Supervisor PINAZZA Ombretta

SOSA SÁNCHEZ Citlalli Teresa

Summer student report: **A 3D model of ALICE's detectors**

ALICE DCS GROUP

I have worked with the ALICE DCS group, which is responsible for the Detector Control System. My assigned task was making a 3D model of the ALICE experiment [1], where every detector shows its own state with a color code, and the color can change depending on the status of the detector. In this way, the graphical representation can immediately give an idea of the overall status of the experiment. Two different states are simulated: an operational status related to data taking can be described through the *fsm*. An *fsm* (Finite State Machine) is a simple way to describe the status of an object with keywords, thus hiding the complexity. Furthermore, a *safe* status related to the LHC has been introduced in ALICE, to describe whether or not the current configuration of a detector is appropriate to avoid damages by the LHC beams.

In order to do this 3D model, I have studied how to use a software called WinCC OA [2], branded by Siemens, and a CERN extension called JCOP Framework [3]; together, these programs allow to build human-machine interfaces that can be used to monitor physical variables (connected to hardware devices and sensors) and operate devices, as needed in detector controls.

WinCC OA concept is based on *datapoints*. First it is necessary to know what are a datapoint type (dpt) and a datapoint element (dpe). A dpt is the definition for datapoints: an array with elements of different types (floats, integers, strings, booleans, etc.), that is predefined in the program or by the user. The dpe is every element (variable) inside dpt.

In order to simulate a detector, I have created a special dpt with two nodes: one is *state* and the other is *action*. Inside there are two string variables, *fsm* and *safe*. Through a script it is possible to simulate detector's states in such a way that when the variable action of the dpt has certain value, the variable state gets a new value. In the following image the dpt structure is presented.

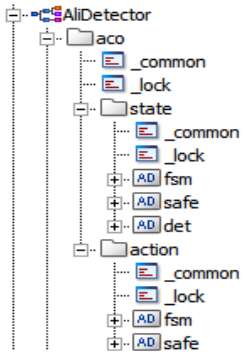

Fig. 1. Datapoint type

The dpt name is AliDetector. Based on this structure I've created different datapoints: besides the dp called *aco,* that corresponds to ACORDE detector, there are sixteen more for each Alice's detector. The variable *state.fsm* can have the following values: STBY_CONF, STANDBY, OFF, READY, BEAM_TUN, MOVING_STBY_CONF, MOVING_STANDBY, MOVING_OFF, MOVING_READY, MOVING_BEAM_TUN, and MOVING_READY. The variable *state.safe* can

be: SAFE, NOT_SAFE and SUPERSAFE. The same strings represent the real detectors states.

For the node *action*, the string *action.fsm* can take GO_STANDBY, GO_STBY_CONF, GO_READY, GO_BEAM_TUN and GO_OFF. On the other hand, the string *action.safe* can take the values GO_SAFE, GO_NOT_SAFE and GO_SUPERSAFE. The shifters operating the ALICE detector during data taking use the same commands.

These states will be used in the model of ALICE detectors to show in a graphical way something similar to what is happening in the real experiment. My supervisor gave me 2D views of ALICE's detectors from inside and outside the experiment. Starting from these two sketches, I reworked the panels and connected each detector to the corresponding state and action nodes. The actual values retrieved from the *fsm* and *safe* variables are represented in the panel as detector's color. From within this panel it is possible to send a command using *action.fsm* and *action.safe*, such that it changes the values in state node, and therefore updates the corresponding detector in the panel.

The first approach was to connect every detector in a single panel and it looks like this:

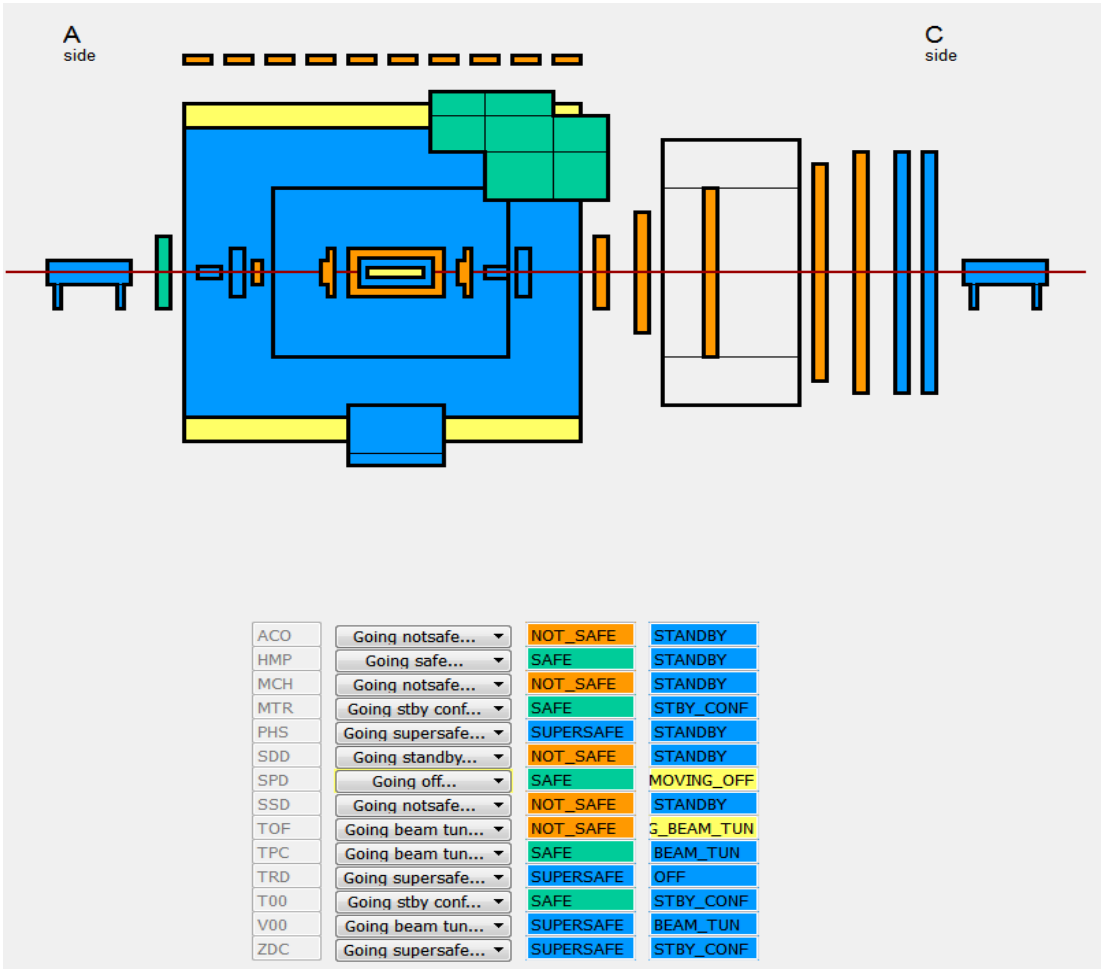

Fig. 2. Alice Inside View

In **Fig. 2** above, one can see that every state has a specific color and every detector shows the same color as its corresponding box. However, this is not a really convenient way because there are a lot of unnecessary rows. Instead, I modified the code in such a way that it is possible to select the detector, and hence the state and variable one wants to change. In the following image (**Fig. 3**) I'm presenting the *fsm* and *safe* subpanels, which allow changing the state of every detector.

Fig. 3. Subpanels programmed to select detector and action.

Then, I combined the 2D (**Fig. 2**) model with these two subpanels (**Fig. 3**). So, instead of having that huge panel above, the result is a smaller panel with a list of options, thus allowing choosing the detector to be operated (**Fig. 4**). Working with the 2D models allowed me to understand how the software works, and how to connect the elements in the panel with the datapoint variables. Afterwords, I began building the 3D model.

In order to do every connection and every panel, it was necessary to write a code with instructions. WinCC offers three different functions to interact with datapoints: dpSet (to insert a value in a datapoint), dpConnect and dpGet (to retrieve values from a datapoint). Using dpGet the actual value is retrieved once. On the other hand dpConnect opens a thread to stay connected to the datapoint, and obtain the value every time it changes.

When the code is written, the difference between dpGet and dpConnect is really notable: the first only needs the variable where the gotten value will be stored and the address where the dpe resides, while the second requires the definition of a new function to be executed whenever the element is changing. Here is an example:

```
string fsm;
dpGet(fsm,"System1:aco.state.fsm");

dpConnect("show_state","System1:aco.state.fsm");
show_state(string dp, string fsm)
{
   //here the actions over the fsm variable can be written
}
```

As it can be seen in **Fig. 4**, this new panel is simpler than the previous one; doing this modification required introducing a new function and a new dpe where I've saved the name of the detector selected by the user. Let's suppose that we want to change the tof's safe state: first we will select *tof* detector and automatically the variable *det* in state node will take the value "tof". In this way the selected action will be applied on that detector, and the same for fsm state. To implement the reading of this new dpe, it was necessary to insert a new dpConnect function in the code. It made possible the constant actualization for the detector's name.

The last step involves directly the 3D detectors' model. My supervisor provided me with two panels with detectors already done, that I could study to understand and interpret the code. Then, I have programmed other detectors to build a more complete model. The final 3D model includes tof, emc, spd, sdd, ssd, phs, trd and hmp detectors.

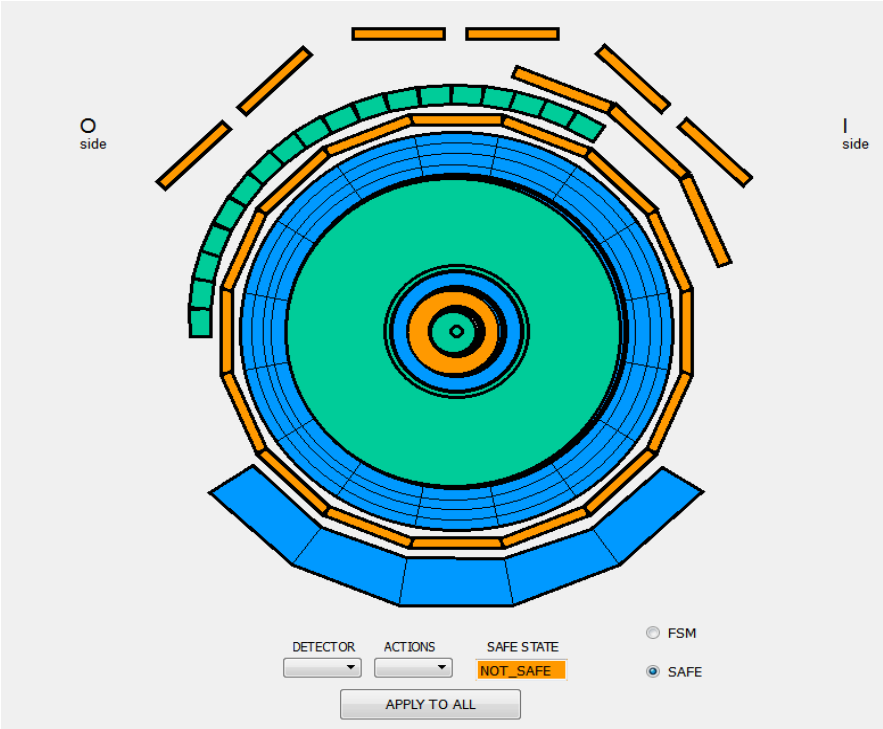

Fig. 4: Alice Outside View

The visualization is connected with every fsm and safe state in such a way that it is possible to change the selected fsm or safe state, and send an action through the cascade button. The image below (**Fig. 5**) shows the final view for the detectors.

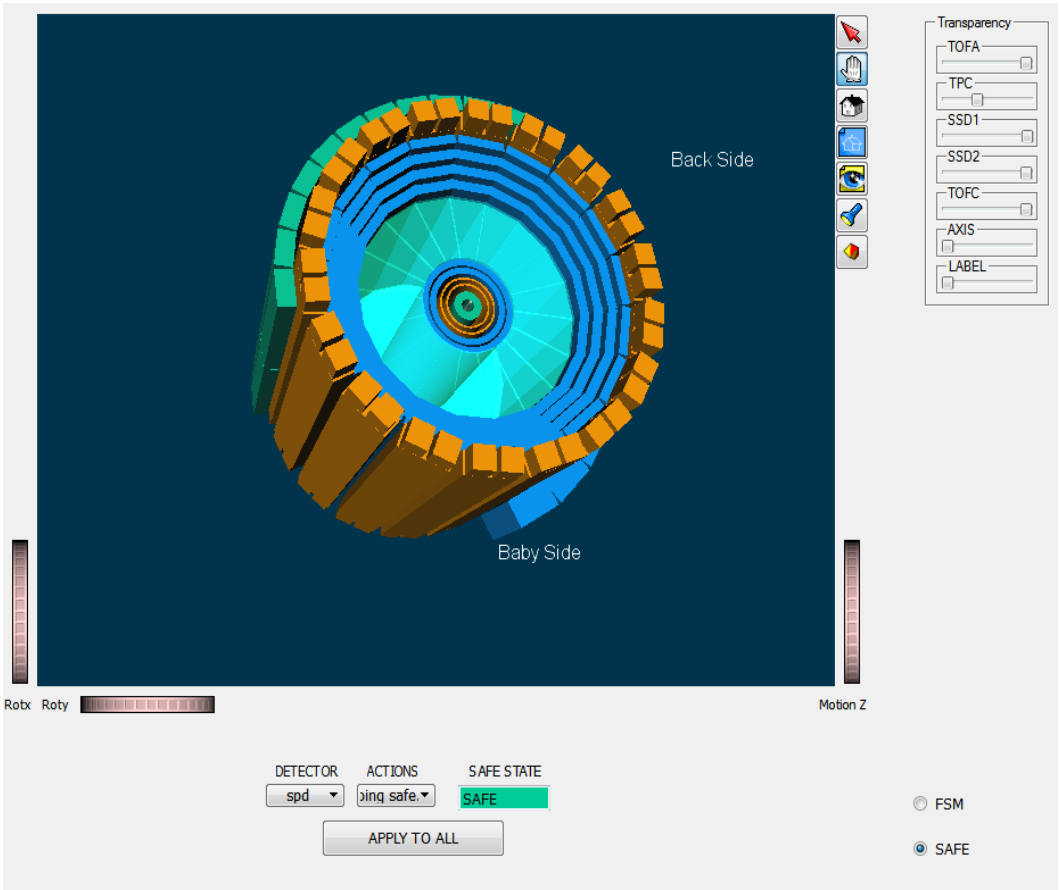

Fig. 5. ALICE in 3D

The buttons on the right side (**Fig. 5**) allow making transparent or solid color for some detectors. This view can be rotated to see the so called *back side* and *baby side*.

When I worked with 2D models, I used predefined tools to draw the elements like squares, lines, curves, circles, etc. For the 3D model there are no predefined shapes available, but a special WinCC object called EWO Viewer, where all the 3D elements are programmed. For this reason I had to write a piece of code to define every detector, then I connected every object with its corresponding datapoint and subpanel.

For example, to insert the Phos detector I had to search its general geometry, like the azimuthal angle it covers. A piece of code looks like this:

```
draw_phs( )
{
   string  xx, yy, sx1, sx2, sy1, sy2, rot, mod, name;
   float angle;
   int i;
   for ( i = 0; i <= 4; i++){
      sprintf(mod,"%02d",i);              // module number
      angle = deg2rad(260+(15)*i);        // separation angle between each module
      sprintf(xx,"%8.6f",460*cos(angle));
      sprintf(yy,"%8.6f",460*sin(angle)); // xx & yy will be the module position
      sprintf(sx1,"%8.6f",cos(angle));
      sprintf(sx2,"%8.6f",-sin(angle));
      sprintf(sy1,"%8.6f",sin(angle));
      sprintf(sy2,"%8.6f",cos(angle));
      /* The instructions will save the measures in different variables,
         that will be used to rotate the object with a 3×3 matrix called rot.
      */
      rot = sx1+","+sx2+", 0.0 ,"+sy1+","+sy2+", 0.0, 0.0, 0.0, 1.0"; // rotation matrix
      name = "PHOS"+mod+"A";              // name for every object element
      this.addShapeInGroup("PHSA","Box",name, makeDynString("x",xx, "y",yy, "z","90",
                           "dx","35.0","dy","60.0", "dz","50", "rotationMatrix",rot));
      // this will add the shape in EWO viewer
}
//The underlined code should be repeated to do the back side of the detector.
```

Conclusions.

This 3D model is an interactive way to give a general idea of the operational status of a complex object like a detector. Through this work I have learned tools used worldwide to operate in industrial controls, and I've familiarized with the way DCS experts work to monitor and control the ALICE detector. Eventually, my panels will be integrated in the ALICE simulator and exposed during the CERN Open Day 2013, to illustrate visitors how the shifters operate in the real control room.

Acknowledgments. I am grateful for the attention given by my supervisor Ombretta Pinazza also for her knowledge, kindness and patience; it has been a pleasure to work with her.

References.

[1] The ALICE collaboration et al., "The ALICE experiment at the CERN LHC", JINST 3, S08002, 2008.

[2] Simatic WinCC Open Architecture
http://www.automation.siemens.com/mcms/human-machine-interface/en/visualization-software/simatic-wincc-open-architecture/pages/default.aspx

[3] Holme O et al., "The JCOP Framework", proceedings ICALEPCS 2005, Geneva CH, 2005.