European Organisation for Astronomical Research in the Southern Hemisphere

Atacama Large Millimeter/submillimeter Array

ESO ALMA Support Centre

Integrated Alarm System Architecture

Document Number: ESO-293482

Document Version: 1

Document Type: Design Report (DER)

Released On: 2017-02-22

Document Classification: Public

Prepared by:	Caproni, Alessandro
Validated by:	Schmid, Erich
Approved by:	Schmid, Erich
	Name

Authors

Name	Affiliation
Alessandro Caproni	ESO Directorate of Engineering

Change Record from previous Version

Affected Section(s)	Changes / Reason / Remarks

Contents

1 Introduction

1.1 Scope

This document defines the architecture of the Integrated Alarm System (IAS), which is being developed as an EU funded ALMA Development Project.

1.2 Definitions, Acronyms and Abbreviations

This document employs several abbreviations and acronyms to refer concisely to an item, after it has been introduced. The following list is aimed to help the reader in recalling the extended meaning of each short expression:

ALMA	Atacama Large Millimeter/submillimeter Array
ASCE	Alarm System Computing Element
BSDB	Back Stage DataBase
DASU	Distributed Alarm System Unit
ESA	European Space Agency
ESO	European Southern Observatory
GUI	Graphic User Interface
IASIO	Integrated Alarm System Input/Output
IAS	Integrated Alarm System
JAO	Joint Alma Observatory
JVM	Java Virtual Machine
LTDB	Long Term DataBase
RDB	Relational DataBase
TF	Transfer Function

2 Related Documents

2.1 Applicable Documents

The following documents, of the exact version shown, form part of this document to the extent specified herein.

AD references shall be specific about which part of the target document is the subject of the reference.

AD1 <Document name>;
<Document number Version X>

2.2 Reference Documents

The following documents, of the exact version shown herein, are listed as background references only. They are not to be construed as a binding complement to the present document.

RD1 Integrated Alarm System for the ALMA Observatory;
ESO-287159 Version 1

3 Introduction

The ALMA observatory is composed of many hardware and software systems like the Array, the cooling system, the power plant and so on. Each of these systems must be correctly functioning to ensure the maximum efficiency of the site. At the present, operators in the control room, as well as engineers sitting at their desks, follow the operational state of the observatory by looking at a set of non-homogeneous panels. In case of problems in one of the systems, they have to find the reason by looking at the right panel or log file, interpret the information and implement the proper counter-action. The procedure to recognize a problem and start the counter-action is therefore not optimal: for that reason, we have investigated the actual situation with the collaboration of an alarm system expert of the European Space Agency (ESA) and produced a report of the actual situation [RD1].

In [RD1] we have found that each monitored system has a hierarchical structure that can be modelled with an acyclic graph whose nodes represent the components of the system (see the right side of the graph in Figure 1). Each node, or component, of the monitored system can be working properly or be in a non-nominal state. In the latter case, the error could or could not generate an alarm to catch the attention of the operator or engineer. In fact, depending on the particular operational phase, an error could be safely ignored without distracting the operators. This is for example the case of failure generated by a non-operational antenna during the maintenance. This shows that having an error does not correspond 1-to-1 to an alarm: the monitor points in input to a component must be elaborated against a user provided heuristic to decide case by case if a non-nominal value in one or more of them is enough to produce an alarm for the operator. The model graph in the right side of Figure 1 shows the nodes that are working well in green and those in a non-nominal state in orange or red. Such information can be used to map the information in the model in the panels for engineers and operators as shown in the left side of the same Figure 1.

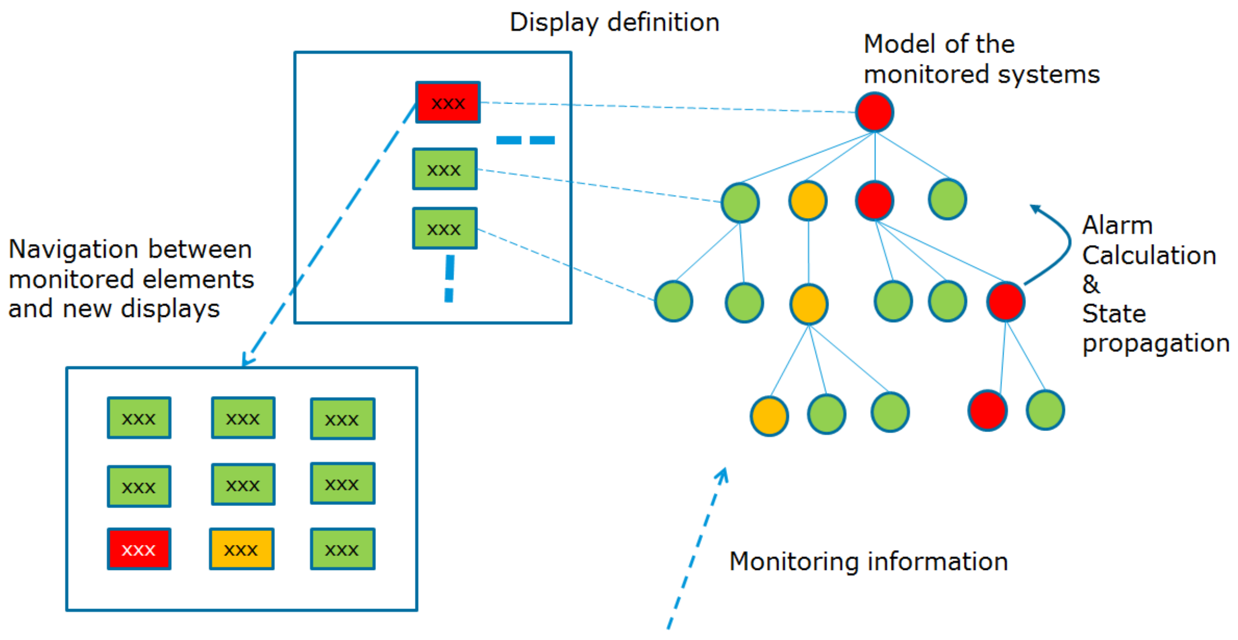
Display definition
Model of the monitored systems
xxx
Navigation between monitored elements and new displays
Alarm Calculation & State propagation
Monitoring information

Figure 1 The schema of the alarm system from [RD3]

In the scope of the report in [RD1], a study of the actual situation in the ALMA control room has been carried out (see Appendix 1 of [RD1]) to identify the major problems of the actual alarm system and to draw the road map for the development of a new alarm system called *Integrated Alarms System (IAS)*; integrated because it will show in a centralized way all the alarms coming from different monitored systems to increase the operator situational awareness and increase the overall efficiency of the observatory.

More formally, the Integrated Alarm System is a distributed software system whose main purpose is to get alarms and monitor points from different sources and generate alarms to present to the users, who can range from operators in the ALMA control room, to engineers sittings at their desks at the Executives. The alarm system is a message passing facility that routes abnormal situations detected from hardware or software to the user in a uniform and an easy to understand format. The purpose of such tool is to increase the situational awareness of the audience and help understanding the root cause of a chain of problems in order to minimize the reaction time and improve the overall efficiency of the facility.

4 High level picture

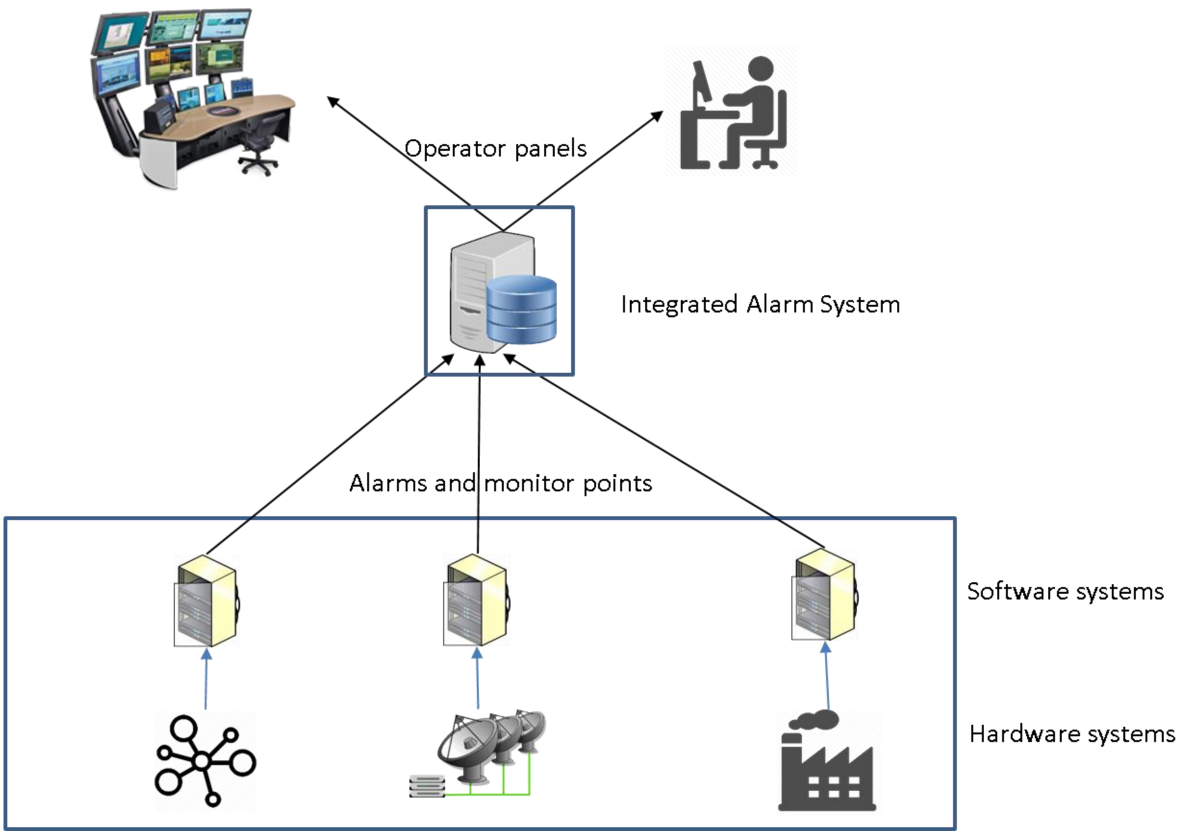

Figure 2 High level picture of the Integrated Alarm System.

The inputs of the IAS come in the form of values from monitor points like a temperature sensor, or alarms generated by other alarm sources such as specialized software systems like for example the ALMA Common Software (ACS) or the control system of a power plant. The set of inputs is therefore very heterogeneous: the IAS must be able to elaborate each type of input independently of its format and the software system who provides it. In the scope of this document, we will call the input to the IAS *Integrated Alarm System Input/Output (IASIO)*, regardless if they are the values of monitor points or alarms, and without distinction of the software source that produces them. Figure 2 shows, at the bottom, three hardware systems, the network, the ALMA array and the power plant with their software systems. The software systems monitor the hardware and route monitor point values and alarms to the IAS that, after a proper elaboration, sends alarms and status information to the alarm system.

The difference between the IAS and a traditional alarm system is that latter is tailored to a specific software system while the IAS deals with alarm sources and monitor points generated by different software systems. A traditional alarm system does what the IAS does when all its the inputs are produced by the same software system and, as such, the IAS is an alarm system extended to get values from heterogeneous sources and presents them to the users in a homogeneous format.

The IAS reads all the IASIOs and, if an abnormal situation is detected, generates one or more alarms that will be shown to the audience by means of a set of GUIs. The GUIs show the state of the system at a given point in time. A GUI layout with a consistent look and feel is very important to let the operator realize immediately what a root cause of a problem is and how to fix it. For that reason, we organized a workshop (in Appendix 2 of [RD1) with the participation of all the relevant stakeholders to sketch the most important panels for operators and engineers. A main panel, always visible in the control room, shows the overall state of all the various monitored systems. In case of a problem the user can click on the relevant area to drill into the details of the alarms by opening other sub-panels. Some of these sub-panels can show the geo-location of devices or a schematic representation of the monitored hardware. A flat table of the alarms and monitor points of the monitored system will also be provided.

This document describes the architecture of the IAS software based on the studies described so far.

5 Integrated Alarm System Building Blocks

This section describes the main building blocks of the Integrated Alarm System, their core functionality and how they are used to model the monitored systems and subsystems.

5.1 DASU

The core of the IAS is a distributed software system composed of *Distributed Alarm System Units (DASU)* that concurrently evaluate the IASIOs in input and, if appropriate, produce one or more alarms. Sometimes when the translation of IASIOs into alarms is very complex or a number of IASIOs must be correlated, the DASU produces an intermediate value instead of an alarm. We call such temporary value a synthetic parameter. The number, configuration and deployment of DASUs depends on the systems to monitor and is a choice of the IAS administrator. Typically, a DASU represents the IAS model of a particular subsystem of the observatory. For more complex subsystems, it can also make sense to break them down into a hierarchy of DASUs, most likely following the natural hierarchy of the subsystem.

The output produced by a DASU being an alarm or a synthetic parameter can be, in turn, the input to another DASU.

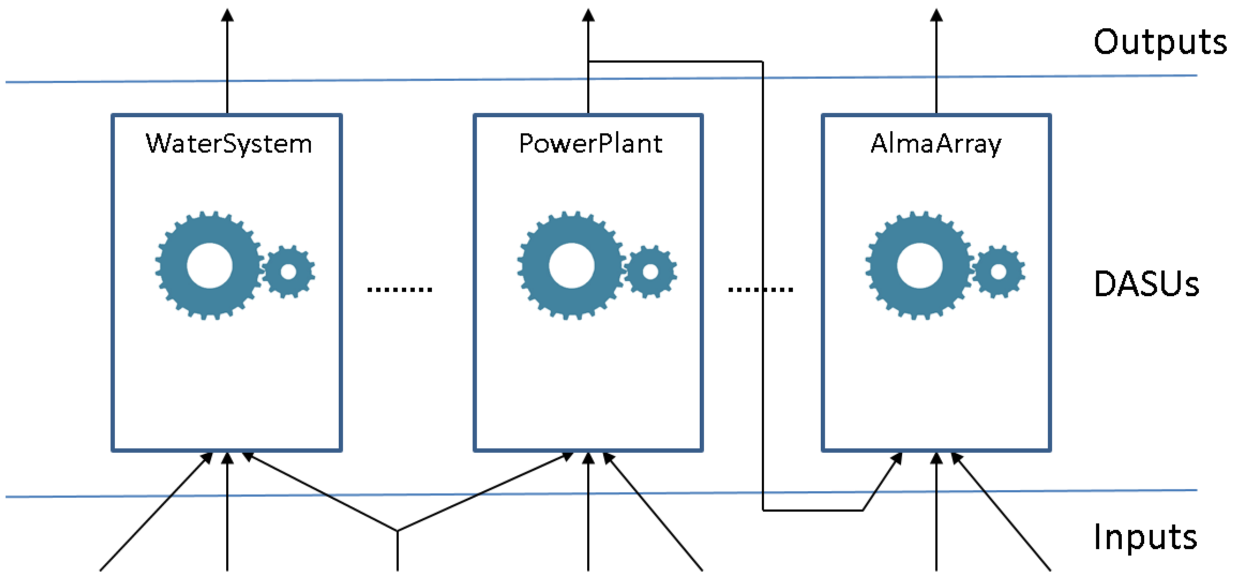

Figure 3 The DASUs collaboration diagram.

This design follows the principle that the software systems that produce the IASIOs in input to the IAS can or cannot be completely separated. For example, (see Figure 3), the Power Plant is completely independent of the Water System. Suppose that the DASU in the middle represents the entire Power Plant and DASU on the left side the Water System. The output of the Water System DASU can show an alarm if the level of the water in the tanks is too low. The Power Plant is not affected by the level of the water so the relative DASU does not show an alarm. If it makes sense, the same IASIO can be part of the inputs to more DASUs as shown in Figure 3. Suppose that the Power Plant takes the water for cooling from the Water System. In such a case the level of the tank can be one of the inputs of the Water System DASU as well as one of the inputs of the Power Plant DASU.

On the other hand, if there is an alarm in the Power Plant, it could affect the ALMA Array, the DASU on the right side, so the output generated by the Power Plant DASU is one of the inputs of the ALMA Array DASU; depending on internal logic the alarm from the Power Plant can or cannot trigger an alarm in the ALMA Array. In the control room, there could be a panel showing one box for each DASU where the alarmed ones are represented in red. By clicking on a red box, the user is presented with a more detailed panel to analyse the root cause of the alarm.

The example above suggested to associate one DASU to each of the monitored software systems (as described in Figure 2) but it is not the only possible way to use the DASUs. The number of DASUs and their interconnections represents a convenient decomposition of the real system allowing to model the observatory so the definition of the DASUs is a task of the alarm system administrator. Other examples of DASUs are modelling of a device, or an entire antenna.

In case the IASIOs in input to a DASU are produced by an external software system, they must be converted to the proper format before being processed by the DASU. The IAS will provide specialised software components, called plug-ins, for each connected system.

A DASU runs inside a Java Virtual Machine (JVM) and it is possible to deploy more than one DASU in the same JVM. To reduce the network traffic or improve performances, it can be useful to deploy a DASU close to the sources of its inputs but due to the network topology and the nature of a software system it is not always possible. For example, to avoid performance degradation during observations it might not be allowed to deploy software that gets the values of the monitor points by polling the components of the running system. Other control systems are closed i.e. do not allow to start additional processes at all.

5.2 Alarm System Computational Element

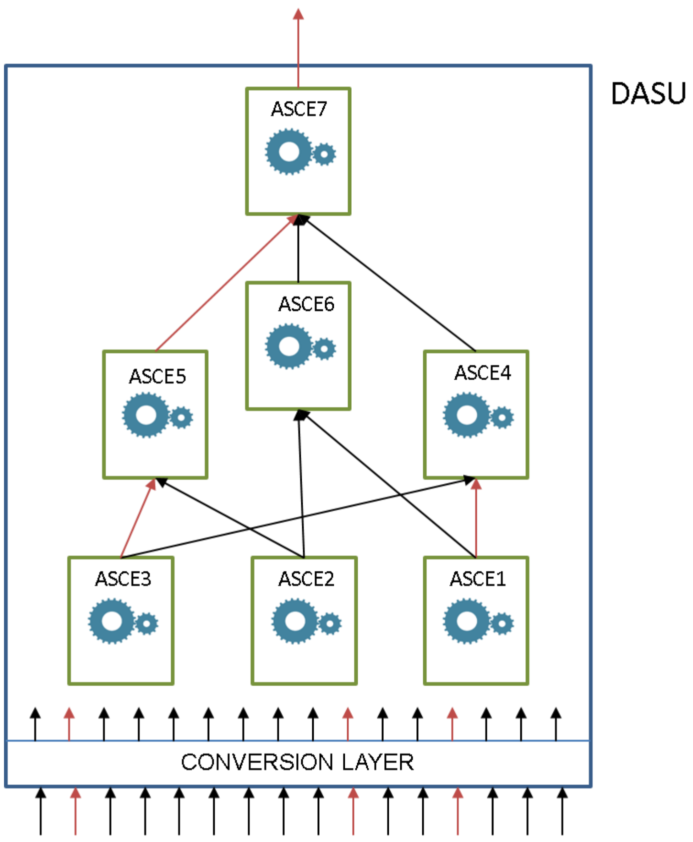

Figure 4 The Alarm System Computational Elements run inside a DASU

Each DASU is composed of one or more Alarm System Computational Elements (ASCE) that are the software components that perform the evaluation of the inputs to produce, in output, alarms or synthetic parameters.

The DASU in Figure 4 is composed of seven ASCEs. At the bottom the arrows represent a set of inputs coming from different heterogeneous sources. As we already said, they can be monitor point values or other alarms some of which are currently active and represented in red. If needed, the inputs are converted to the proper data structure by the conversion layer before entering the DASU: converted alarms and monitor point values are the only information circulating through the various parts of the IAS. The conversion will be performed by dedicated plugins that interface a remote alarm system or monitor point source to the IAS. The conversion layer in Figure 4 is therefore a logical representation of the conversion itself as it can happen in other parts of the IAS, i.e. not necessarily in the DASU itself.

Converted inputs are routed to the proper ASCEs (not shown in the picture) by a simple subscription mechanism. Each ASCE applies specialized heuristics to its inputs to produce an output that, in turn, can be the output of the entire DASU (ASCE7) or the input of other ASCEs.

The output of ASCE2, for example, is the input, possibly together with other IASIOs, of ASCE5 and ASCE6. The output of an ASCE can also be the input of another component running in another, possibly remote, DASU (not shown in the picture). The output produced by each ASCE is made available to all other ASCEs running or not in the same DASU.

Each ASCE has a rule to transform its inputs into an alarm or a synthetic parameter (output). As an example of a single input, think of an ASCE having a temperature as its only input. The rule to transform the input into an alarm can be to check the value of the temperature against a threshold and generate an alarm if it is too high. However, the case of a greater number of inputs is more interesting because the rule correlates many inputs to generate the output. For example, suppose to have an ASCE with two inputs, a temperature and the status of a fan: now the ASCE raises an alarm if the temperature is over the threshold and the fan is not running; in all other cases the alarm (in output) is not raised. As such an ASCE can decide that an alarm received from one of its inputs is not relevant and not propagate it to the output. It is the case of the ASCE4 in the picture.

Not propagating an alarm ultimately means that the alarm itself does not need to be announced to the operators because no action needs to be done. Suppose for example to have an alarm because one of the interlocks of an antenna is set. This alarm normally needs an action because the antenna is not operational and cannot be used for observing. But the same interlock alarm can be safely ignored if the antenna is in maintenance or not part of an array as it does not affect the observation.

In this context, we call the heuristics, or rule to transform the IASIOs into the output of an ASCE a Transfer Function (TF), in analogy with neural networks where a neuron transfers the values of all its inputs to a single output. More formally, we can say that the output *O* of an ASCE *E* with *n* inputs, $I_1 \ldots I_n$, is the result of applying the transfer function $\sum$ to its inputs:

$$O_E = \sum_{i=1}^{n} I_i$$

The heuristics of the transfer function of each ASCE is provided by the operators or the engineers that have a deep knowledge of the system and coded by the alarm system administrator. A set of TFs for the most common and generic cases like the threshold presented before, will be provided by the IAS. For each ASCE the IASIOs in input and which TF to use are defined in the Configuration Database (CDB).

For performance reasons, the TF must be compiled into JVM byte code. The association of TFs to ASCEs is defined in the Configuration Database, where each ASCE gets assigned the name of a Java class that provides the TF. The code of the class implements a well-defined interface with a method to execute to produce the output from the actual inputs. The only parameter passed to TF is a map of <*ID, IASIO*> that can be used to get the IASIO in input with the *ID* identifier.

The entry of an ASCE in the Configuration Database contains the name of this user provided Java class: the ASCE loads the class during the bootstrap. The configuration of an ASCE might also contain a user defined set of properties, as required by the used TF, in the form of <*name, value*>, like for example the threshold to raise an alarm if a temperature is too high. Such properties allow the reuse of the same TF in more than one ASCE.

The connection of ASCEs is acyclic. The number and connection of ASCEs allows to model the hierarchy and interdependencies of real equipment as shown before in Figure 1. Modelling a piece of equipment is the task of the alarm system administrator and the engineer responsible for such a device. The number of ASCEs to deploy in a DASU, their IASIOs, their connections and the rules to transform inputs to outputs is entirely configurable.

A central concept of this architecture is that the heterogeneous inputs coming from outside of the IAS, after being converted, and the outputs generated by ASCEs and DASUs have the very same type i.e. inside the IAS they are absolutely indistinguishable and treated in the same way.

The output of each ASCE is updated at a given time interval that is the refresh rate of the output it produces: the changes of the values of the IASIOs can be processed immediately or stored in a temporary data structure and evaluated when the time interval elapses.

5.3 Transfer Function Use Cases

The same Transfer Function is reusable by any number of ASCEs. For example, the TF to raise an alarm if a value passes a given threshold can be used for different monitor points ensuring that the proper threshold is set in the Configuration Database.

An example helps to clarify this concept. Suppose that we want to write a generic TF to check if the value of one single IASIO in input is in the operational range *[min, max]* and generate an alarm in output if the value is lower than *min* or greater than *max*. The algorithm of the TF is pretty simple and limited to few *if-then* statements. In the configuration of an ASCE we insert the name of the class (TF) we just wrote, the ID of the IASIO in input and the ID of the IASIO to produce and we set two user properties with the *max* and *min* thresholds needed by the algorithm. To reuse the same TF in another ASCE, it is enough to associate to it the proper values of the thresholds.

A frequent use-case for reusability in alarm systems is given by the modelling of *n* identical devices. As an example, suppose that one ASCE is enough to model one of such devices, as this case is easy to extend to more complex situations. As always, the CDB contains one entry for each of the *n* identical devices (or ASCEs). The TF to run is the same for each device so the CDB contains the same TF class name of the devices to model. Since the devices are identical, any property we might need to set is replicated for each ASCE, but we also set in the CDB an additional property with the sequence number of the device we are modelling like for example <*devNumber,0*>, <*devNumber, 1*>, and so on.

The output of each identical ASCE could have an ID string ending with the sequence number of the device like for example *PWSUPPLYn*. The input of each ASCE is the same in all the devices but their ID ends with the number of the device. So for example the device *0* produces the output *PWSUPPPLY0* and get as inputs for example *OVERUCURRENT0*, *ENGINETEMP0*, *FANRPM0*. In this way the TF of each ASCE accesses the IASIOs in input by appending their IDs to the passed device number. As this kind of replication is almost entirely provided by the CDB we can then implement some mechanism to reduce the entries in the database for such cases. For example, the administrator defines the inputs and output of only one device (template) from which the replicated *n* configurations for the *n* devices are automatically generated by a software procedure that ensures that the pattern described upon is correctly implemented.

There will also be a need for more elaborate schemas to identify identical devices. E.g. the ALMA antennas use four letter codes, *CM01-CM12, DA41-DA65, DV01-DV25* and *PM01-PM04*, where a simple sequence number clearly will not work. The IAS will therefore also provide the ability to provide user defined schemas when creating such devices from a template.

Another frequent use case is the dynamic relocation of components. In ALMA the classic example is given by the antennas belonging to arrays: an antenna can be assigned to an array, moved from one array to another or taken out of an array. One possible solution to model an ASCE for a given array *A* that sends an alarm if one of its antennas is not operational is to define one input (a boolean *operational/not operational*) for each possible antenna in the system and one input *N* with the names of the antennas belonging to the array (an array of strings for example). The array's TF checks only the inputs of the antennas in *N* and ignores the others. If one antenna is removed from the array by the operator, its name will be removed from *N*: the TF will then refresh the output discarding such an antenna. The same ASCE can be replicated for many arrays with the technique we described before.

5.4 Putting all together

Figure 5 summarizes the concepts presented so far. The IAS runs on two servers with a total of five DASUs each of which contains three computational elements (ASCEs). In *Server 1* both DASUs run inside the same JVM while in the *Server 2* one DASU runs in one JVM and the other two in the other.

A DASU is responsible to collect the inputs, forward them to the ASCEs it contains, collect the output of each ASCE it contains, and propagate them to the other DASUs and the internal ASCEs.

The ASCEs' main task is to evaluate the inputs and generate the output applying the user provided Transfer Function. This freedom gives great flexibility, but user errors in the Transfer Function could endanger the IAS at run time, so parameters like execution time must be constantly monitored at run-time. Each TF must be tested before being used in production. A simulator will be provided for testing purposes.

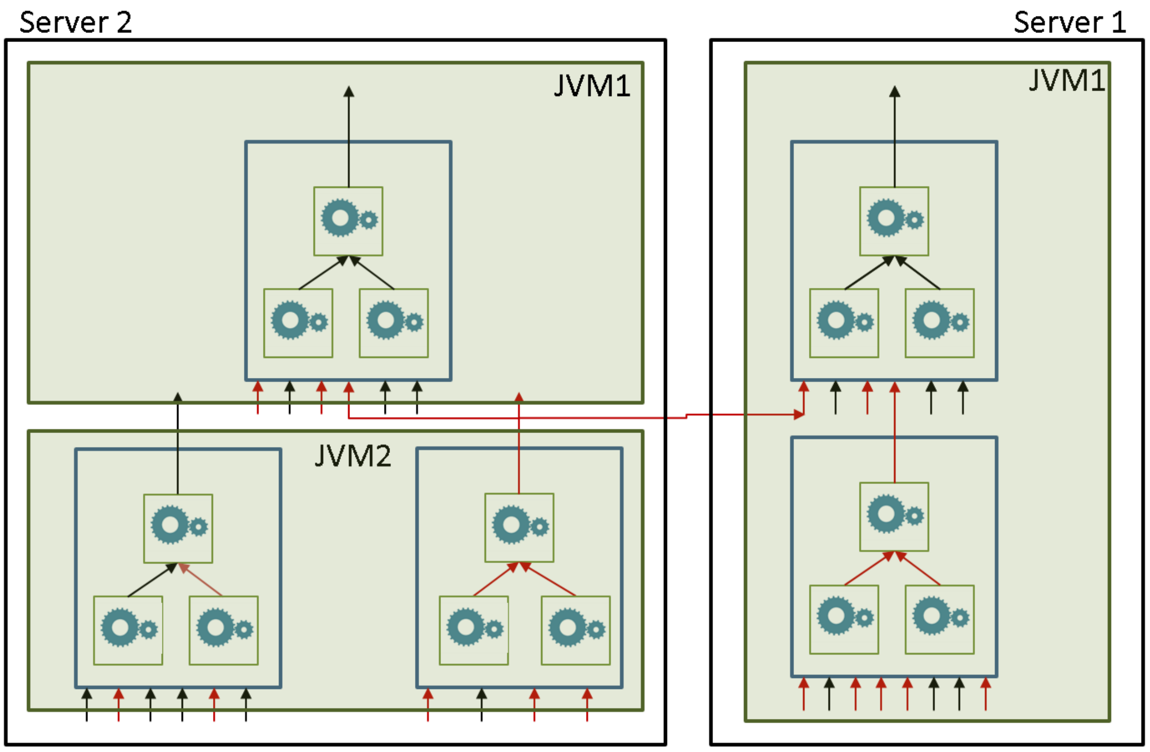

Figure 5 IAS deployment overview

5.5 A modelling example

The following artificial example clarifies the usage of DASU and ASCEs and is deliberately not related to ALMA because the IAS is generic and does not depend on the telescope. We have a little imaginary diesel power generator, like the ones we often see when grid power is not available (Figure 6), and assume that it is possible to read some variables of this device through a software interface. We want to see an alarm in our display if something is wrong with the power generator, so that we can understand what the problem is and fix it in a short time avoiding a blackout.

Figure 6 A diesel power generator.

The table below shows the monitor points for this device and the identifiers we associate to each of them. In this example, we focus only on the ASCEs and DASU and not on the conversion of the values of the monitor points to IASIOs as well as the propagation in the alarm system.

Nr.	ID	Description
1	*ENGNOTRUNNING*	An alarm set if the engine is not running
2	*FUELLVL*	The available fuel. If <5 we must refuel.
3	*OILQTY*	The quantity of oil in the engine. If <3 we must add oil.
4	*FAN*	A Boolean set to `true` if the if the cooling fan is running
5	*RPM*	RPM of the engine. If greater than 5000 we must shut down the engine
6	*TEMP*	If the temperature is greater the 95 we must shut the engine down
7	*220VACFREQ*	The frequency of 220 AC power must be in [45,55]
8	*220VAC*	The voltage of the 220 line must be in [215,225]
9	*12DC*	The voltage of the 12 DC current must be in [11,13]
10	*MAINT*	A boolean that is `true` if the device is in maintenance (refuelling, refilling the oil, tuning the currents…)

FAN has no constraint as the fan runs only when the temperature is high. *MAINT* signals that maintenance is on-going: no alarms are published during maintenance if the engine is not running or the current is out of the specifications.

Such a device can be modelled with only one ASCE with ten inputs. The output, an alarm, is set if one of the conditions in the table is violated and no maintenance is on-going. With this implementation when the output is set, we know that something is wrong, but we have to check the values of the inputs to understand what was wrong to fix the problem. The panels display all the values of the IAS in the system but they are not allowed to make any computation. This is the task of the alarm system.

So if there are only 2 litres of gas in the tank, the output of the ASCE is an alarm because the minimum threshold of *FUELLVL* is 5. The GUI panel shows the alarm generated by the ASCE and possibly the actual values of the inputs. The operator must then check all the values and understand what the problem is. For this simple example with only ten inputs it is clearly not a big deal, but it is not feasible for a device with hundreds of correlated monitor points: finding the reason of a problem could take too much time, especially if there is a cascade of alarms. Also, to develop a TF that deals with all the possible combinations of hundreds of inputs is almost impossible.

If we look deeply into the structure of the device and the type of actions for the operator in case of problems, we note that there are four main classes of problems, and their corrective actions:

- Problems with the engine not working properly (*RPM, TEMP*): immediate shut down
- Supply problem (*OILQTY, FUELLVL*): add oil and/or fuel
- Quality of 220 Volt production (*220VACFREQ, 220VAC*): disconnect 220V appliances
- Quality of 12 Volt production (*12DC*): disconnect 12Vappliances

A possible decomposition of this device is represented in Figure 7. The big green box represents the DASU where seven ASCEs run. At the bottom, the IASIOs in input are represented with white boxes. Each ASCE is drawn as a blue box with the name of the output it produces. Inputs are connected by black lines while blue lines show the connection of the outputs. The graph is acyclic.

This is how each output is calculated i.e. the functioning of the TF:

- *LOWOIL* and *LOWFUEL*: the TF checks the available quantity against the threshold and sets an alarm if the value is too low.
- *HIGHTEMP*: this is a synthetic parameter, a Boolean, calculated by checking the actual temperature *TEMP* against two thresholds (*t1* < *TEMP* <*t2*) and the status of the fan (*FAN*); if the temperature is greater than *t1* and the fan is not running or is greater the *t2* then the output is *true* in all other cases *false* (i.e. if the temperature is greater than *t1* but the fan is working we expect the temperature to decrease quickly)
- *ENGFAIL*: an alarm is set if it is not running, *RPM* is too high or *HIGHTEMP* is true
- *220CUR*: checks if the *220AC* current is in the nominal range, but only if the engine is running
- *12CUR*: checks if *12DC* current is in the nominal range, but only if the engine is running
- *PWGEN*, the entire generator: is alarmed if a) there is not enough oil or fuel or b) if there is no maintenance on going and there is an alarm in the engine or in one of the currents.

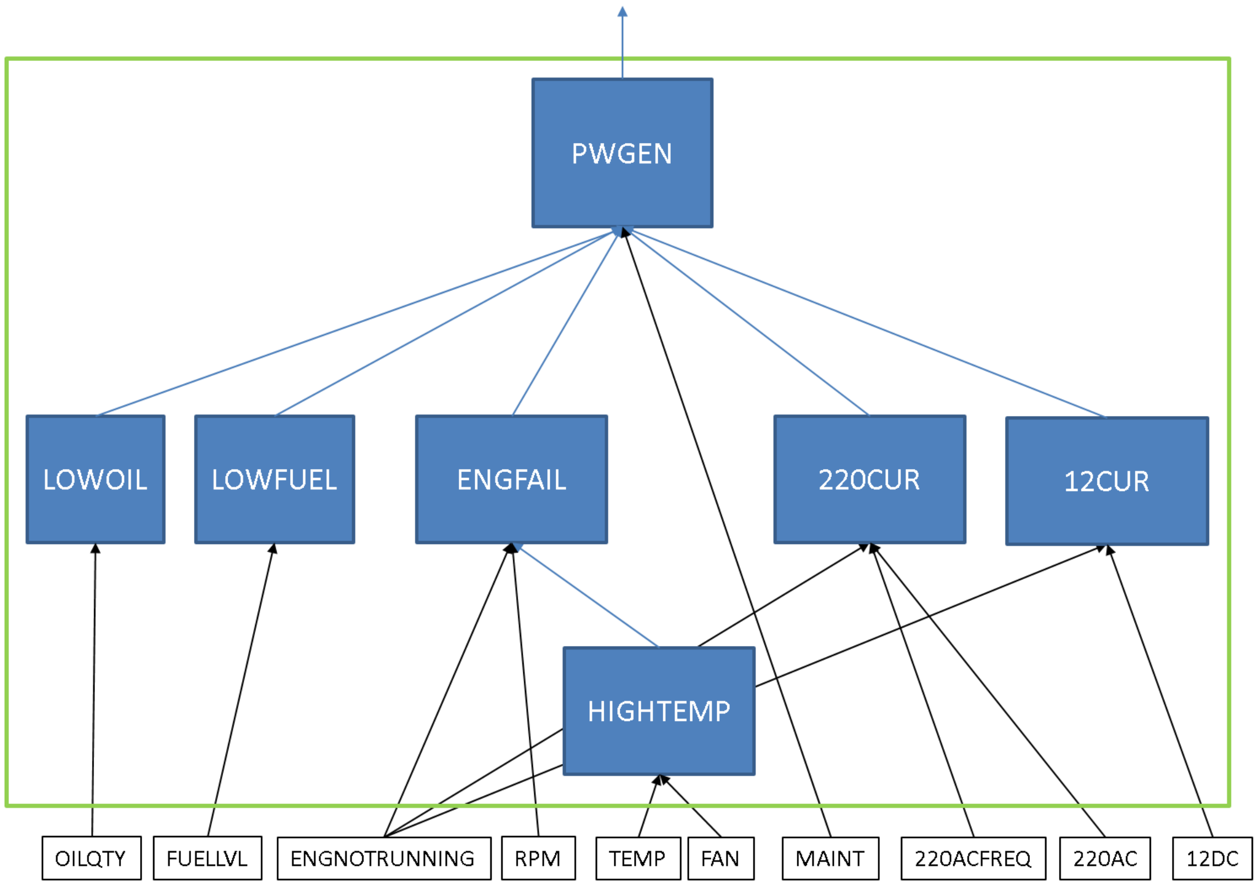

Figure 7 Example of ASCEs for the power generator.

This decomposition is only to show the mechanism we put in place, there can be many other decompositions of course. In reality, the connections of ASCEs will resemble the dependencies between the components of the devices. In this case the device is very simple, but enough to explain the main concepts: it shows a synthetic parameter *HIGHTTEMP* that correlates two inputs to produce a Boolean value in input to *ENGFAIL*. It shows also how the output of an ASCE can be the input of another ASCE. Each ASCE has its own TF even if *LOWOIL* and *LOWFUEL* clearly reuse the same algorithm, so there is no need to write the same function twice.

A GUI panel for this case could show the *PWGEN* only. If it is in alarm, it changes its colour to red. If the user clicks on it, the panel shows the other ASCEs with the alarmed ones in red, so operators will immediately understand what is failing and how fix the problem. The values of the ten IASIOs in inputs can be displayed on demand: if the *ENGFAIL* is alarmed and the *HIGHTEMP* is alarmed, the user can see the actual value of *TEMP*.

6 The Core of the IAS

The core of the IAS is composed of the DASU and ASCEs we already saw together with other components like the Configuration Database (CDB) and the Back Stage Database (BSDB), and the IASIO data structure.

All these components collaborate to evaluate the inputs provided by the remote software systems against the model and defined rules, and ultimately generate a number of alarms, either set or cleared. Clients of the IAS, like operator GUIs, are notified of these alarms and display them together with additional context information, such as the values provided by the monitored components.

6.1 The Integrated Alarm System Input/Output (IASIO)

The IAS architecture is based on the uniformity of the monitor points and alarms produced by external subsystems (after a proper conversion) and alarms and synthetic parameters produced by the IAS itself.

Values received from external systems are represented by triplets in the form

```
<identifier, timestamp, value>
```

where

- `identifier` is the unique identifier used in the IAS. It is not the identifier used by the external software system. It could be composed of the external identifier and the name of the remote software system that produced it.
- `timestamp` is the point in time when the value has been produced[1].
- `value` is the actual value of the monitor point or alarm: it can be a numeric value, an array of values, a bit mask, a pattern, an alarm or something else.

These triplets need to be converted into an IASIO data type before being used by the core of the alarm system.

6.1.1 Validity of an IASIO

The IASIO enriches the triplet by adding properties like the expected refresh rate of the value taken from the configuration database, to assess its validity. The validity tells if the value has been refreshed within the expected time interval or the new value never arrived, for example because of a network problem. The purpose of the validity is to show to operators and engineers that a value displayed in a panel may not be up-to-date by using a dedicated colour coding. The validity is a property of an IASIO stored in a particular ASCE. The ASCE calculates the validity at regular intervals and updates the respective property of the IASIO.

With respect to validity, there are two possible scenarios:

- if the IASIO holds a value produced by a remote subsystem: the validity is calculated by comparing the actual time with the timestamp of the last received value, i.e. the value of the triplet received from the remote subsystem
- the IASIO holds an alarm or a synthetic parameter produced by a ASCE: the validity is the minimum validity of the inputs of the ASCE

6.1.2 Updates of IASIOs

To reduce network traffic and to avoid potential interference with external systems, we do not want to poll remote subsystems to get alarms and monitor points out of their control software. As previously described, each subsystem sends the data

[1] If needed, the triplet can be enriched with more timestamps to track for example the point in time when it has been received or processed by the IAS. Such information can be useful for debugging or monitoring the processing time of IASIOs from the instant they have been produced to the moment they generate an alarm.

- on change[2], or
- at regular time intervals.

If the subsystem does not provide the functionality out of the box, it must be implemented by specialized software accessing the control software by the API it provides or with some other strategy if no API is available. This effort is part of the integration work necessary for each monitored system. The scope and approach depends entirely on the interfaces provided by the external systems.

On input each value is converted into an IASIO ready to be digested by the ASCE. The IASIOs have an entry in the Configuration Database with at least the data type of the value (e.g. alarm, integer, double), the expected refresh rate, and optionally a tag that is human readable name of the IASIOs.

6.2 Identifiers

IASIOs, Computational Elements and DASUs all have an identifier that allows to uniquely identify them at run time. The identifiers are defined in the CDB and are composed of three parts:

ID	The unique identifier
parentID	The identifier of the parent
runningID	Stringified representation of the identifier

The *ID* uniquely distinguishes one object from another. The *parentID* is the unique *ID* of the parent, making the identifier a recursive data structure: the chain of *ID*s allows to quickly identify who owns an object and where it runs. The *runningID* is a stringified version of the identifier that consists of the *ID* plus the *ID*s of all its parents. Its purpose is to improve performance at run time simplifying an identifier comparison to a string comparison instead of a recursive call to all *parentID*s. The recursive data structure, and the related *parentID*, is very useful for debugging: it says for example which ASCE produced an IASIO and in which DASU it runs.

The *ID* of an IASIO is statically configured in the CDB. For example, it can be something like “FUELTANK0-LEVEL”, “ARRAY01-ANTENNA_DV05” or whatever else is meaningful with the only constraint that it must be unique. The *parentID* states the owner, or producer of a particular object. For instance, the owner of an ASCE is the DASU where it runs. This definition allows from one side to immediately recognize one specific item and, from the other, helps routing or locating elements inside the IAS.

The following table describes a possible assignment of parent IDs of IAS components whose final version depends on the final implementation:

IAS Component	**Parent**
Value from a remote software system	The ID of the remote software system
IASIO	• the ASCE that produced it • the plugin that generated the IASIO for the value received from a remote software system

[2] The number of allowed changes per time interval must be limited to avoid flooding the system with tons of notifications that operator could not follow.

DASU	Nothing
ASCE	The DASU where it runs
Conversion plugin	Depends on the final implementation: it could be part of a DASU or a standalone process

Monitor point values and alarms received from a remote software system have their own identifier. The ID uniquely identifies them in the scope of the IAS, but is not necessarily identical or even related to the identifier they have in their control software. However, it will often be convenient to reuse the same string.

For example, in the array control software the ambient temperature sensor of the water vapor radiometer of the DV16 antenna is

CONTROL/DV16/WVR/AMBIENT_TEMPERATURE

which is unique in the array control software.

In the IAS this could e.g. be mapped to *ID*:

AMBIENT_TEMPERATURE$WVR$ANTENNA_DV16

The '*$*' is a special character used by convention only to express a containment hierarchy in the identifier itself. It is particularly useful for replicated systems. E.g. every one of the 66 ALMA antennas has exactly the same ambient temperature monitoring point for its WVR. The IAS itself does not enforce such naming schemes, for its purposes the only requirement is that the ID is unique.

The *parentID* of this IASIO could be

ACS_NC

assuming that the IASIO is generated by a conversion plugin that listens to the ALMA Common Software Notification Channel.

The *runningID* of this IASIO would then be

AMBIENT_TEMPERATURE$WVR$ANTENNA_DV16@ACS_NC

The conventions described in the table above are subject to change depending on the selected distribution system and the final implementation. For example, in the case of a DASU it could be useful to set the parent to the name of the host where it runs. Figure 8 shows a possible example of identifiers in a DASU named ARRAY01. The strings in the picture are the *runningID* strings composed of ID of the item concatenated with the IDs of the parents separated by a \`@\` character.

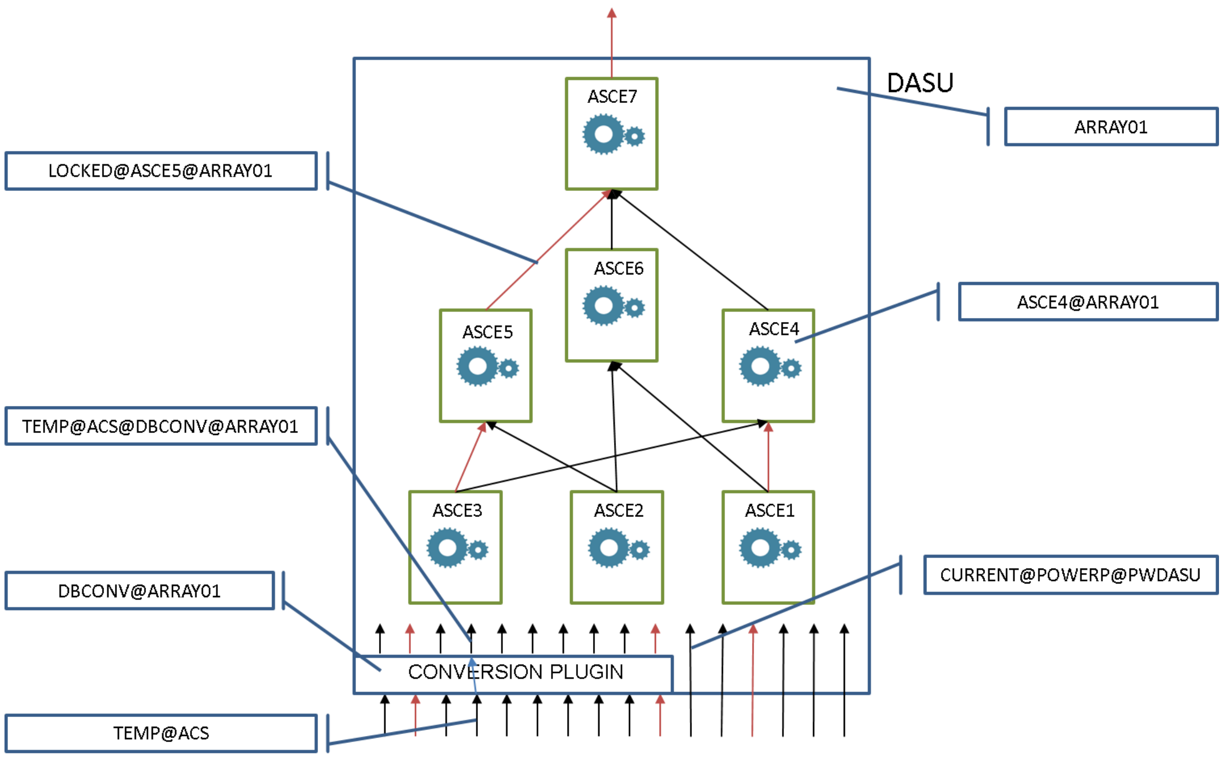

Figure 8 The identifiers.

6.3 Configuration database

The Configuration Database (CDB) stores the configuration of all the IAS components i.e. IASIOs, ASCEs, DASUs, conversion plugins, and the Transfer Functions. Each component is uniquely identified by its *ID* and has an optional name mainly oriented to human readability, e.g. for displaying an entity in a GUI panel.

The Configuration Database provides the deployment configuration for each DASU, the association of ASCEs to DASUs, the IASIOs in input to each ASCE and the output produced by ASCEs and DASUs. A relational database suffices for this purpose. It will be possible to define a CDB with text files for testing and to export/import to/from text files in the relational database.

A dedicated web or desktop application will allow to browse and edit the CDB. It also ensures the consistency of the inserted data, if such functionality cannot be directly implemented by RDB rules. It is desirable to have support for creating and editing replicated elements using templates from the GUI.

6.4 Interface with Other Alarm Software Systems: BSDB

The inputs of the IAS are the monitor point values and alarms produced by external software systems. As discussed in the DASU section in page 7, all such alarms and monitor points must be converted into the proper format before being propagated to IAS computing elements.

Since external software systems are very different from one another, it is not possible to foresee only one communication strategy that works for all the cases, but we will most likely need to develop a separate plugin for each different system. Such plugins can receive notifications from the external software systems about new alarms being set or cleared or changes in the value of monitor points. They can also poll external software system through their API to get the values. In both cases, monitor points and alarms must be sent to the IAS

- When their values change
- At regular time intervals

The latter informs the IAS that the values it owns are always actual and that the connection with the remote system is alive. As a secondary benefit, when the IAS is started it receives all the alarms and monitor points at the latest when the time interval elapses and without polling the remote subsystem for the actual values. Such refresh of alarms and monitor points is used internally by the core of the IAS to evaluate the "healthiness", or validity, of the value itself.

The proposed solution consists of a Back Stage Database (BSDB) to store alarms and monitor points produced by the external software systems: they are represented by triplets of the form

```
<identifier, timestamp, value>
```

as introduced in section 6.1 above.

Such triplets, coming from the remote software systems, are stored in the BSDB instead of being directly injected into the IAS, decoupling the external software systems from the IAS itself.

The IAS, in turn, gets the inputs, i.e. the triplets converted into IASIOs, from the BSDB instead of interacting with the other software systems. These IASIOs are routed to the DASUs and from there to the ASCEs, where their updated values are stored. At regular intervals each ASCE runs the Transfer Functions of all its inputs, checks their validity, and produces its outputs, i.e. other IASIO elements, accordingly. The output produced by ASCEs is made available to other DASUs and ASCEs to propagate the computation through the hierarchical graph modelling the real system. From what we just said, converted triplets and the outputs produced by the ASCEs must all be propagated to other ASCEs and DASUs, making them indistinguishable in the sense that the IAS treats IASIOs produced by ASCEs and those generated by converting the triplets in the very same way.

Figure 9 represents the logical view of this communication where three software systems feed data into the BSDB; the IAS, in turn, gets data out of the BSDB. In the final implementation, depending on the network topology and other factors, it might not be possible or desirable to deploy any software on external software system. In that case a specialized piece of software, running in one of the IAS servers, polls the data from the remote software system and feeds the database[3].

[3] We assume that the remote software system offers an interface to get data out of it.

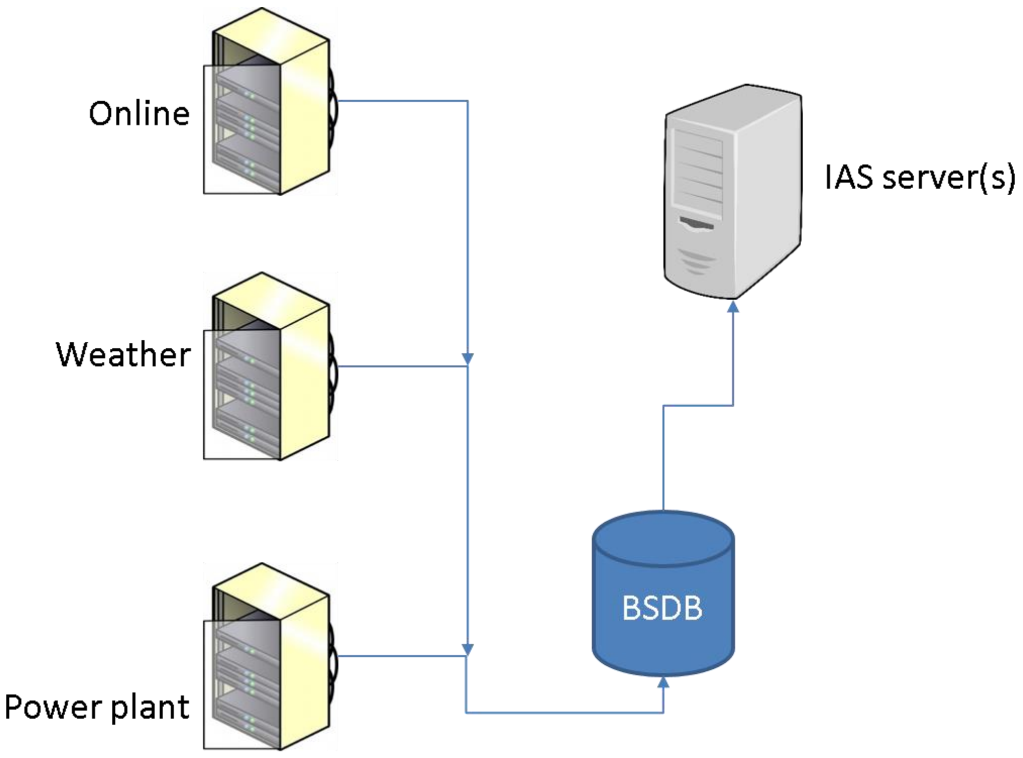

Figure 9 Back stage database

Figure 10 shows the data flow of the Integrated Alarm System. The external software system on the top left side produces monitor points and alarms in form of triplets that are sent (1) to the temporary queue *T*. This communication depends on the particular software system and the API it provides, but most likely a dedicated piece of software interfaces with the remote software system to get the values, formats them and sends the triplets to *T*. Such software can perform additional tasks on the monitor points like for example damping, or averaging. This tool is an IAS deliverable but it is not part of the IAS core: it matches with the concept of a plugin we described before.

A converter[4] extracts the triplets from the temporary *T* queue (2), translates them into IASIO objects and pushes them into the *IOs* queue. The converter may need to access the CDB for the translation of triplets into IASIOs. As we saw before the IASIOs are the IAS data type for monitor points and alarms and are the only data type that circulates in the IAS core. IASIOs contain more information than just the value extracted by the triplet, like for example the validity.

A publisher[5] finally gets the IASIOs from the *IOs* queue (4) and sends them to the Long Term database (LTDB) for permanent storage, and to the clients like the DASU and ASCEs, and technical GUIs, i.e. the GUIs for the alarm system administrator. An example of such GUIs are to define and manage the deployment of DASUs and ASCEs.

The number and deployment of the *T* and *IOs* queues depends on the framework adopted, but in principle there should be one *T* queue for each remote software system to

[4] Or a pool of converters.
[5] Or a pool of publishers.

reduce the load on the converters; for the same reason, more than one *IOs* queue could be deployed.

Figure 10 also shows a DASU with two ASCEs. The inputs of the ASCEs are IASIOs that come from the *IOs* queue (4). The output produced by ASCEs, of IASIO type as usual, are in turn stored into the *IOs* queue (3) to be propagated from there to other ASCEs (not shown in the picture). From the picture it is clear that triplets from external software systems after conversion and IASIOs produced by ASCEs are indistinguishable and processed the same way.

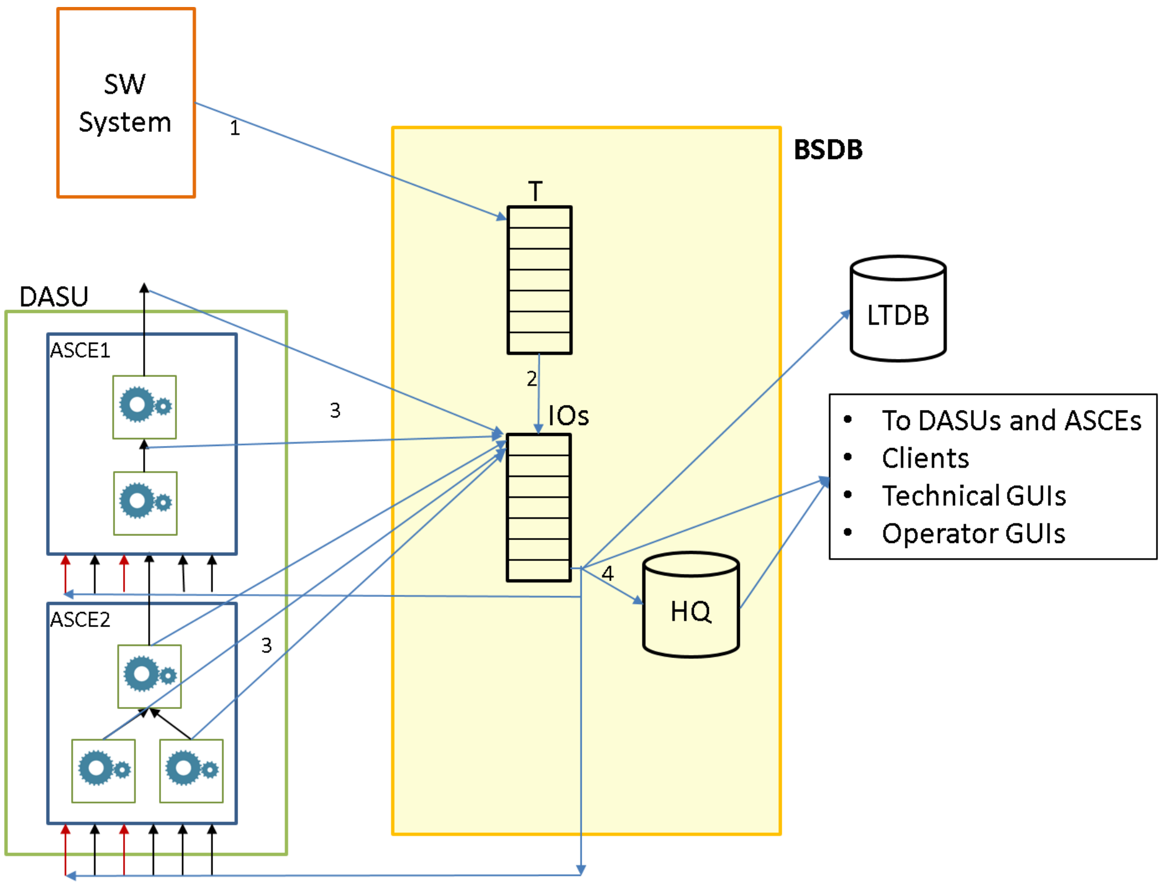

Figure 10 Data flow

Data extracted from *T* and *IOs* queues are immediately discarded: the standard way a client has to be informed about IASIOs updates is by subscribing to the events produced by the publishers attached to the *IOs* queue(s).

If needed, IASIOs can be stored in a Hash Queue (*HQ* in Figure 10) to be easily accessible by clients through their identifiers. Getting IASIOs from *HQ* correspond to polling and shall be avoided whenever possible. The reason to have the *HQ* is mostly justified for debugging and the deployment of *HQ* must be configurable in the Configuration Database.

The IASIOs in the *IOs* queue(s) have a unique identifier, *ID* as described in the "A modelling example" chapter on page 13. This unique identifier is a symbolic name. The complete identifier, composed of the *ID* of the IASIO plus those of its parent (from the *parentID* field) is useful for the internals of the IAS core as it contains, in its recursive data structure, information about the deployment that can be useful for technical GUIs. The *runningID* and the complete identifier with its recursive data structure are not intended to be used by clients and GUIs whose code must be independent of the deployment: they must use the unique *ID* of an IASIOs to refer to it. Some specialized tool for the alarm system administrators could still directly use the *parentID* of an IASIO to show the deployment.

The *ID* of an IASIO is a unique identifier; mostly a string targeted at ASCE computation, not necessarily human readable. It is not intended to bring information to operators or engineers. For that purpose, each IASIO must also have a unique, human-readable and possibly meaningful string that we call a tag. Such a string is intended to be displayed in the panels.

For example, the *runningID* of the *mean of the temperatures* of the *metrology* component of antenna *DA48*, a synthetic parameter calculated by *ASCE4* running on *DASU5* could be:

MEANTEMP$METR$DA48@ASCE4@DASU5

This is clearly a pretty cryptic string for an operator or an engineer, but brings a lot of useful information:

- the *ID* of the synthetic parameter,
- the *ID* of the ASCE that calculated the mean, and
- the DASU where it is deployed

The *ID* of this IASIO is *MEANTEMP$METR$DA48* but this does not say much so, even if it is useful for the IAS internals does not bring any valuable information to the audience.

In the configuration database it is the possible to associate a human readable tag to such an identifier, like for example "*Mean of metrology temperature sensors of antenna DA48*" that brings valuable information. The tag by itself is still not enough: full documentation of each IASIO, what it means and, if it is an alarm, how to fix an abnormal situation, must be available and quickly accessible for example in a Wiki page.

The fact that clients like GUIs are only notified when a value changes or the refresh rate time interval elapses, means that at the start-up the alarms and values displayed will be marked as invalid. They will become valid as soon as the first refresh for each item has been received.

To get the high performances required for the BSDB, we are intending to adopt an in-memory database like Redis or a fast file system database like Apache Kafka. Both have advantages and disadvantages and similar features including a publisher/subscriber paradigm and both fit with our needs. We will have to compare performances, evaluate the coding complexity and estimate pro and cons of a BSDB persisted on files against a volatile in-memory implementation to take the final decision. It could be needed to run different instances of the database to better partition the data and improve performances and reliability.

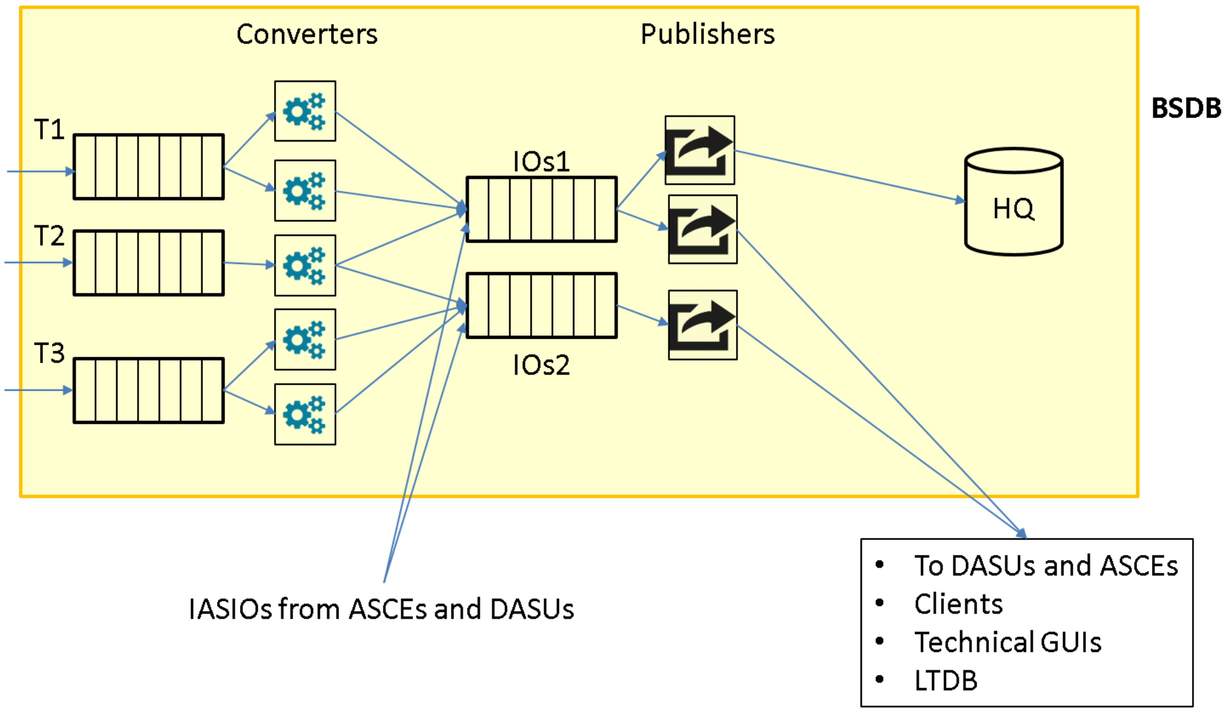

Figure 11 A zoom into the BSDB.

Figure 11 shows an example of a BSDB configured with three temporary queues to receive triplets from remote software systems, *T1*, *T2* and *T3*. *T1* and *T3* have two converters; *T2* has only one. Each converter polls triplets out of its *Ti* queue, converts the triplets into IASIOs and pushes the IOSIOs into the *IOs1* and *IOs2* queues. *IOs1* and *IOs2* receive IASIOs produced by ASCEs, too. The content of *IOs1* and *IOs2* is identical and replication is managed by the framework, i.e. the converters only write once. The reason to have more replicated queues is to improve performance or reliability. It could be convenient for example to have one queue close the web server applications so that communication happens locally. However, as simplicity is the main rule, one queue should be used unless it is proven to not be enough. In the example there are three publishers, one in charge of populating the HQ, while the other two publishers send the IASIOs to clients. Some of the frameworks for the BSDB provide a publisher/subscriber feature: in that case the publishers in the figure do not correspond to a running code.

The number of *T* and *IOs* queues as well as the number of converters and publishers depend on the load of the system. For example, it could make sense to have more converters associated to the *T* queue of a software system that delivers a great number of triplets.

Note that in the schemas in Figure 10 and Figure 11 all the components are grouped into the same box, BSDB. This does not mean that all the components of the BSDB run in the same server

The LTDB is the place to save everything produced by the IAS like logs, IASIOs, command history and so on. We expect this database to be stressed by many writes but not so much for reading; a possibility could be offered by Cassandra. Feeding the *IOs* queues with the IASIOs stored into the LTDB, it is possible to regenerate the state of the IAS at a given point in time.

7 GUIs

The IAS has to provide GUIs that provide operators and engineers with a clear understanding of what is going on in the system and actively help them fixing abnormal situations in the shortest possible time. The main difference between the engineering and operator GUIs is the fact that operators are allowed to interact with the alarm system, acknowledging or shelving alarms, while engineers are only able to view the current state of the system. Clearing of alarms is not possible as the alarms clear automatically when the abnormal situation has been fixed.

The panels display the alarms with a consistent colour coding so that it is easy to get the state of the system with a glance even from a far distance from the display. Invalid alarms will be represented in all the panels with the same colour in all the panels, the same will happen for active and inactive alarms. A newly activated alarm blinks to catch the attention of the operator to something that is changed and it will continue to blink until the operator explicitly acknowledges it. The operators in the control room are the only ones that can acknowledge a newly activated alarm: they have also to insert a comment. The acknowledgment and the comment will be recorded in the LTDB and the alarm will stop blinking in the panel.

If an alarm is active and acknowledged in the panel for a long time it can be disturbing and distracting from important new alarms. To mitigate such a situation, it can be shelved for a defined time interval. Shelved alarm will be masked in the GUI for a proper time interval before being un-shelved automatically. The purpose of shelving an alarm is to display an alarm as if it would be inactive for a time interval. A typical use case for shelving an alarm is when the action to fix a problem has been initiated by the operator but it requires long time before the problem is solved and the alarm automatically cleared in the panel. In this case shelving that alarm will remove it from the panel and the operator is not distracted by an alarm whose solution is already in progress. The shelving of an alarm lasts for a defined time interval to be sure that it is not forgotten forever. It is for example advisable to un-shelve alarms at the end of one shift to let the operator of the next shift aware of any problem in the system.

The ultimate design of each panel closely depends on the system to model. A general layout for the most useful panel has been drawn during a workshop [RD1]. The final GUI design will be elaborated from these initial design sketches.

To allow operators in Chile, as well as engineers at the Executives to access the alarms produced by the IAS, the most suitable solution is to have web interfaces served by a web server located close to the IAS having access to the IASIOs in the BSDB queue as of Figure 10.

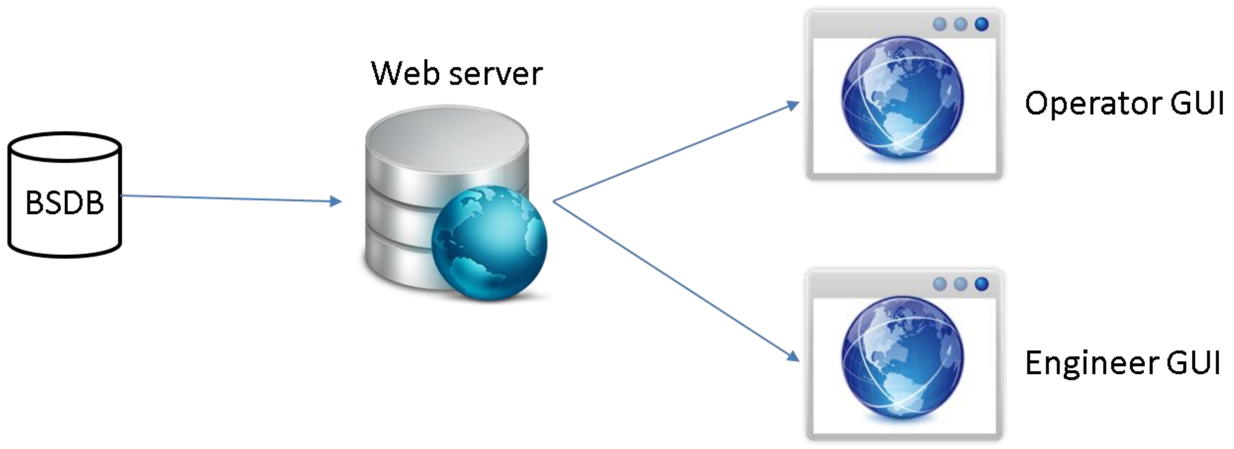

Figure 12 Operator and engineering GUIs.

Figure 12 shows the high level picture of the GUIs framework of the IAS. The IASIOs pushed into the BSDB are sent to an application running in the web server, which is in charge of forwarding the IASIOs to the web browsers, where an application draws the panels for operators or engineers.

Due to the implementation of the propagation of the IASIOs, there will be a delay between the moment the operator opens a GUI and the point in time when the panels receive all the IASIOs. During such time interval the IASIOs will be invalid. This is a not a problem considering that alarm panels stay open most of the time, and the relatively short refresh intervals for most of the data. To mitigate the delay, a panel could forcibly poll the web application on the server for an immediate refresh.

The IAS will provide a certain number of technical panels to be run locally, i.e. within the OSF operational network, by the alarm system administrators to monitor the current state of the system, debugging, assess performances and so on. For example, a simulator to feed a DASU with a certain set of IASIOs and check the outputs they produce after running the TF. Or a tabular view of inputs and outputs of the ASCEs to follow the computation at run-time. A web application to browse the LTDB is also foreseen.

The architecture of the GUI framework described so far is mostly logical as the final design depends on the GUI framework adopted and decided at a later stage.

--- End of document ---