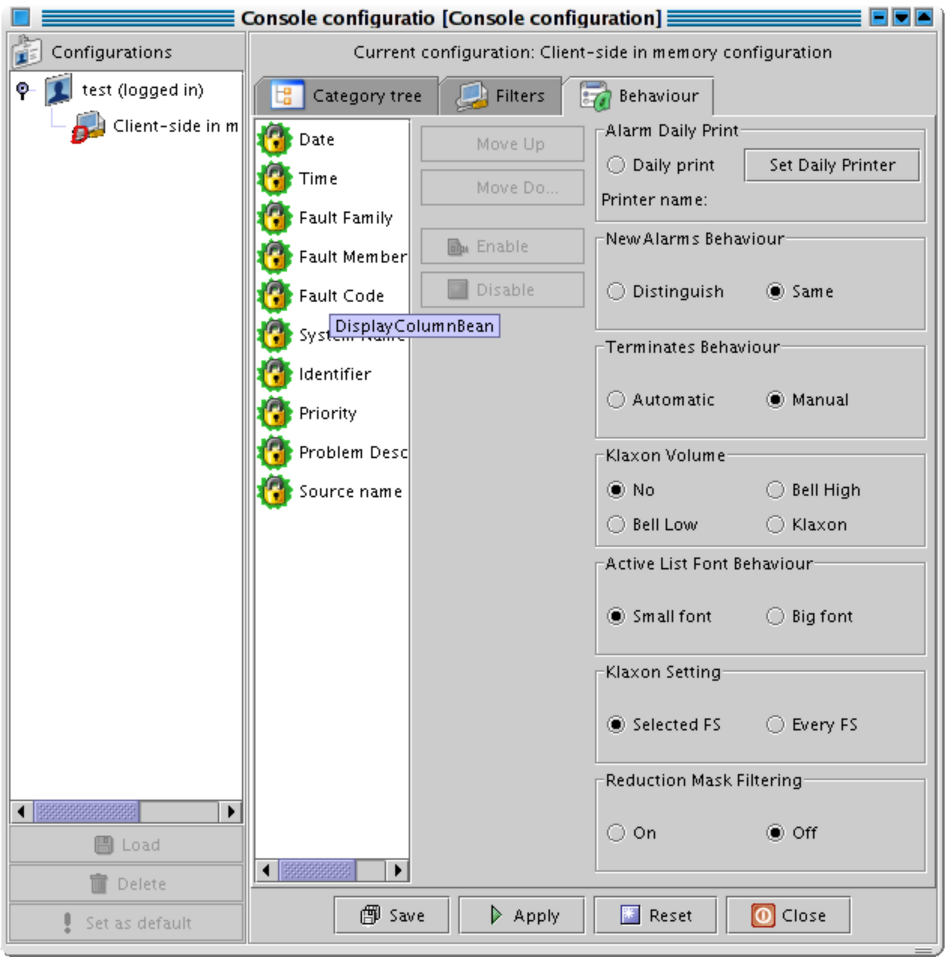
Figure 10: A snapshot of the operator GUI from the LHC control room.

The picture above shows a black and white screenshot of the GUI used at CERN. The inner window is the user configuration dialog.

The left side of the configuration window, shows the user configurations currently available. The setup is in the center of the window.

In the left side, all the available categories are shown. If you are using the test CDB, you should see the root category and two subcategories[17]: current and source. You should select the ROOT and press the add subcategories button: doing that you will see the selected categories shown in the right part of the window as shown in the picture below.

[17]It might be that you have to wait a little bit before seeing the current and source subcategories below root in the Availbale categories list.


Console configuratio [Console configuration]

Configurations

test (logged in)

Client-side in m

Current configuration: Client-side in memory configuration

Category tree | Filters | Behaviour

Date
Time
Fault Family
Fault Member
Fault Code
System Name
Identifier
Priority
Problem Desc
Source name

DisplayColumnBean

Alarm Daily Print
Daily print | Set Daily Printer
Printer name:

New Alarms Behaviour
Distinguish [✓ Same]

Terminates Behaviour
Automatic [✓ Manual]

Klaxon Volume
[✓ No] Bell High
Bell Low Klaxon

Active List Font Behaviour
[✓ Small font] Big font

Klaxon Setting
[✓ Selected FS] Every FS

Reduction Mask Filtering
On [✓ Off]

Figure 12: The console configuration dialog: behavior definition.

The behavior tab presents more options. The last one, reduction Mask Filtering enables and disables the reduction of alarms. This dialog is shown below.

Once the configuration is ready, press the apply button and then the close button.

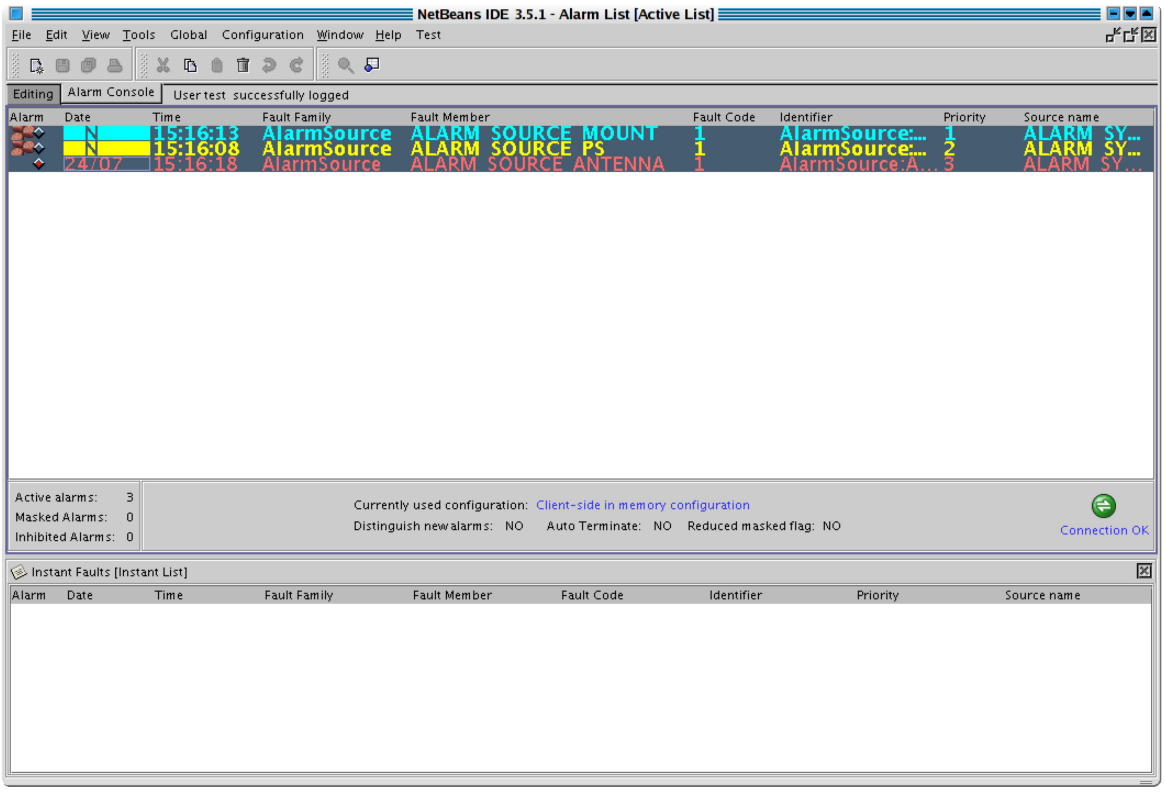
The main window is composed by a central table where all the alarms are shown, one per line. The picture below shows the GUI without reduction with all the alarms generated by the demo. The snapshot is taken with the test CDB and without reduction. If the reduction are active, the PS would be the only alarm shown because it is the root cause of the chain of alarms as defined in the CDB.

The alarms are shown with different colors depending by their priority.

The green icon in the right bottom shows the status of the connection with the ASC. The antenna and PS alarms are new in the meaning that the user has not yet selected them. The Antenna instead has been selected and the N has been replaced by the actual date.

The alarms are all active. If the user executes the terminate method of the PS, all the alarms become inactive and their color in the GUI switches to green. Inactive alarms are not removed from the table because the user must explicitly acknowledge the abnormal situation.

This is done by pressing the right mouse button of the alarm and selecting the Acknowledge item of the popup menu: a new dialog is shown where the user has to write his comment into. Only at this point the alarm is removed from the main window.


NetBeans IDE 3.5.1 - Alarm List [Active List]
File Edit View Tools Global Configuration Window Help Test
Editing
Alarm Console

User test successfully logged

Alarm	Date	Time	Fault Family	Fault Member	Fault Code	Identifier	Priority	Source name
	N	15:16:13	AlarmSource	ALARM_SOURCE_MOUNT	1	AlarmSource:...	1	ALARM_SY...
	N	15:16:08	AlarmSource	ALARM_SOURCE_PS	1	AlarmSource:...	2	ALARM_SY...
	24/07	15:16:18	AlarmSource	ALARM_SOURCE_ANTENNA	1	AlarmSource:A...	3	ALARM_SY...

Active alarms: 3
Masked Alarms: 0
Inhibited Alarms: 0

Currently used configuration: Client-side in memory configuration
Distinguish new alarms: NO Auto Terminate: NO Reduced masked flag: NO

Connection OK

Instant Faults [Instant List]

Alarm	Date	Time	Fault Family	Fault Member	Fault Code	Identifier	Priority	Source name

Figure 13: The main window of the operator GUI with the alarm generated by the demo and the reduction rules disabled.

As we said before, a new version of the GUI, written with java SWING library is ready at CERN and we do not plan to work further on this version. The look and the feel as well as the functioning of the two GUIs is very similar.

Not all the functionalities of the original GUI have been ported in ACS so it could happen that you try to do something not yet implemented or not working. We kindly suggest you to play with the GUI keeping in mind that every missing functionality or not working feature can't cause further malfunctioning in the system.

7 ACS implementation

ACS specific code has been developed to adapt CERN alarm system to ACS or to extend the basic CERN implementation to fit ALMA needs.

To better understand the following discussion, it is better to refer to the following figure.

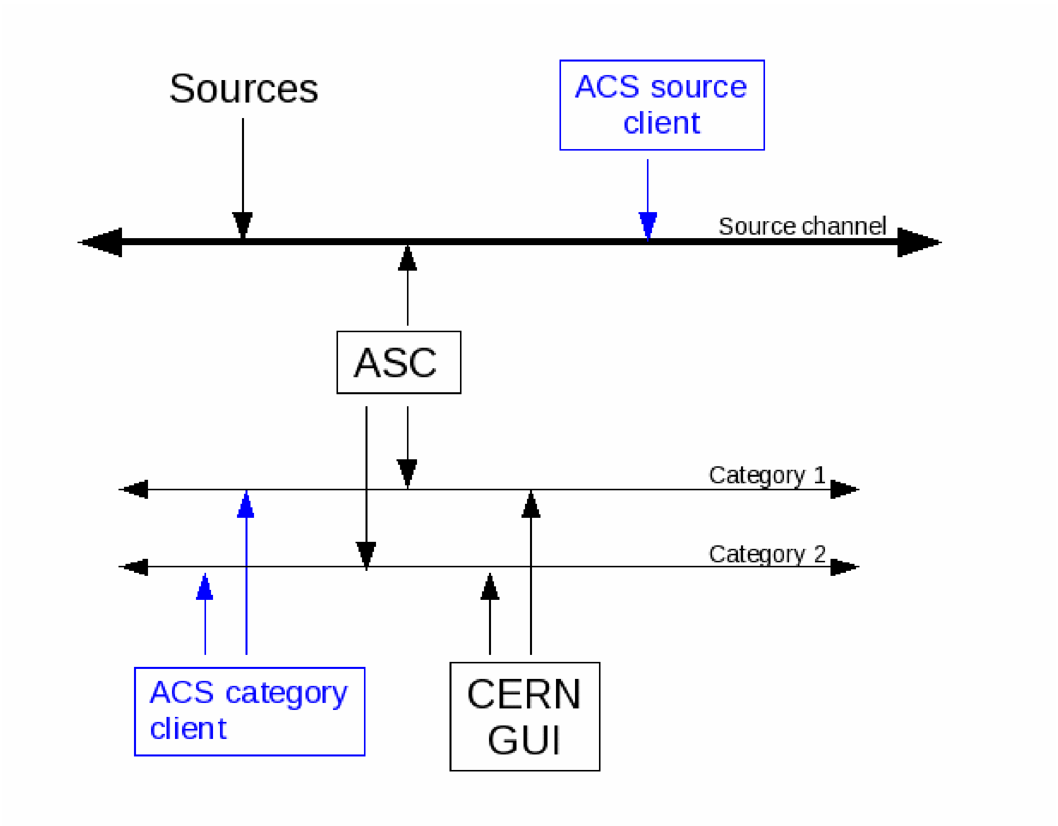


Figure 14: CERN architecture and ACS clients

Black lines and boxes represent the CERN part of the alarm system. The sources publishes the alarms in the source channel where the ASC listens to. For each received alarm, the ASC publishes its enriched view in one or more category channel where the CERN operator GUI listens to alarms to show in the table.

ACS adds the blues lines and boxes, i.e. clients to listens to sources and categories alarms. The graph shows the different view of the alarms of the two ACS clients[18].

ACS source and category clients are in turn used by the alarmSourcePanel and the alarmPanel to display alarms .

ACS category client is an API to access alarms published by the ASC, i.e. the same alarms the CERN operator GUI shows in the table. In this context, ACS category client is an helper class to easily access categories from inside ACS code.

The ACS source client is instead something not present in CERN alarm system where nobody is allowed to listen to source alarms besides the ASC.

[18] Both the clients are in ACSLaser/alarm-clients.

7.1 Alarm source client

SourceClient is a java alarm source client i.e. it connects the the source channel and listen to the alarms published by the sources. The client is used by the alarmSourcePanel.

The following example shows the usage of the source client:

```
private ContainerServices contSvcs;
private SourceClient sourceClient= new SourceClient(contSvcs);
sourceClient.addAlarmListener(this);
sourceClient.connect();
...
public void faultStateReceived(FaultState faultState) {
            System.out.println(faultState.getFamily());
      }
```

The client is initialized passing an instance of `ContainerServices`. The listener to the alarms must then be added to the source client that will start sending alarms after the call to `connnect()`.

Alarms are received in the `faultStateReceived` method of the listener (in the example above, the name of the family of each received alarm is written in the stdout).

This class can be of help when writing tests of the alarms sent by sources.

7.2 Alarm category client

CategoryClient is a category client allowing to listen to alarms published by the ASC into categories. This client is used by the alarmPanel.

The following shows the usage of CategoryClient:

```
import alma.acs.container.ContainerServices;
import alma.alarmsystem.clients.CategoryClient;
import cern.laser.client.data.Alarm;
import cern.laser.client.LaserException.LaserSelectionException;

...
private ContainerServices contSvc;
private CategoryClient categoryClient;
...
categoryClient=new CategoryClient(contSvc);
categoryClient.connect(this);
...
public void onAlarm(Alarm alarm) {
      System.out.println(alarm.toString());
}
...
public void onException(LaserSelectionException e) {
      ...
}
```

The client is built passing the `ContainerServices`.

The call to the connect triggers a sequence of actions:

- the client connects to the alarm system and gets the list of all the available categories
- the listener is connected to the alarm system to be notified about the arrival of alarms
- the category client gets all the active alarms i.e. all the alarms active, even those activated before the client connects to the alarm system
- all the active alarms are sent to the listener

A not `null` listener to receive the alarms published by the ASC must be set as parameter to the `connect`. The CategoryClient sends alarms to the listener through `onAlarm`, as defined by the `AlarmSelectionListener` interface.

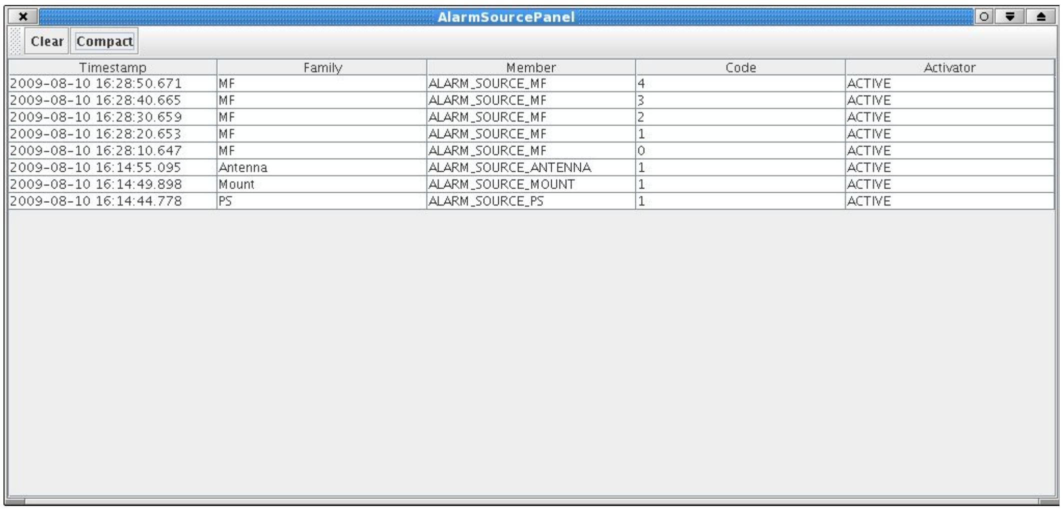
By default, the client connects to all the available categories, as shown in the example. However, there is an overloaded `connect` that accepts an array of Category to connect to listen to alarms:

```
public void connect(AlarmSelectionListener listener, Category[] categories)
```

7.3 alarmSourcePanel

The alarmSourcePanel is a little dialog showing the alarms published by sources i.e. alarms before they are processed by the ASC.

The panel, that needs to be started against a running ACS session, connects to the source channel and shows the alarms published by the sources inside a table. The alarms shown in this table are the same alarms received by the ASC because this panel listens to the same NC used by sources to send the alarms. Therefore the alarms sent to the alarm service through a CORBA call will never appear in this table.



Timestamp	Family	Member	Code	Activator
2009-08-10 16:28:50.671	MF	ALARM_SOURCE_MF	4	ACTIVE
2009-08-10 16:28:40.665	MF	ALARM_SOURCE_MF	3	ACTIVE
2009-08-10 16:28:30.659	MF	ALARM_SOURCE_MF	2	ACTIVE
2009-08-10 16:28:20.653	MF	ALARM_SOURCE_MF	1	ACTIVE
2009-08-10 16:28:10.647	MF	ALARM_SOURCE_MF	0	ACTIVE
2009-08-10 16:14:55.095	Antenna	ALARM_SOURCE_ANTENNA	1	ACTIVE
2009-08-10 16:14:49.898	Mount	ALARM_SOURCE_MOUNT	1	ACTIVE
2009-08-10 16:14:44.778	PS	ALARM_SOURCE_PS	1	ACTIVE

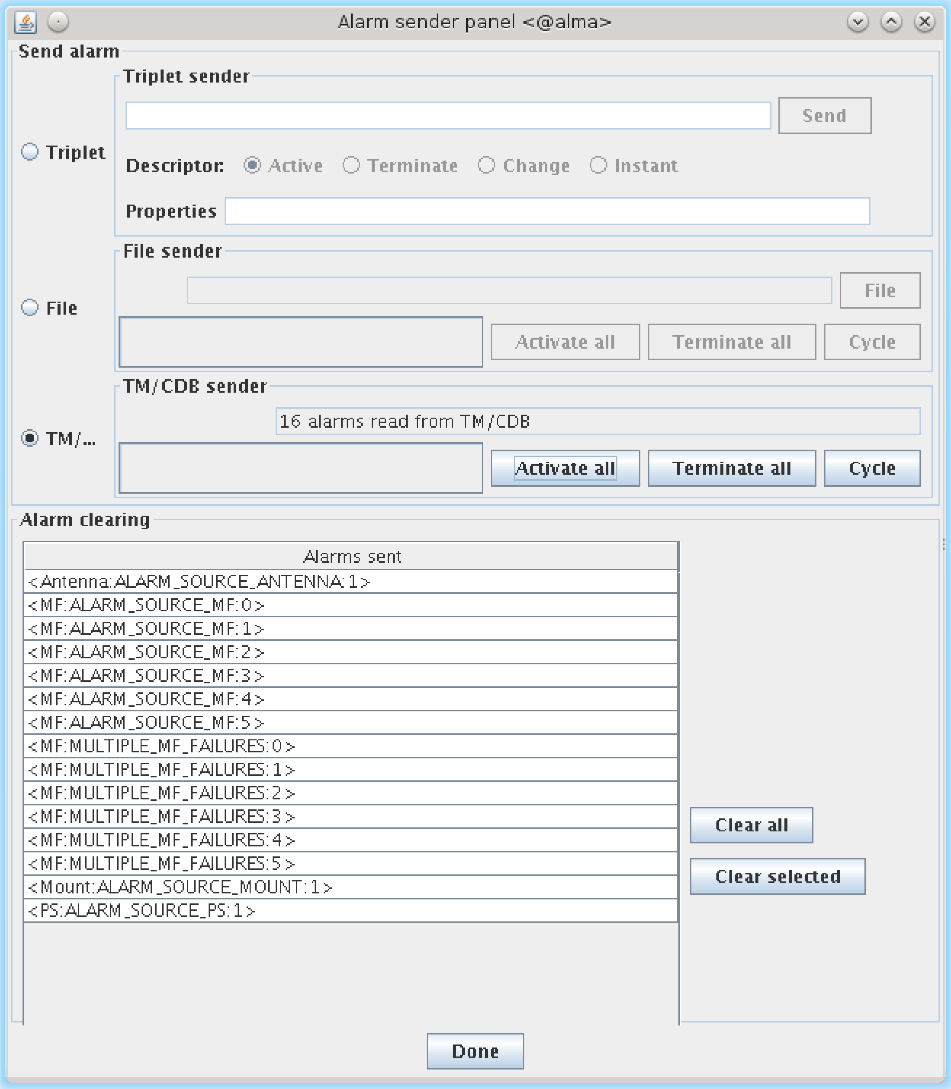
Figure 15: A snapshot of the alarm source panel.

This panel is very useful fro debugging: when the user wants to monitor the alarms published by sources. The table shows the triplet and the activator (the ACTIVE/INACTIVE status) of each alarm.

The toolbar has two buttons, Clear that clear the table and Compact that cause the table to show the alarms in a more compact view containing only one column for the triplet and another for the activation state.

7.4 The alarm sender panel

ACS provide a panel for testing the AS: it allows to send alarms with the desired triplet to the AS. Theis panel is launched issuing the alarmSenderPanel command:



Alarms sent
<Antenna:ALARM_SOURCE_ANTENNA:1>
<MF:ALARM_SOURCE_MF:0>
<MF:ALARM_SOURCE_MF:1>
<MF:ALARM_SOURCE_MF:2>
<MF:ALARM_SOURCE_MF:3>
<MF:ALARM_SOURCE_MF:4>
<MF:ALARM_SOURCE_MF:5>
<MF:MULTIPLE_MF_FAILURES:0>
<MF:MULTIPLE_MF_FAILURES:1>
<MF:MULTIPLE_MF_FAILURES:2>
<MF:MULTIPLE_MF_FAILURES:3>
<MF:MULTIPLE_MF_FAILURES:4>
<MF:MULTIPLE_MF_FAILURES:5>
<Mount:ALARM_SOURCE_MOUNT:1>
<PS:ALARM_SOURCE_PS:1>

The panel allows to send alarms with the methods each of which is selected with a radio button:

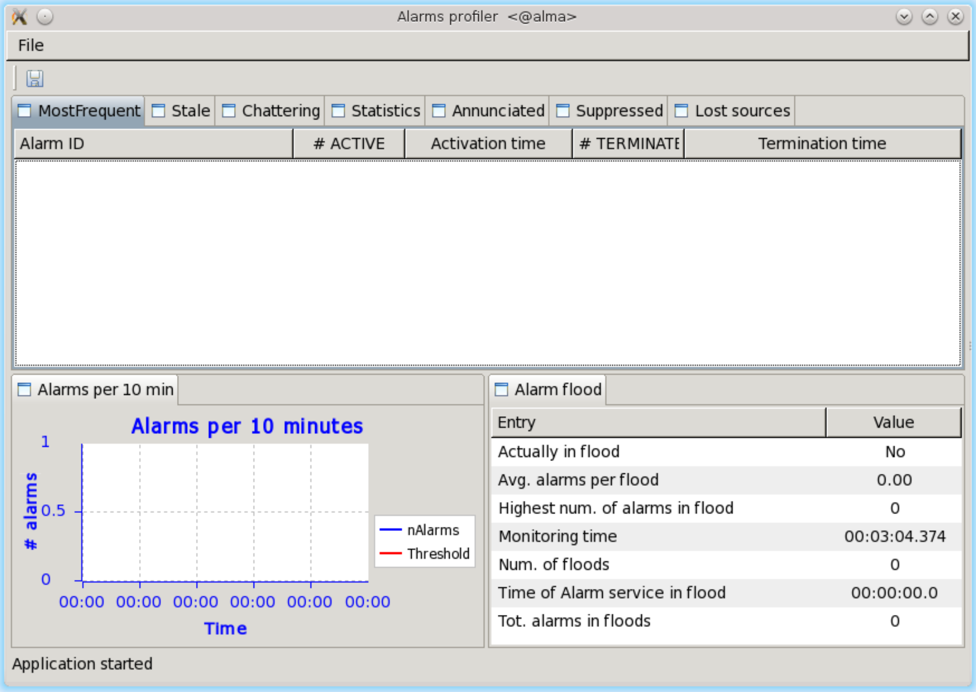
- Triplet: the triplet must be written in the text field in the form FF,FM,FC; the activation state of the alarm to send must then be selected between Activate and Terminate (the other 2 states Change and Instant are not yet implemented)
- File: the triplets of the alarms to send are written in a text file
- TM/CDB: the definition of the triplets of the alarm to send to the ASC are read from the CDB. This is probably the most useful modality.

For the File and the TM/CDB methods there i s Cycle button: when it is pressed the panel randomly activates/terminates alarms to strres the ASC.

In the Alarm clearing area, there is a table with all the alarms that have been activated by the panel. The user can select which one he wants to terminate or terminate all of the at once.

7.5 Profiling of the alarm system

ACS provides a way to profile the alarm system and identify problematic area of the alarm system. The tool is launched issuing the alarmsProfiler. It is a eclipse RCP whose sources are in acsGUIs/AlarmsProfiler. Note that this application is not built by default i.e. it is not part of ACS binary distribution.



Entry	Value
Actually in flood	No
Avg. alarms per flood	0.00
Highest num. of alarms in flood	0
Monitoring time	00:03:04.374
Num. of floods	0
Time of Alarm service in flood	00:00:00.0
Tot. alarms in floods	0

The application starts profiling the alarm system as soon as the “Connect to ACS” menu item of the File menu is selected. At that moment, the alarmsProfiler connects to the categories and to the source notification channels and listens to all the alarms published by the sources and those sent by the ASC to the clients and elaborate statistics. The profiling should last a relevant amount of time to let the tool gather enough information.

The panel shows a tab for each different statistic it calculates and shows a graph of the number of alarm sent to the operator GUI in the last ten minutes.

A ASCII report can be generated by pressing the disk icon in the tool bar: the output of the ASCII file is formatted to be copied into a twiki page.

7.5.1 Most frequent alarm

It is a ranking of the most frequent to the least frequent alarm during the analysis period. Often only a few alarms are a significant fraction of the entire system load.

We need to check if those frequent alarm have been designed to show up so frequently. Working on these alarms can dramatically improve the alarm system.

The tool create the list by listening at alarms produced by the source (i.e. not yet processed by the alarm service). The table has one row for each alarm identified by its id (the triplet), its number of activations and deactivation and the last time of each state change.

7.5.2 Stale alarms

The list of active alarm and how long they are in the activation state. Alarms that are always active do not provide useful information to the operators.

The tool shows a table associating the ID of the alarm to the time it is in activate state.

7.5.3 Chattering alarms

A list of alarms that change their state from activate to terminate at least 3 times per minute. Of course these alarms are not been cleared because of an operator action so we have to fix them.

The tool shows one table with one row per each alarm identified by its triplet. It reports the number of activations, deactivations, state changes and the peak timestamp of the event.

7.5.4 Statistics

A useful set of numbers:

- activation alarms sent by sources
- terminated alarms sent by sources
- total number of source events
- activated alarms sent by the alarms service to the clients (i.e. to the alarm panel and visible to operators)
- activated alarms sent by the alarms service to the clients (i.e. to the alarm panel and visible to operators)
- suppressed (i.e. reduced) alarms sent by the alarm service to the clients (the operator can or cannot see these events depending on the setting of the alarm panel)
- the number of alarms per each priority level

This panel gives the feeling of the load of the system at source level as well as at level of the alarm clients.

7.5.5 Annunciated alarms

The alarms annunciated to operators i.e. the active alarms that the alarm service sent to the clients.

The table has one row for each alarm with the number of times it has been sent to the clients. Given that the alarm service does not send twice the same alarms, it means that the state of the alarm must have been changed from active to terminate.
This table is correlated to the most frequent alarms of the first tab but the alarms shown here are the alarms sent by the alarm service to the clients instead of those directly produced by the source. For example if a source activates 2 times in sequence the same alarm, it appears with a number of 2 in the most frequent alarm tab but with a number of 1 in this tab because the alarm service does not send the same activate event 2 time.

7.5.6 Suppressed alarms

It is the list of alarms reduced by the alarm service.This tab gives an estimation of how many alarms are active in the system but hidden to the operator.Actually, the alarm service sends reduced alarms to the client with a flag and the operator can see in the alarm panel the reduced alarms by changing the setting of the GUI.

7.5.7 Lost sources

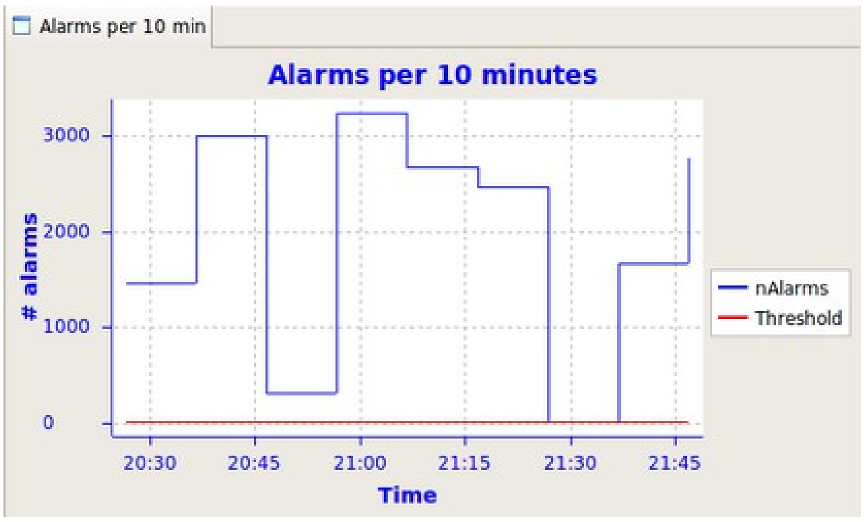
this table identifies the alarms produced by the sources but that do not have any documentation in the CDB. When the alarm service receives these types of alarms, it does nothing because it can't find what to do in the database. These alarms never arrive to the operators that could miss some important notification.

The panel provides a list of alarm IDS that the developer should document as soon as possible.

7.5.8 Alarms per 10 minutes

It is a graph showing the number of alarms received in the last 10 minutes. This allows to identify bursts of alarms and to correlate them to a particular activity in the control room.

If the number of alarms per 10 minutes is greater then 10, then there is an **alarm flood**. Alarm floods means that the number of presented alarms is too big to be handled efficiently by the operator. The flood threshold of 10 alarm is the red line in the graph.



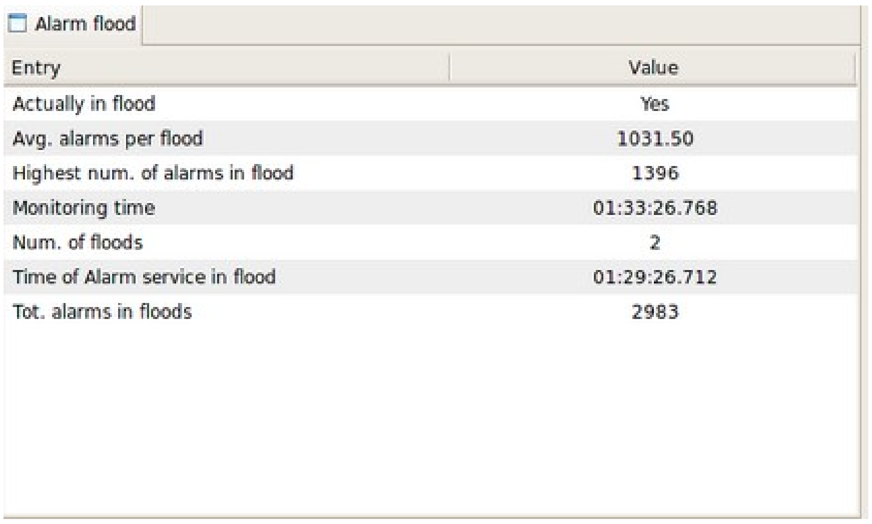
7.5.9 Alarm flood

As we said before, an alarm flood happens when the operator receives 10 or more alarms per minute. Given that an alarm flood is a situation that the operators presumably can't handle they should never happen or have a very short duration.

The alarm flood tab says if the alarm system is currently in a flood, an information that can help correlating the actual activity in the control room with the performance of the alarm system.
it also shows the number of alarms in the actual flood to have an idea on how far is that number with the ideal of 10.

This panels also reports the total monitoring time i.e. the total time that tool is profiling the system. It is important to correlate this time with the time spent in flood. ideally, the time in flood should be around zero.

The total number of flood and the average number of alarms in flood is also presented by the tool.



Entry	Value
Actually in flood	Yes
Avg. alarms per flood	1031.50
Highest num. of alarms in flood	1396
Monitoring time	01:33:26.768
Num. of floods	2
Time of Alarm service in flood	01:29:26.712
Tot. alarms in floods	2983

7.6 BACI properties

BACI properties have been modified in order to send an alarm when their value becomes lower than the low limit or higher than the high limit for continues read-only (RO) BACI property like ROdouble, ROfloat, … Or, when the value (state) is in alarm state for discrete RO BACI properties like ROpattern, RO<enum>, … The low and high limits, or alarm states of a property are defined in the CDB.

To enable checking the value of a BACI property for alarms the user has to set the value of *alarm_timer_trig* in the CDB entry of the property[19] on a value different than 0. The default value for this field, defined in the schema, is 0 meaning that the checking for alarms is disabled.

Besides the *alarm_timer_trig*, the methods `startPropertiesMonitoring()` and `stopPropertiesMonitoring()` of `CharacteristicComponentImpl` allow the developers to enable/disable the monitoring of the properties at the API level.

[19] Please, refer to the BACI documentation for further details about the CDB entry of a property.

As it is true for other alarms also alarms generated by BACI properties are represented with triplet: FF, FM and FC. The value of FF/FM of the triplet of an alarm from the BACI property can be set in several ways:

- default value which is taken if no value is specified by other mean (CDB or API):
    - for FF hardcoded string "**BACIProperty"**
    - for FM the name of the BACI property
- value defined in CDB; there are two optional attributes for each read-only property:
    - `alarm_fault_family` for FF
    - `alarm_fault_member` for FM
- set via API: each read-only BACI property, like ROdouble, has two methods:
    - `setAlarmFaultFamily` to set FF
    - `setAlarmFaultMember` to set FM

Setting FF/FM using API has precedence over value read from CDB, which has precedence over default value.

The current set value of FF/FM can be retrieved at any point using API of read-only BACI properties: `getAlarmFaultFamily`, and `getAlarmFaultMember`, respectively.

The value of FC is a fixed number that can have different values depending on what kind of BACI property generated the alarm:

- **"1"** for *discrete* RO properties such as ROpattern, or RO<enum>, if the corresponding alarm indicates that the property value (state) is in an **alarm state**.
- **"2"** for *continuous* RO properties like ROdouble, ROfloat, if the corresponding alarm indicates that the value is **under the low limit**
- **"3"** for *continuous* RO properties like ROdouble, ROfloat, if the corresponding alarm indicates that the value is **over the high limit**

It is not possible to modify the FC value from the CDB or the baci API.

Each BACI alarm can also have a defined level. The level can be specified in the CDB where each RO property has an optional attribute: `alarm_level`. Its default value is 0. The level value appears in the alarm system as the *BACI_Level* alarm property.

Besides the triplet (FF, FM and FC) and *BACI_Level,* BACI alarms can contain several other alarm properties:

- **BACI_Value** the value of BACI property that triggered the alarm
- **BACI_Property** the name of the BACI property
- **BACI_Description** description of the BACI property read from CDB.

- **BACI_Position** position of the value in the sequence that triggered the alarm – only for sequence BACI properties.

- **BACI_BitPosition** bit position in ROpattern that triggered the alarm – only for ROpattern

- **BACI_BitDescription** description of the bit that triggered the alarm BACI_BitPosition – only for ROpattern

The FS of each BACI property can be ACTIVE or INACTIVE depending on whether its value is in the proper range or not – exceed low or high limit or not.

As discussed in the CDB section, the sources can publish alarms even if they are not defined in the database. On the other hand, the ASC discards all the alarms it receives that do not match with an entry in the CDB.

In other words, it means that there is nothing to configure in order for a BACI property to send alarms when its value is out of the allowed range. However you must configure the `Alarms` branch of the CDB if you want to have these alarms processed by the ASC and visible in the GUI, that is normally the desired behavior. Usage of default members is very helpful defining the CDB for BACI properties.

7.7 Sending of alarms through an IDL CORBA call

The IDL of the alarm service, contains a method to directly send alarms to the ACS i.e. without passing through the notification channel:

```
...
interface CERNAlarmService: AlarmService {
   ...
   void submitAlarm(
                  in alarmsystem::Triplet triplet,
                  in boolean active,
                  in string sourceHostName,
                  in string sourceName,
                  in alarmsystem::Timestamp sourceTimestamp,
                  in CosPropertyService::Properties alarmProperties)
                   raises
                        (ACSErrTypeCommon::BadParameterEx,
                        ACSErrTypeCommon::UnexpectedExceptionEx);
      };
...
```

The following java example shows how to send an alarm through a CORBA call:

```
CernAlarmServiceUtils utils = new CernAlarmServiceUtils(contSvcs);
CERNAlarmService alarmService =
          CERNAlarmServiceHelper.narrow(utils.getAlarmService());
Triplet triplet = new Triplet(IDLTestFF, IDLTestFM, IDLTestFC);
Timestamp timestamp = new Timestamp(System.currentTimeMillis(), 0);
org.omg.CosPropertyService.Property[] props =
          new org.omg.CosPropertyService.Property[0];
alarmService.submitAlarm(
9                        triplet,
10                       true,
11                       "ThisHostName",
12                       this.getClass().getName(),
13                       timestamp,
14                       props);
```

To send alarms with a CORBA call, the software has to get a CORBA reference to the alarm service (lines 1-3). The triplet of the alarm must be built by passing the fault family, member and code (line 4).
Optionally, a set of user properties can be associated to the alarm (lines 6-7).
Finally the IDL method can be invoked (lines 8-14).

To keep the example short, we do not get the real name of the host wher ethe software is running.

Sending alarms with this method must be avoided: it is intended to be used only when the notification channel is not available or for very special cases like for example by ACS daemons.