ACS Alarm System

A. Caproni[ac]

a. ESO, European Organization for Astronomical Research in the Southern Hemisphere

b. CERN, European Organization for Nuclear Research

c. INAF-OATs, Osservatorio Astronomico di Trieste

d. COSYLAB, Control System Laboratory

Alarm System

The AS collects information from the environment and shows to the user what's happening and possibly how to fix an abnormal situation in a short time.

- shows only the relevant alarms (can show all the active alarms)
- has the concept of authorized users
- doesn't take any action to fix a problem
- alarm must be acknowledged
- stores the alarms in the database

ACS Alarm system

2 available implementations:

- ACS
- CERN

ACS implementation is used if:

- CBD/Alarms folder does not exist
- Implementation property not set to “CERN” in the CDB

ACS implementation logs a message for each alarm (active/inactive):

```
Alarm sent: <AlarmSource,ALARM_SOURCE_MOUNTCPP,1> ACTIVE

Alarm sent: <AlarmSource,ALARM_SOURCE_MOUNTCPP,1> TERMINATE
```

Log level Alert/Emergency

ACS Alarm system

CERN implementation consists of the alarm system developed at CERN for the LHA collider integrated with ACS.

We have:

- a working prototype
- an agreement with CERN
- the new version of the source code

The CERN implementation is part of ACS (need to be explicitly activated)

Things can change and they'll do!
Hopefully not the interfaces.

Large Hadron Collider Alarm Service (LASER)

LASER is the second generation alarm service developed at CERN for the Large Hadron Collider.

It is a messaging system: collects, stores, manages and distributes information about abnormal situations: the Fault States.

Each **Fault State** (FS) is identified by a triplet:

FS = <FF,FM,FC>

- the **Fault Family** (FF) groups elements of the same kind
- the **Fault Member** (FM) represents the failing instance
- the **Fault Code** (FC) identifies the particular problem

A Fault State can be **active or inactive**

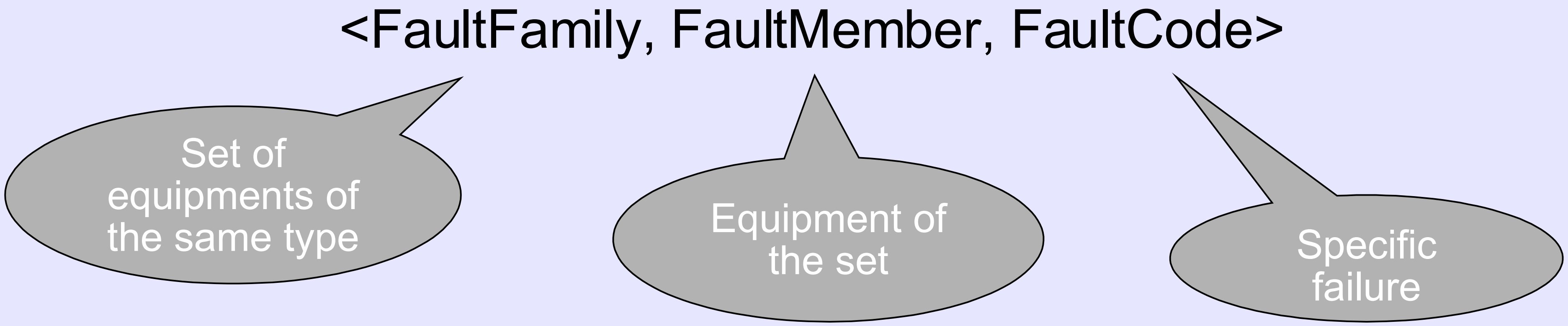
FaultState (triplet) - 1

FS = <FF, FM, FC>

- FF groups similar equipments (sw/hw) for example Antenna(s), ACS components
- FM identifies which particular equipment in the family is failing (for example Antenna3, MOUNT_3 component)
- FC is an integer identifying the particular problem occurring in the failing equipment
- It has a status, *active* ↔ *inactive*
- It has an heartbeat

The FS is a unique identifier! → One triplet per each possible alarm!

FaultState (triplet) - 2

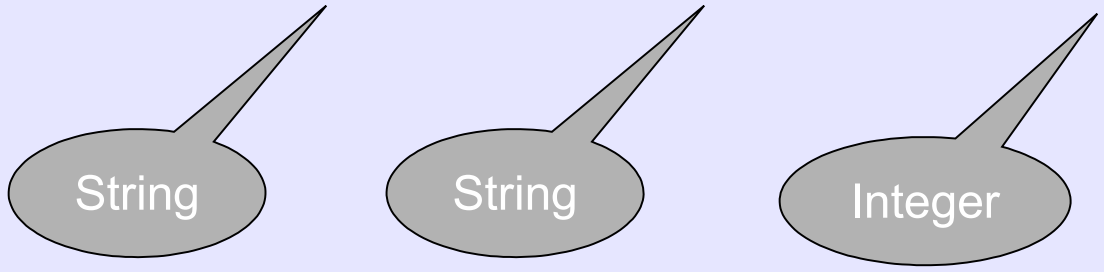

<FaultFamily, FaultMember, FaultCode>
Set of equipments of the same type
Equipment of the set
Specific failure

Examples:

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

FaultState (triplet) -3

<Fault Family, Fault Member, Fault Code>

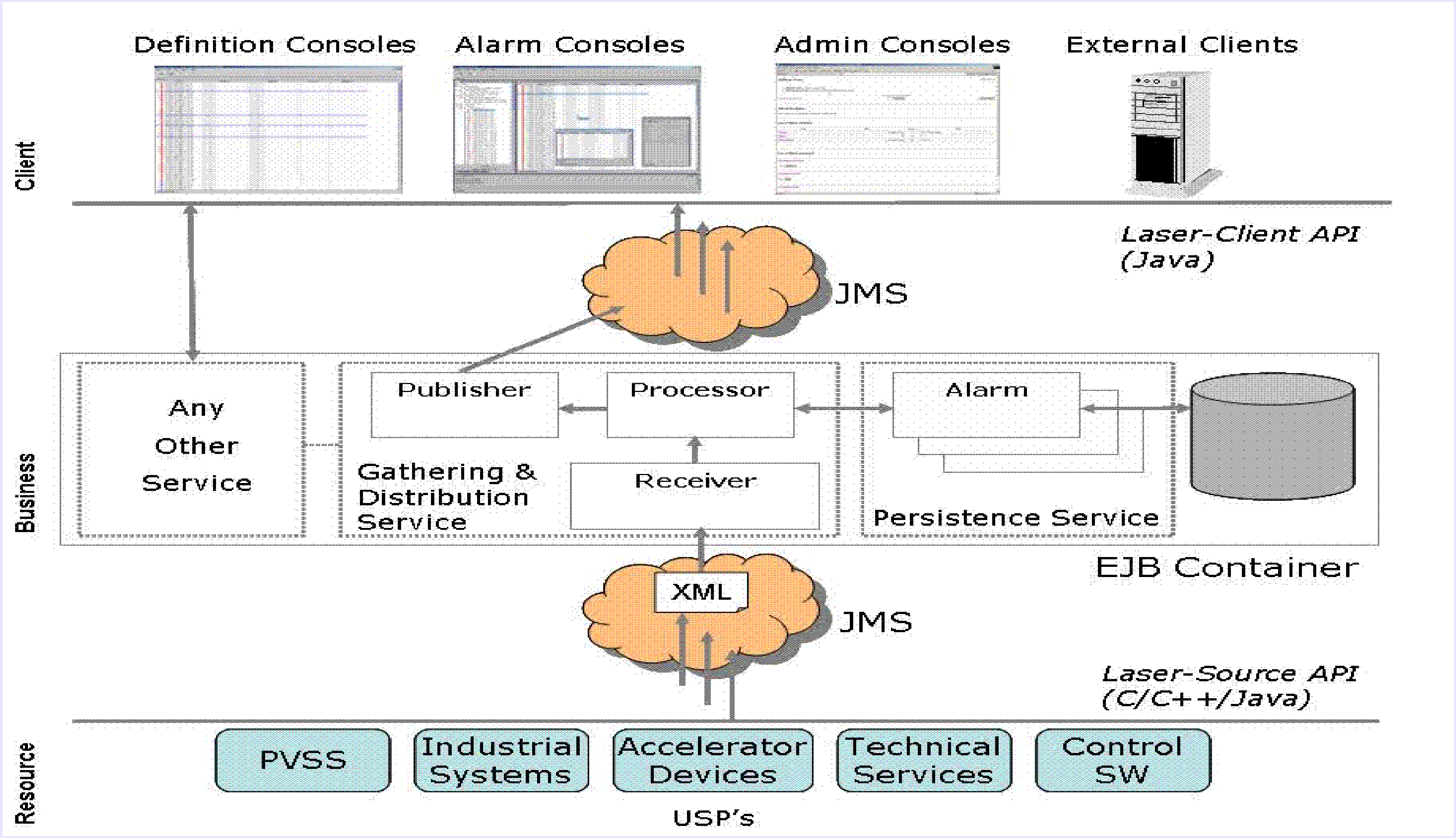


The definition of a FS is whatever you prefer!

$\downarrow$

- meaningful (maintenance, readability…)
- unique (it is an ID!)
- it says who, where and how something is failing

LASER - Overview



LASER – Resource tier

The resource tier is composed of sources of alarms:

- different programming languages (java, C, C++)
- different platforms
- each source has a definite set of FS whose state can be active/inactive
- each source sends an heartbeat to the service
- categories of alarms

LASER-source API:

- allows the sources to send alarms to the business tier
- no need of specific knowledge of the alarm service

LASER – An example

```
String faultFamily=”PowerSupplies”;
String faultMember=”PS3”;
int faultCode=100;
String faultState=FaultState.ACTIVE;

AlarmSystemInterface alarmSource =
    AlarmSystemInterfaceFactory.createSource(this.name());

FaultState fs = AlarmSystemInterfaceFactory.createFaultState(
    faultFamily, faultMember, faultCode);

fs.setDescriptor(faultState);
fs.setUserTimestamp(new Timestamp(System.currentTimeMillis()));

Properties props = new Properties();
props.setProperty(...);
...
fs.setUserProperties(props);

alarmSource.push(fs);
```

LASER- Business tier

- listen for FS changes and heartbeats
- build a complete view of each alarm
- reduces or masks alarms depending of:
- the status of the system
- the knowledge of the environment
- persist the FS
- track and archive the FS changes
- authenticate user of the clients
- allow the management of the FS without stopping the service
- publish alarms to the clients

LASER - Alarm informations

- System name
- Identifier of the system
- Description of the alarm
- The cause of the alarm
- The action to fix the problem
- The consequence of the alarm
- The priority of the alarm (4 priorities)
- The URL with detailed information
- A phone number to call
- The location of the source
- The name/ID of the responsible person

LASER - Reduction rules

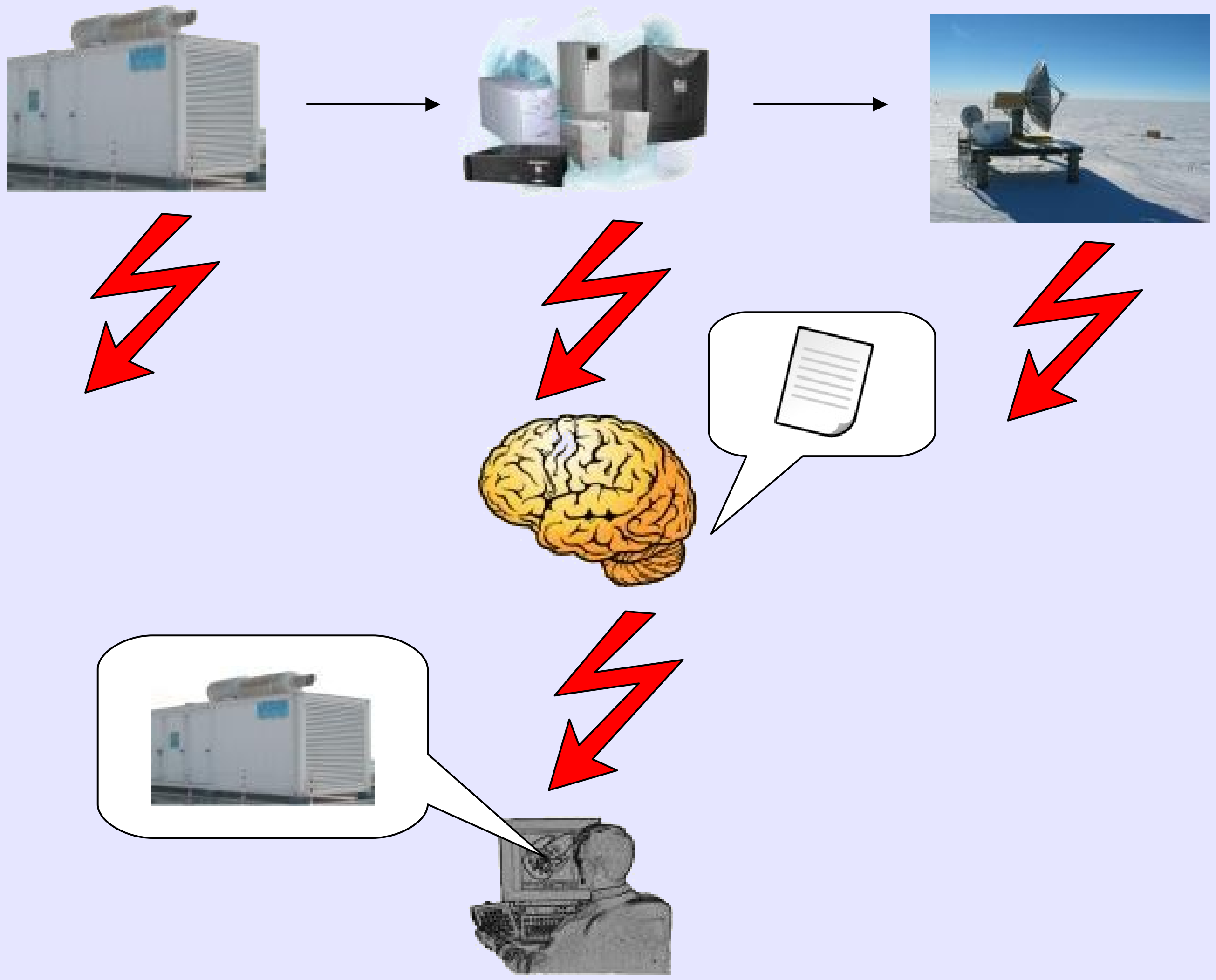
The reduction aims to help the user to easily identify the root cause of a cascade of alarms.

The service has a knowledge of the environment by means of a set of reduction rules.

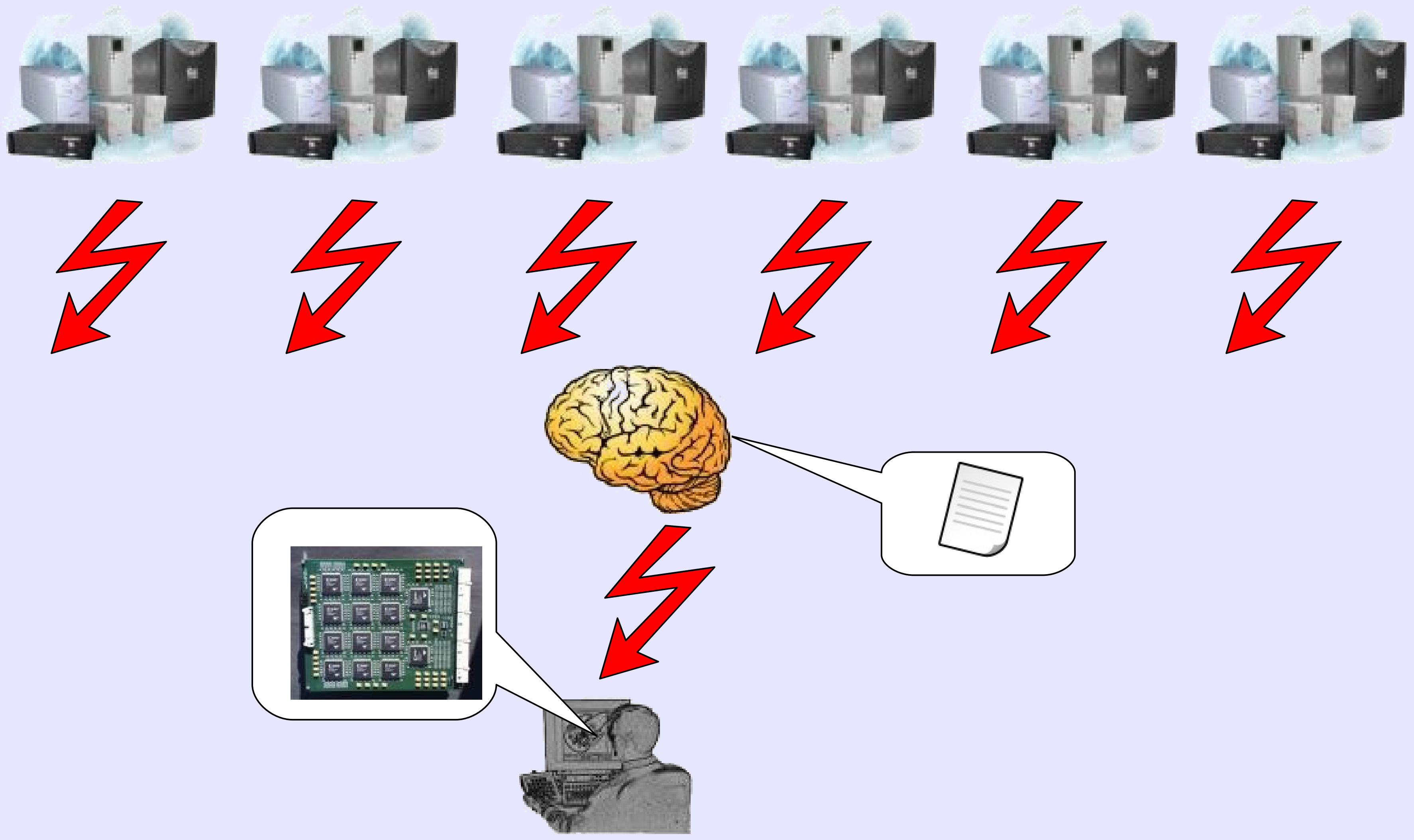
Node reduction (NR)
If a failure in the equipment A triggers a failure in another equipment, B, then the latter is reduced.

Multiplicity reduction (MR)
If the number of alarms *of the same type* is greater then a given threshold then these alarms are reduced and a *new alarm* is produced

Node reduction



Multiplicity reduction



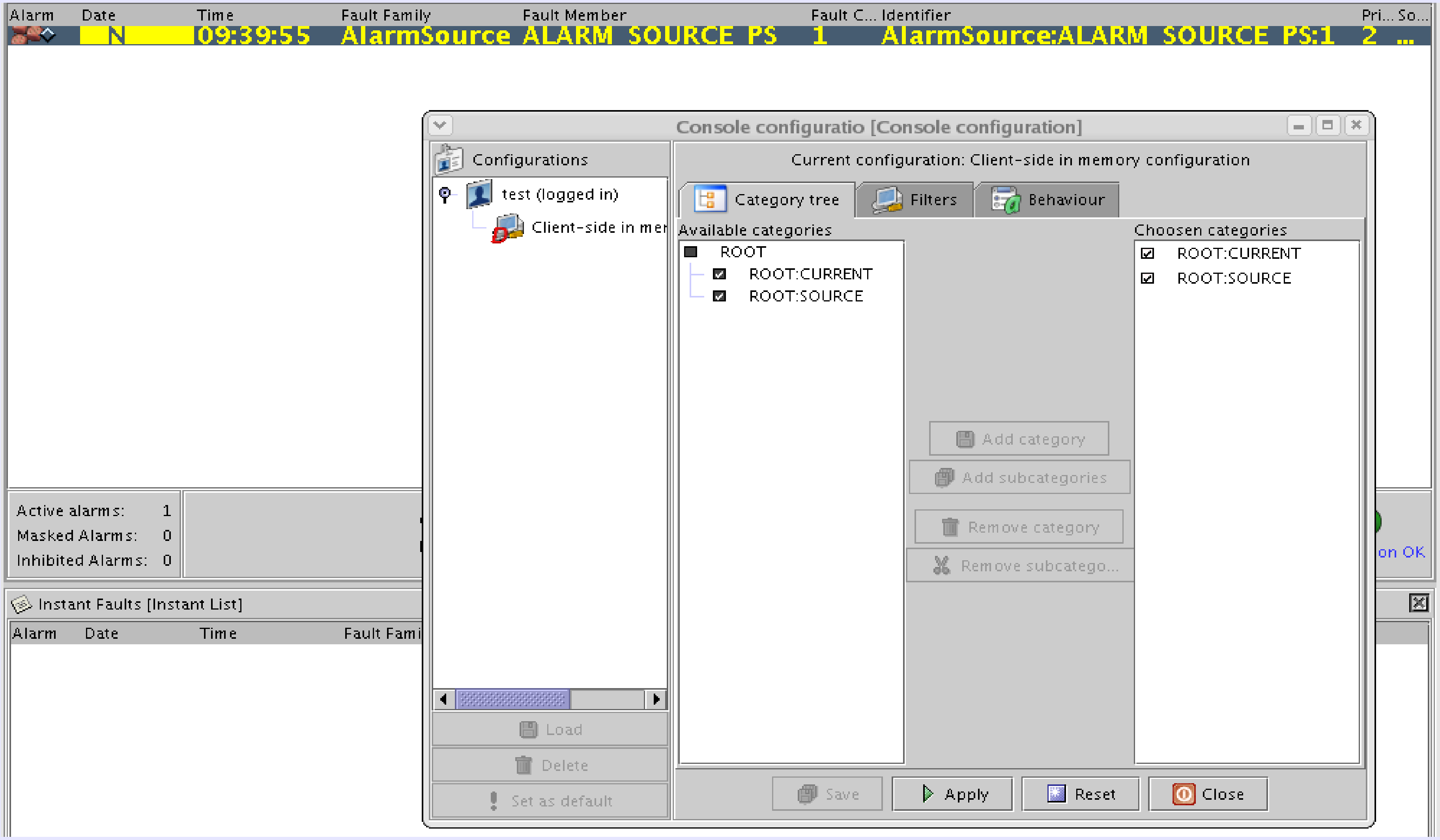
LASER – Client tier

Composed of java applications that consume the data produced by the business tier.

- alarm console
- definition console
- login and configuration facilities

There is no need to stop the alarm service while administering the system.

Operator GUI



Alarm	Date	Time	Fault Family	Fault Member	Fault C...	Identifier	Pri...	So...
	N	09:39:55	AlarmSource	ALARM_SOURCE_PS	1	AlarmSource:ALARM_SOURCE_PS:1	2	...

Active alarms: 1
Masked Alarms: 0
Inhibited Alarms: 0

Instant Faults [Instant List]

Console configuratio [Console configuration]

Configurations
test (logged in)
Client-side in mem

Current configuration: Client-side in memory configuration

Available categories
ROOT
☑ ROOT:CURRENT
☑ ROOT:SOURCE

Choosen categories
☑ ROOT:CURRENT
☑ ROOT:SOURCE

ACS alarm system

ESO, Garching, Jan 2007

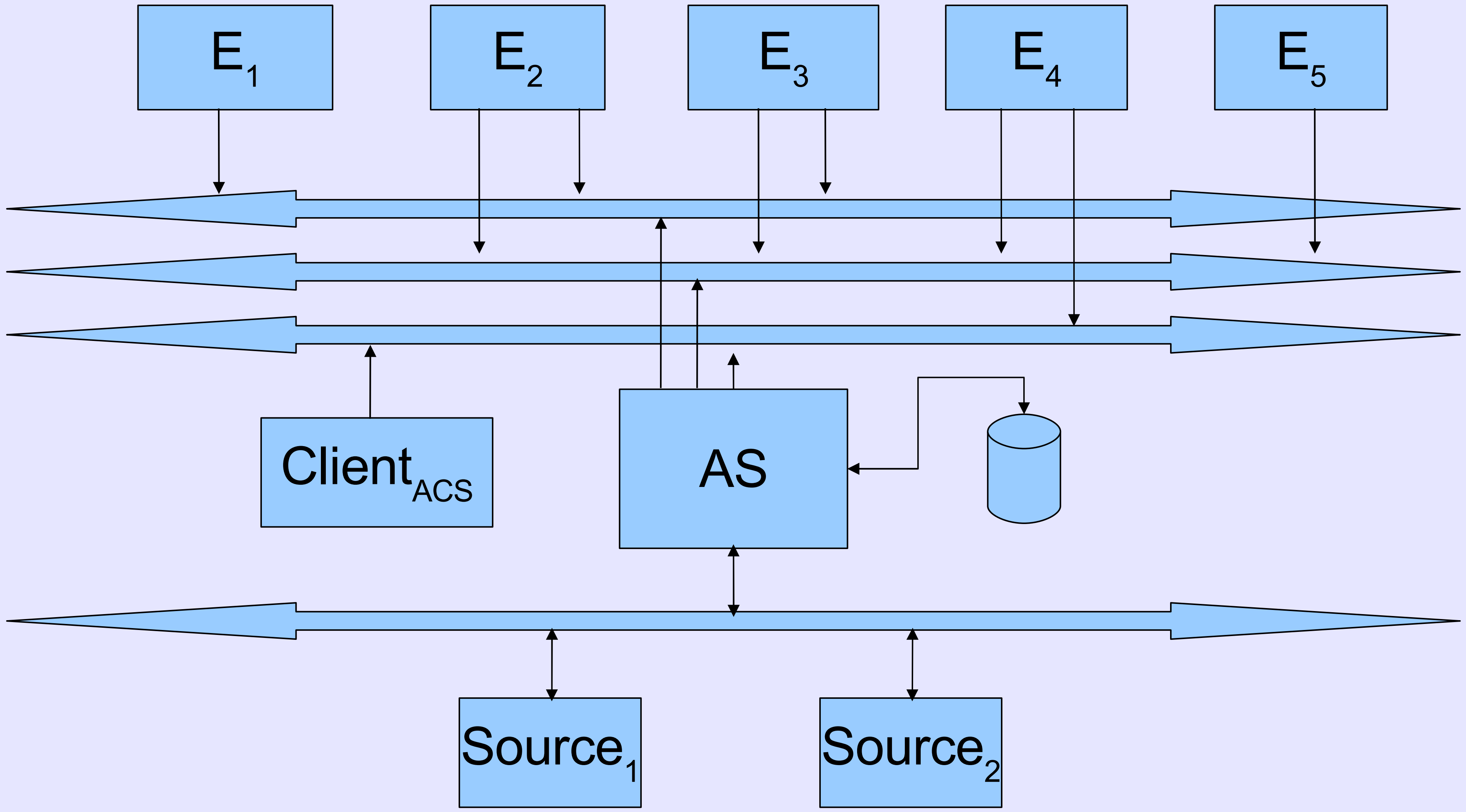
Communications between the tiers

The communication between the tiers happens by means of JMS.

JMS implementation of ACS sends messages through NCs

It is possible to write clients that listen to the NC and take actions independently of the Alarm Service

ACS LASER structure



BACI properties

< FF=”BACIProperty”, FM=property_name, FC=1 >

- This feature must be explicitly enabled by the user changing the value of *alarm_timer_trig* in the CDB entry of the property.
- An alarm is sent when the value of a property become lower then the minimum or higher then the maximum. The min and max possible values of a property are defined in the CDB.
- Besides the alarm_timer_trig, the methods **startPropertiesMonitoring()** and **stopPropertiesMonitoring()** of `CharacteristicComponentImpl` allow the developers to enable/disable the monitoring of the properties.

The CDB must be configured in order to see BACI alarms in the GUI.

Sending alarms in C++

- C++ API mirrors the Java API
- Integrated with BACI (i.e. CharacteristicComponent) properties
- Libraries to add to the Makefile:
  - `acsAlSysSource`
  - `acsErrTypeAlarmSourceFactory`

Sending alarms in C++: how to program

```
// constants we will use when creating the fault
string family = "AlarmSource";
string member = "ALARM_SOURCE_MOUNT";
int code = 1;

// Step 1: create the AlarmSystemInterface using the factory
auto_ptr<AlarmSystemInterface> alarmSource = AlarmSystemInterfaceFactory::createSource();

// Step 2: create the FaultState using the factory
auto_ptr<FaultState> fltstate = AlarmSystemInterfaceFactory::createFaultState(family, member, code);

// Step 3: set the fault state's descriptor
string stateString = faultState::ACTIVE_STRING;
fltstate->setDescriptor(stateString);

// Step 4: create a Timestamp and use it to configure the FaultState
Timestamp * tstampPtr = new Timestamp();
auto_ptr<Timestamp> tstampAutoPtr(tstampPtr);
fltstate->setUserTimestamp(tstampAutoPtr);

// Step 5: 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);

// Step 6: push the FaultState to the alarm server
FaultState stateToPush(*fltstate);
alarmSource->push(stateToPush);
```

Java API - *AcsAlarmSystemInterfaceFactory*

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

```
public static ACSAlarmSystemInterface createSource()
      throws ACSASFactoryNotInitedEx, SourceCreationErrorEx;

public synchronized static ACSAlarmSystemInterface createSource(
  String sourceName)
      throws ACSASFactoryNotInitedEx, SourceCreationErrorEx;

public synchronized static ACSFaultState createFaultState()
      throws ACSASFactoryNotInitedEx, FaultStateCreationErrorEx;

public static ACSFaultState createFaultState(
   String family,
   String member,
   int code)
      throws ACSASFactoryNotInitedEx, FaultStateCreationErrorEx;

...
```

Java API - *AcsAlarmSystemInterface*

```
...

public void push(ACSFaultState state);

public void push(Collection states);

public void pushActiveList(Collection active);

...
```

Java API - *AcsFaultState*

```
public final static String ACTIVE = "ACTIVE";
public final static String TERMINATE = "TERMINATE"


...
public void setDescriptor(String descriptor);
public void setUserProperties(Properties properties);
public void setUserTimestamp(Timestamp timestamp);
...
```

C++ API - *AcsAlarmSystemInterfaceFactory*

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

```
static auto_ptr<AlarmSystemInterface> createSource(string sourceName);

static auto_ptr<AlarmSystemInterface> createSource();

static auto_ptr<acsalarm::FaultState>createFaultState(
   string family,
   string member,
   int code);

static auto_ptr<acsalarm::FaultState>createFaultState();
```

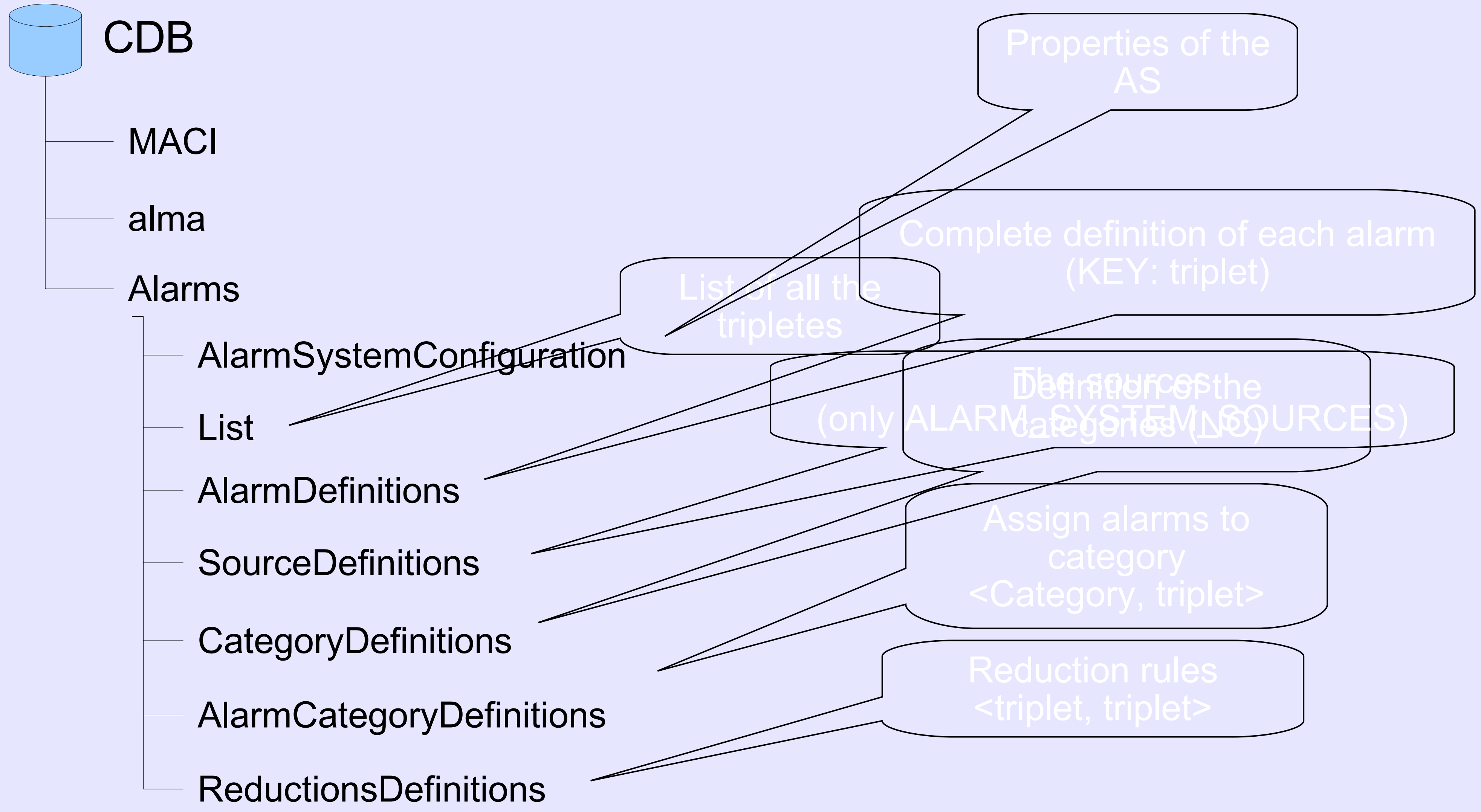
C++ API - *AcsAlarmSystem*

```
...

virtual void push(FaultState & state); //raises ASIException = 0;

virtual void push(vector<FaultState> & states); // raises ASIException
= 0;

virtual void pushActiveList(vector<FaultState> & activeFaults);

...
```

C++ API - *FaultState*

```
...
void setDescriptor(const string & newDescriptor);
virtual void setUserProperties(auto_ptr<Properties> theProperties);
virtual void setUserTimestamp(auto_ptr<Timestamp> theTimestamp);
...
```

CDB configuration


CDB
MACI
alma
Alarms
AlarmSystemConfiguration
List
AlarmDefinitions
SourceDefinitions
CategoryDefinitions
AlarmCategoryDefinitions
ReductionsDefinitions
Properties of the AS
List of all the tripletes
Complete definition of each alarm (KEY: triplet)
The sources (only ALARM_SYSTEM_SOURCES)
Definition of the categories (LDO)
Assign alarms to category <Category, triplet>
Reduction rules <triplet, triplet>

CDB population

1. Write **all** the triplets
 `List/List.xml`
2. Write the sources (now only ALARM_SYSTEM_SOURCES)
 `SourceDefinitions/SourceDefinitions.xml`
3. Write the definition of each alarm: require the triplet and the (only one) source
 `AlarmDefinitions/<fault_member>/<fault_member>.xml`
4. Write the definition of the categories
 `CategoryDefinitions/CategoryDefinitions.xml`
5. Assign alarms to categories: require the triplet and the category name
 `AlarmCategoryDefinitions/AlarmCategoryDefinitions.xml`
6. Define the reduction rules: require the triplets
 `ReductionDefinitions/ReductionDefinitions.xml`

CDB - AlarmSystemConfiguration

Alarms/AlarmSystemConfiguration/AlarmSystemConfiguration.xml

```
<alarm-system-configuration ... >

  <configuration-property name="Implementation">
    CERN
  </configuration-property>

</alarm-system-configuration>
```

CDB - List

Alarms/List/List.xml

```
<alarm-definition-list... >
  ..
    <alarm-definition
     fault-family="AlarmSource"
     fault-member="ALARM_SOURCE_PS"
     fault-code="1" />
  ...
</alarm-definition-list>
```

CDB - SourceDefinitions

Alarms/SourceDefinitions/SourceDefinitions.xml

```
<source-definitions ...>
  <sources-to-create>
   <source-definition name="ALARM_SYSTEM_SOURCES">
      <connection-timeout>15</connection-timeout>
      <description>The alarm source for all the sources</description>
      <responsible-id>1</responsible-id>
   </source-definition>
  </sources-to-create>
</source-definitions>
```

CDB - AlarmDefinitions

Alarms/AlarmDefinitions/<FM>/<FM>.xml

```
<alarm-definition-list ...>
  <alarm-definition fault-family="AlarmSource" fault-member="ALARM_SOURCE_MOUNT" fault-
code="1">
        <visual-fields>
            <system-name></system-name>
            <identifier></identifier>
            <problem-description></problem-description>
        </visual-fields>
        <instant>false</instant>
        <!--cause></cause-->
        <!--action></action-->
        <!--consequence></consequence-->
        <priority>1</priority>
        <responsible-id>123</responsible-id>
        <!--piquetGSM>12345</piquetGSM-->
        <!--help-url></help-url-->
        <source-name>ALARM_SYSTEM_SOURCES</source-name>
        <!--location>
            <building></building>
            <floor></floor>
            <room></room>
            <mnemonic></mnemonic>
            <position></position>
        </location-->
        <!--piquetEmail></piquetEmail-->
    </alarm-definition>
</alarm-definition-list>
```

CDB - CategoryDefinitions

Alarms/CategoryDefinitions/CategoryDefinitions.xml

```
<category-definitions ...>
  <categories-to-create>
    ...
    <category-definition path="ROOT:SOURCE">
       <description>
          The category for all alarm states.
       </description>
    </category-definition>
    ...
  </categories-to-create>
</category-definitions>
```

CDB - AlarmCategoryDefinitions

Alarms/AlarmCategoryDefinitions/AlarmCategoryDefinitions.xml

```
<alarm-category-definitions
        xmlns="urn:schemas-cosylab-com:AcsAlarmSystem:1.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <category-links-to-create>
            ...
    <alarm-category-link>
       <category>
          <category-definition path="ROOT:SOURCE"/>
       </category>
       <alarm>
          <alarm-definition
               fault-family="AlarmSource"
               fault-member="ALARM_SOURCE_A"
               fault-code="1"/>
       </alarm>
    </alarm-category-link>
            ...
  </category-links-to-create>
</alarm-category-definitions>
```

CDB - ReductionDefinitions

Alarms/ReductionDefinitions/ReductionDefinitions.h

```
<reduction-definitions ...>
    ...
  <links-to-create>
    <reduction-link type="NODE">
      <parent>
        <alarm-definition fault-family="FF1" fault-member="FM1" fault-code="1"/>
      </parent>
      <child>
        <alarm-definition fault-family="FF2" fault-member="FM2" fault-code="1"/>
      </child>
    </reduction-link>
    ...
  </links-to-create>

  <thresholds>
    ...
    <threshold value="2">
        <alarm-definition fault-family="FF3" fault-member="FM3" fault-code="1"/>
    </threshold>
    ...
  </thresholds>
</reduction-definitions>
```

CDB – ReductionDefinitions - NR

Alarms/ReductionDefinitions/ReductionDefinitions.h

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

CDB – ReductionDefinitions - MR

Alarms/ReductionDefinitions/ReductionDefinitions.h

```
<reduction-link type="MULTIPLICITY">
  <parent>
    <alarm-definition fault-family="AlarmSource" fault-member="MULTIPLE_MF_FAILURES" fault-code="1"/>
  </parent>
  <child>
    <alarm-definition fault-family="AlarmSource" fault-member="ALARM_SOURCE_MF" fault-code="1"/>
  </child>
</reduction-link>
<reduction-link type="MULTIPLICITY">
  <parent>
    <alarm-definition fault-family="AlarmSource" fault-member="MULTIPLE_MF_FAILURES" fault-code="1"/>
  </parent>
  <child>
    <alarm-definition fault-family="AlarmSource" fault-member="ALARM_SOURCE_MF" fault-code="2"/>
  </child>
</reduction-link>
...
<thresholds>
    <threshold value="3">
      <alarm-definition fault-family="AlarmSource" fault-member="MULTIPLE_MF_FAILURES" fault-code="1"/>
    </threshold>
</thresholds>
```

References

- http://www.eso.org/projects/alma/develop/acs/OnlineDocs/AlarmSystem.pdf
- http://proj-laser.web.cern.ch/proj%2Dlaser/ (Laser documentation)
- http://lhc-cp.web.cern.ch/lhc%2Dcp/ (LHC project)

CDB schema:

ACS/LGPL/CommonSoftware/acsalarmidl/ws/config/CDB/schemas/AcsAlarmSystem.xsd

Examples in

ACS/LGPL/CommonSoftware/ACSLaser/demo

ACS alarm system

Questions ?

Exercise

Three components:

- PS: composed of three generators
- MOUNT
- ANTENNA

A failure in PS

- generates 3 alarms, one per generator
- then triggers a failure in MOUNT that sends another alarm
- a failure in MOUNT triggers a failure in ANTENNA that sends the last alarm

The triggering of failure is simulated by means of IDL methods (already implemented)

- Define the triplets for the tree generators, the MOUNT and the ANTENNA alarms
- Send the alarms (ACTIVE/INACTIVE) and check if they work in ACS implementation (log)
- Configure the CDB and check the alarms with the alarm GUI
- Define the reduction rule in order to see only the root cause of the problem

A possible solution

<PowerSupplies,ALARM_COURSE_PS,1>

<PowerSupplies,ALARM_COURSE_PS,2>

<PowerSupplies,ALARM_COURSE_PS,3>

<MOUNT,ALARM_COURSE_MOUNT,1>

<ANTENNA,ALARM_COURSE_ANTENNA,1>