Project 1





**FDIR**

*Spacecraft fault protection system*

**Euro Team**

|  |  |
| --- | --- |
| **Mikko AHVENNIEMI** | 20096680 |
| **Pierre ALAUZET** | 20096699 |
| **Julien COLIN** | 20096706 |
| **Benoît STARCK** | 20096705 |

**Table of contents**

October 17th, 2009

**CS554 - Design for Software & Systems**

[**Illustration table** 4](#_Toc243148181)

[**Introduction** 5](#_Toc243148182)

[**1.** **Problem understanding** 6](#_Toc243148183)

[1.1. Business case & system context 6](#_Toc243148184)

[**1.1.1.** **FDIR utility** 6](#_Toc243148185)

[**1.1.2.** **System requirements** 6](#_Toc243148186)

[1.2. Problem frames 6](#_Toc243148187)

[**1.2.1.** **Domains identification** 6](#_Toc243148188)

[**1.2.2.** **Context Diagram** 6](#_Toc243148189)

[**1.2.3.** **Problem frames** 6](#_Toc243148190)

[**2.** **Functional requirements (use-case model)** 7](#_Toc243148191)

[2.1. Use-case diagram 7](#_Toc243148192)

[**2.1.1.** **Actors description** 7](#_Toc243148193)

[**2.1.2.** **Use-case diagram** 7](#_Toc243148194)

[**2.1.3.** **Use-case specifications** 8](#_Toc243148195)

[2.2. Sequence diagrams 11](#_Toc243148196)

[**2.2.1.** **Fault recovery** 11](#_Toc243148197)

[**2.2.2.** **Safe response in case of hazardous conditions** 12](#_Toc243148198)

[**2.2.3.** **Critical failure** 13](#_Toc243148199)

[**2.2.4.** **Select system & get information** 14](#_Toc243148200)

[**2.2.5.** **Start system, restart system & refresh system state** 15](#_Toc243148201)

[**2.2.6.** **Backup system and shutdown system** 16](#_Toc243148202)

[**3.** **Non-functional requirements** 17](#_Toc243148203)

[3.1. Identified quality attributes 17](#_Toc243148204)

[**3.1.1.** **Testability** 17](#_Toc243148205)

[**3.1.2.** **Availability** 17](#_Toc243148206)

[**3.1.3.** **Adaptability** 17](#_Toc243148207)

[3.2. Improvised quality attributes 17](#_Toc243148208)

[**3.2.1.** **Avaibility** 17](#_Toc243148209)

[**3.2.2.** **Reliability** 17](#_Toc243148210)

[**3.2.3.** **Resilience** 18](#_Toc243148211)

[**3.2.4.** **Performance** 18](#_Toc243148212)

[**3.2.5.** **Usability** 18](#_Toc243148213)

[**4.** **Usability analysis & design** 19](#_Toc243148214)

[4.1. Preliminary user interface design 19](#_Toc243148215)

[4.2. Discussion on usability scenarios 24](#_Toc243148216)

[**Conclusion** 25](#_Toc243148217)

[**References** 26](#_Toc243148218)

**Illustration table**

[Figure 1: Use case diagram 6](file:///D:\Documents\UTBM\MON%20CURSUS\GI05\KAIST\Projects\euroteam\Project%201\Part%20Final\CS554_EuroTeam_Project1.docx#_Toc243147949)

[Figure 2: Sequence diagram - Fault recovery 10](file:///D:\Documents\UTBM\MON%20CURSUS\GI05\KAIST\Projects\euroteam\Project%201\Part%20Final\CS554_EuroTeam_Project1.docx#_Toc243147950)

[Figure 3: Sequence diagram - Safe response in case of hazardous conditions 11](file:///D:\Documents\UTBM\MON%20CURSUS\GI05\KAIST\Projects\euroteam\Project%201\Part%20Final\CS554_EuroTeam_Project1.docx#_Toc243147951)

[Figure 4: Sequence diagram - Critical failure 12](file:///D:\Documents\UTBM\MON%20CURSUS\GI05\KAIST\Projects\euroteam\Project%201\Part%20Final\CS554_EuroTeam_Project1.docx#_Toc243147952)

[Figure 5: Sequence diagram - select system & get information 13](file:///D:\Documents\UTBM\MON%20CURSUS\GI05\KAIST\Projects\euroteam\Project%201\Part%20Final\CS554_EuroTeam_Project1.docx#_Toc243147953)

[Figure 6: Sequence diagram - Start system, restart system & refresh system state 14](file:///D:\Documents\UTBM\MON%20CURSUS\GI05\KAIST\Projects\euroteam\Project%201\Part%20Final\CS554_EuroTeam_Project1.docx#_Toc243147954)

[Figure 7: Sequence diagram - Backup system & shutdown system 15](file:///D:\Documents\UTBM\MON%20CURSUS\GI05\KAIST\Projects\euroteam\Project%201\Part%20Final\CS554_EuroTeam_Project1.docx#_Toc243147955)

**Introduction**

In the purpose of applying and studying real case project for the *Design for Softwares and Systems* course, our team is required to understand and design a fault protection system for a spacecraft as described in the article by Steve Easterbrook et al. [Eas98]. The first part of this project is to understand the problems we have to respond, to specify the needs of our client and to start thinking about a user system interface.

More specifically, we are required to use two different methods for problem understanding, namely **problem frame analysis** and **use-case modeling**. The goal of this study is to get hands-on experience of these two concepts and understand how they can be combined in order to resolve a problem.

Then, we are expected to identify the direct **user needs** and the **non-functional requirements** of the system as well. To conclude, we have to consider the **usability issues** for designing the **user interface** of the system, as we learn a couple of weeks ago.

When performing use-case modeling and problem frame analysis, we have to analyze the requirements as described in [Eas98]. Besides that, when identifying non-functional requirements, however, we could improvise and add more quality attributes that we believe are essential to the system in question.

In this purpose, our report will be divided in four parts. First of all, we will explain the problems encountered to build this system. We‘ll use the problems frames to explain in details that. Secondly, we‘ll precise the needs of the client and the system, so the requirements using use-case models. Next we’ll detail the non-functional requirements (identified and improvised). And finally, we describe the usability scenarios in relation to some of the usability scenarios introduced in [BJK01], and provide some screen shots of our interface.

1. **Problem understanding**
   1. Business case & system context
      1. FDIR utility
      2. System requirements

* 1. Problem frames
     1. Domains identification
     2. Context Diagram
     3. Problem frames

1. **Functional requirements (use-case model)**
   1. Use-case diagram
      1. Actors description

The FDIR system is designed to control systems of the spacecraft. But this system can act alone, would be useless if there were no human control to decide which actions the system must launch. Therefore, this system is designed for the crew of the spacecraft during their mission. Moreover the spacecraft and the crew are helped by the flight controller on Earth. They could do same actions as the crew. For example, during the time the crew sleep, the crew could command the system and do some tests, get information about any devices,…

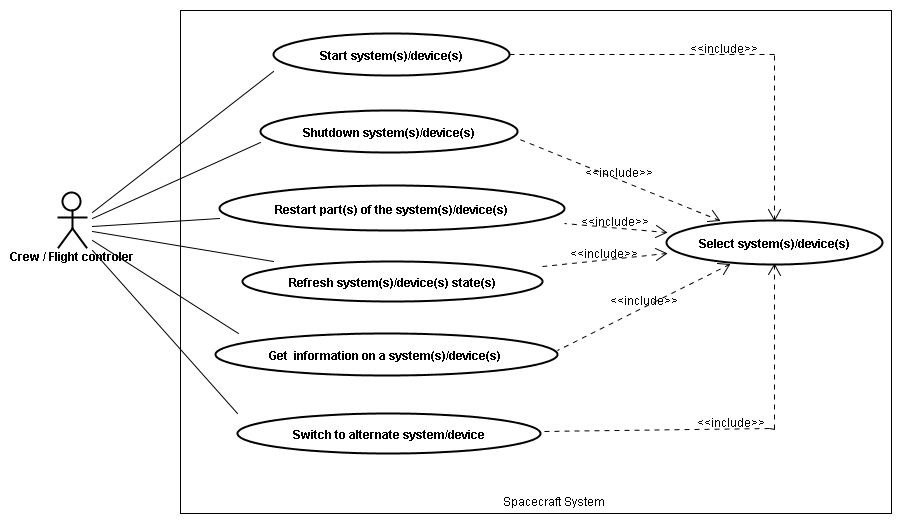
* + 1. Use-case diagram

Figure 1: Use case diagram

* + 1. Use-case specifications

|  |  |
| --- | --- |
| Name | Start system(s)/device(s) |
| Actors | Crew / Flight control |
| Description | User can start device(s) or system(s) of the spacecraft system whenever he wants or because it was shutdown |
| Precondition | - System(s) are off  - One or several device(s) or system(s) have been selected |
| Events flow | 1. Click on the “start” button  2. Wait for the system to start |
| Post-condition | System(s) has been started |
| Exception | - System states could be already turned on while it is still displayed as “off” on the FDIR system |

|  |  |
| --- | --- |
| Name | Shutdown system(s)/device(s) |
| Actors | Crew / Flight control |
| Description | User can shutdown device(s) or a part(s) of the spacecraft system whenever he wants or in case of failure |
| Precondition | - Devices or parts of the system have to be running  - One or several device(s) or system(s) have been selected  - Other systems must not have dependencies to the selected device(s) or system(s) |
| Events flow | 1. Click on the “shutdown” button |
| Post-condition | System(s) has been shutdown |
| Exception | * System states could be already turned off while it is still displayed as “running” on the FDIR system * If other systems have dependencies to the selected devices(s) or system(s) display a warning and information about the dependencies and offer to override. |

|  |  |
| --- | --- |
| Name | Restart part(s) of the system(s)/device(s) |
| Actors | Crew / Flight control |
| Description | User can restart a device or a part of the spacecraft system whenever he wants or in case of failure |
| Precondition | - Device or part of the system has to be running  - Requested system(s)/device(s) have been selected  - Other systems must not have dependencies to the selected device(s) or system(s) |
| Events flow | 1. Click on the “restart” button |
| Post-condition | System is restarting |
| Exception | * System states could be already turned off while it is still displayed as “running” on the FDIR system * If other systems have dependencies to the selected devices(s) or system(s) display a warning and information about the dependencies and offer to override. |

|  |  |
| --- | --- |
| Name | Refresh system(s)/device(s) states |
| Actors | Crew / Flight control |
| Description | User can refresh the states of any device or system to see if this one is still working correctly or not |
| Precondition | - Device or part of the system has to be running  - Requested system(s)/device(s) have been selected |
| Events flow | 1. Click on the “refresh” button |
| Post-condition | System is refreshing |
| Exception | - If the system(s) or device(s) are not responding change status not responding. |

|  |  |
| --- | --- |
| Name | Switch to alternate system/device |
| Actors | Crew / Flight control |
| Description | If the device is not responding or if there is a failure, user may switch to another system/device |
| Precondition | - Select **one and only one** system or device  - Requested part has been selected  - Other system(s) or device(s) must not have dependencies to the selected system, or the switch has to be able to be done seamlessly. |
| Events flow | 1. Click on the “switch” button |
| Post-condition | Alternate system takes the control. |
| Exception | * If the alternate system is broken as well, it may generate a fatal error of the system * If other system(s) or device(s) have dependencies to the selected systems and the switch cannot be made seamlessly, display a warning and information about the dependencies, and offer to override. |

|  |  |
| --- | --- |
| Name | Get information on a system(s)/device(s) |
| Actors | Crew / Flight control |
| Description | User may seek information about any device or system on the spacecraft |
| Precondition | - Requested system(s)/device(s) have been selected |
| Events flow | 1. Specify query  2. Click on the *GetInfo* button |
| Post-condition | Information about the selected system appears on the screen. |
| Exception |  |

|  |  |
| --- | --- |
| Name | Select system(s)/device(s) |
| Actors | Crew / Flight control |
| Description | User can select any system or device in order to issue commands |
| Precondition |  |
| Events flow | 1. Select the requested part(s) |
| Post-condition | The chosen part is selected. |
| Exception | The chosen part is still not selected. |

* 1. Sequence diagrams

We build several sequence diagrams in order to explain in detail the use case diagram and specification. Indeed, as the FDIR system is not represented in the use cases, we need to see how it is interacting with the spacecraft, the crew and the ground controller. Three sequence diagrams are explaining how the FDIR is working: *Fault recovery*, *Safe response in case of hazardous* **conditions** and *Critical failure*.

Moreover, we present three sequences diagrams about the crew and flight controller actions: *Select system & get information*, *Start system, restart system & refresh system state* and *Backup system and shutdown system*.

* + 1. Fault recovery

Fault recovering scenario shows interactions between the FDIR system and the spacecraft when a fault is detected. FDIR is controlling several values. If one is in an out-of-tolerance state, it be automatically returned to the FDIR which will start the fault localization process. Once the fault has been located, FDIR will be able to proceed to several actions like *recovery*, *shutdown* or *retry*.

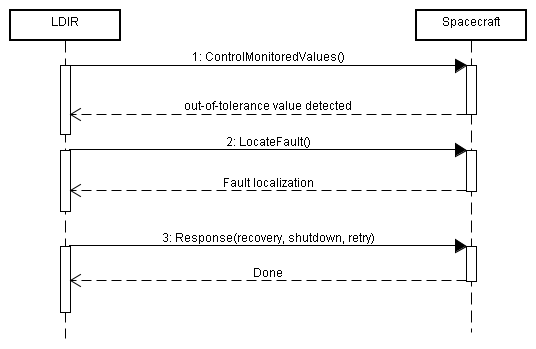


Figure 2: Sequence diagram - Fault recovery

* + 1. Safe response in case of hazardous conditions

This scenario start after that FDIR has detected a hazardous condition of problem in the spacecraft. After having isolated the problem, FDIR is able to proceed to two different actions depending of the kind of spacecraft (unmanned or manned spacecraft).

Within an unmanned spacecraft, FDIR will shut down all the non-critical functions in order to focus on the device/system problem and minimize the damages. It will also move the antenna to point toward earth in order to receive human commands and decisions.

Within a manned spacecraft, process is easier because as humans are inside the spacecraft, they can directly interact with the system without needing is functions shutdown process or antenna redirection. Then LDIR is just giving the hand to the crew.

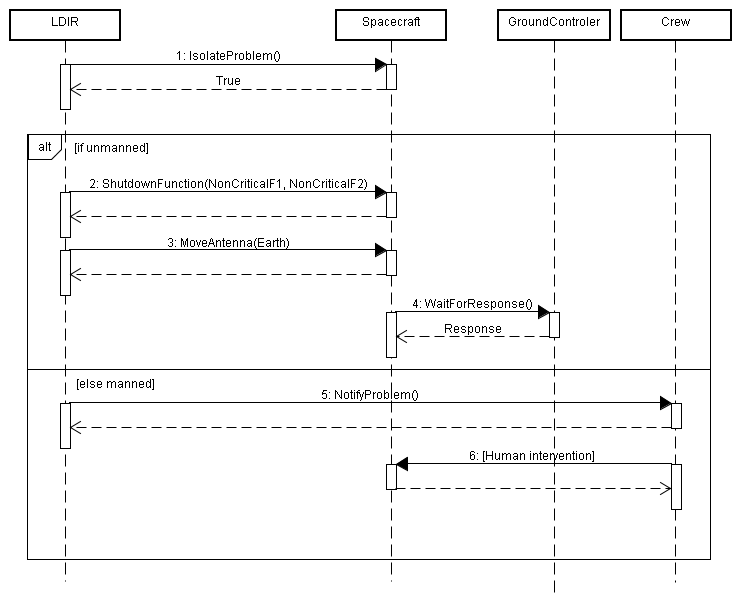
****

Figure 3: Sequence diagram - Safe response in case of hazardous conditions

* + 1. Critical failure

Critical failure scenario starts as the *Fault recovery* scenario (cf. 2.2.1) as we consider the detection of an out-of-tolerance value in the spacecraft. But if a failure cannot be recovered, FDIR system is going thru different decisions and actions. After localizing the error, FDIR is giving the hand to the crew and put itself in a manual mode state. Crew can then execute several actions like *backup*, *shutdown* or *retry*.

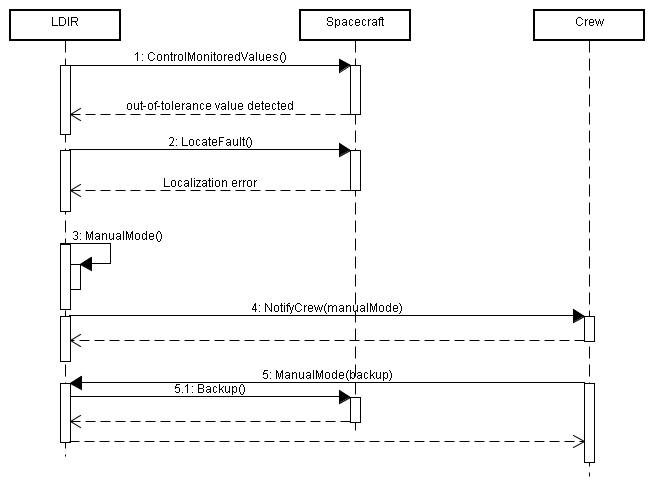
****

Figure 4: Sequence diagram - Critical failure

* + 1. Select system & get information

This sequence diagram is introducing the first actions the crew or space controllers are doing. First of all, they have to select the system they want to access to do several actions. Once selected, they can check its information state. All the systems are displayed in a first screen called *System List screen*. We represent this screen by the *SystemListUI* object. When the crew push the button *display information* (*getSystemInformation* attribute), the second screen is receiving the action and shows the system information. This one is called *Information system screen*, and represented by *SystemInformationUI*.

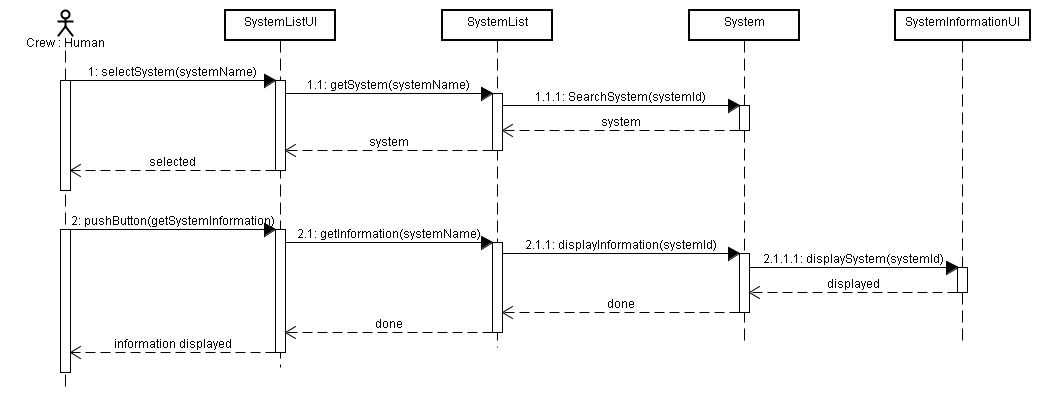
This and the two next sequences diagrams are containing the System object. This one is representing a system in general, but it could be one of the system’s devices.

Figure 5: Sequence diagram - select system & get information

* + 1. Start system, restart system & refresh system state

The goal of the following sequence diagram is to explain the behavior of action buttons while system is selected and the relevance of the *refresh system* button.

Indeed, the refresh button is really useful as a system is sometime doing something whereas its state is not updating. For example, a system can be showed at turned off while it was just restarting. User is attending to do some actions like *start* the system whereas it is already running. This is this case of system behavior we try to explain thanks to this diagram. Moreover, it is also helping you to understand how the *start* and *restart* actions are working.

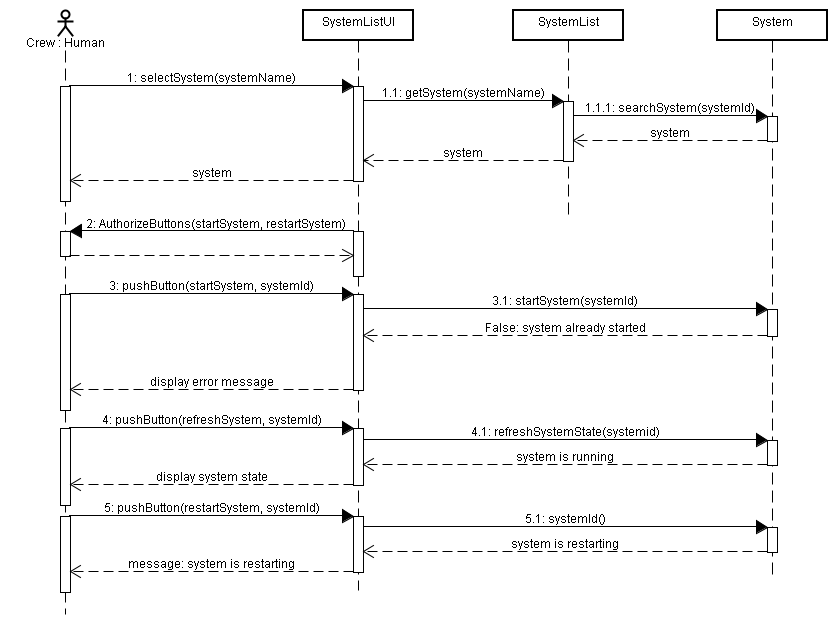


Figure 6: Sequence diagram - Start system, restart system & refresh system state

* + 1. Backup system and shutdown system

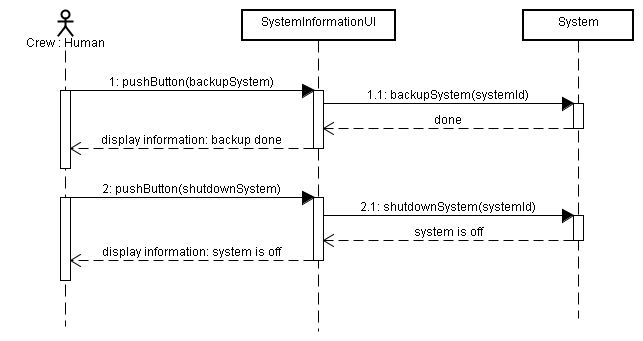
To conclude, let us display the backup and shutdown system scenarios. We already consider that a system is selected and located before being able to do these actions. As user does not have anything else to do besides pushing the buttons, actions are really simple in this diagram.

Figure 7: Sequence diagram - Backup system & shutdown system

1. **Non-functional requirements**
   1. Identified quality attributes
      1. Testability

The system and its parts have to be able to be tested through inspections, simulations and analyses before on-board installation.

“Hence, FDIR functionality must be validated through a combination of inspection, simulation, and analysis.” [EAS98]

Priority: High

* + 1. Availability

The system must not lock or stall when processing data. It must work asynchronously and must be available all the time.

“Fault protection operates asynchronously, and may be invoked at any time” [EAS98]

Priority: High

* + 1. Adaptability

The system has to be configurable in order to adapt to several environment. FDIR has to be adaptable for manned and unmanned spacecraft. It also has to adapt to several hardware component, different from one spacecraft to another.

“For unmanned spacecraft“ “additional requirements over those for unmanned craft”[EAS98]

Priority: High

* 1. Improvised quality attributes
     1. Avaibility

FDIR is processing a lot of critical information that shouldn’t be lost. Redundant storage system has to be added in order to avoid any waste of data.

Priority: Medium

* + 1. Reliability

The system must be reliable in all operating conditions. System failure could lead to loss of human life. However, as reliability is inversely related to complexity in software application, this non-functional requirement should be one of the top priorities.

Priority: High

* + 1. Resilience

The system must be able to maintain an acceptable level of service in spite failures in parts of the FDIR system.

Priority: Medium

* + 1. Performance
       1. Response time

The system must respond in timely manner so that problematic systems can be shut down before any damage is done.

Priority : Medium

* + - 1. ThoughtPut

The system has to be able to deliver and receive a lot of requests and messages at the same time, and so we have to avoid overloading the FDIR.

Priority : Medium

* + 1. Usability

Priority : Low

1. **Usability analysis & design**
   1. Preliminary user interface design

Displaying information continuously 🡪 multiple screens but no tabs

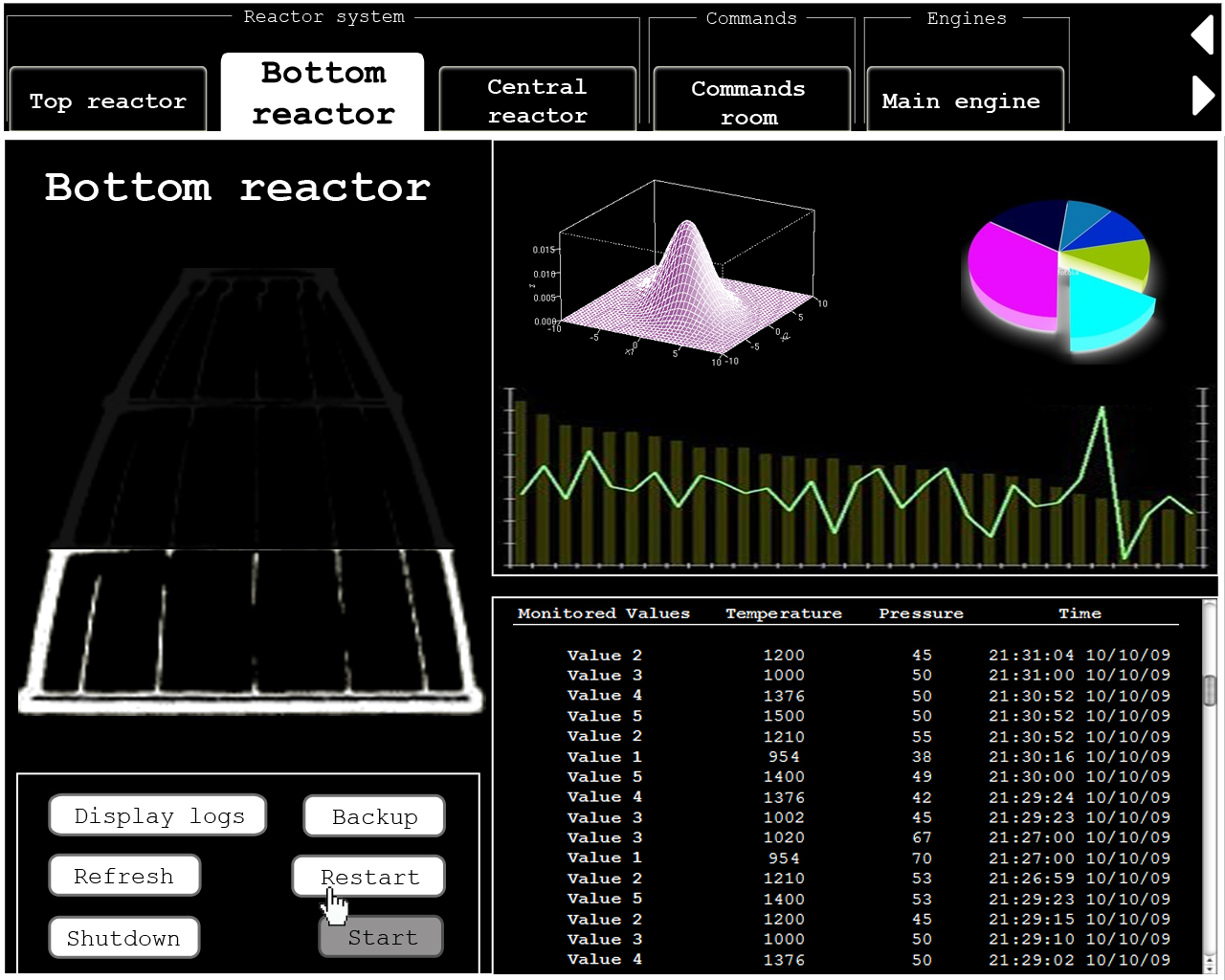
Overview of the spacecraft always available 🡪 list of systems & subsystems with information (monitoring values, temperature, pressure)

Professional interface 🡪 non useful features like displaying spacecraft screens. Crew would be formed to be on the spacecraft and do not need to have a visualization of the spacecraft. Title of device or system is enough

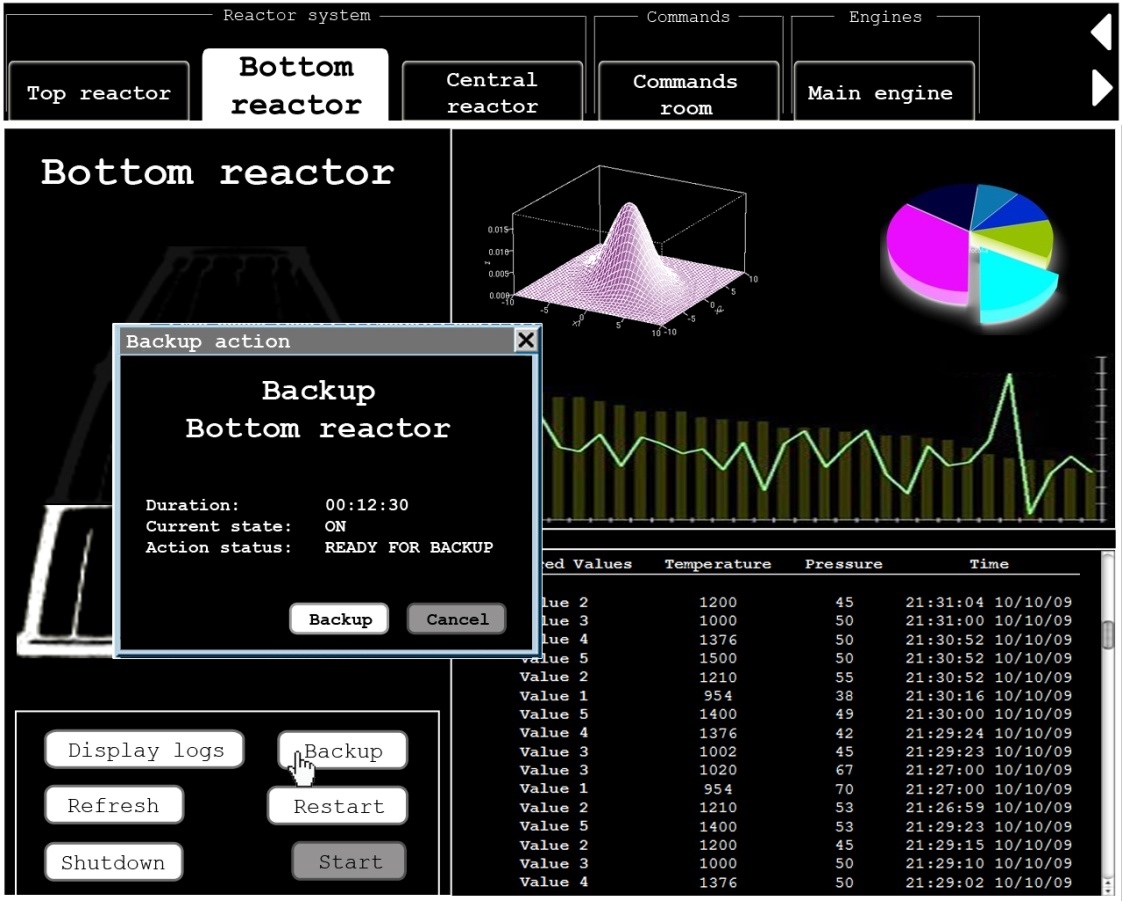
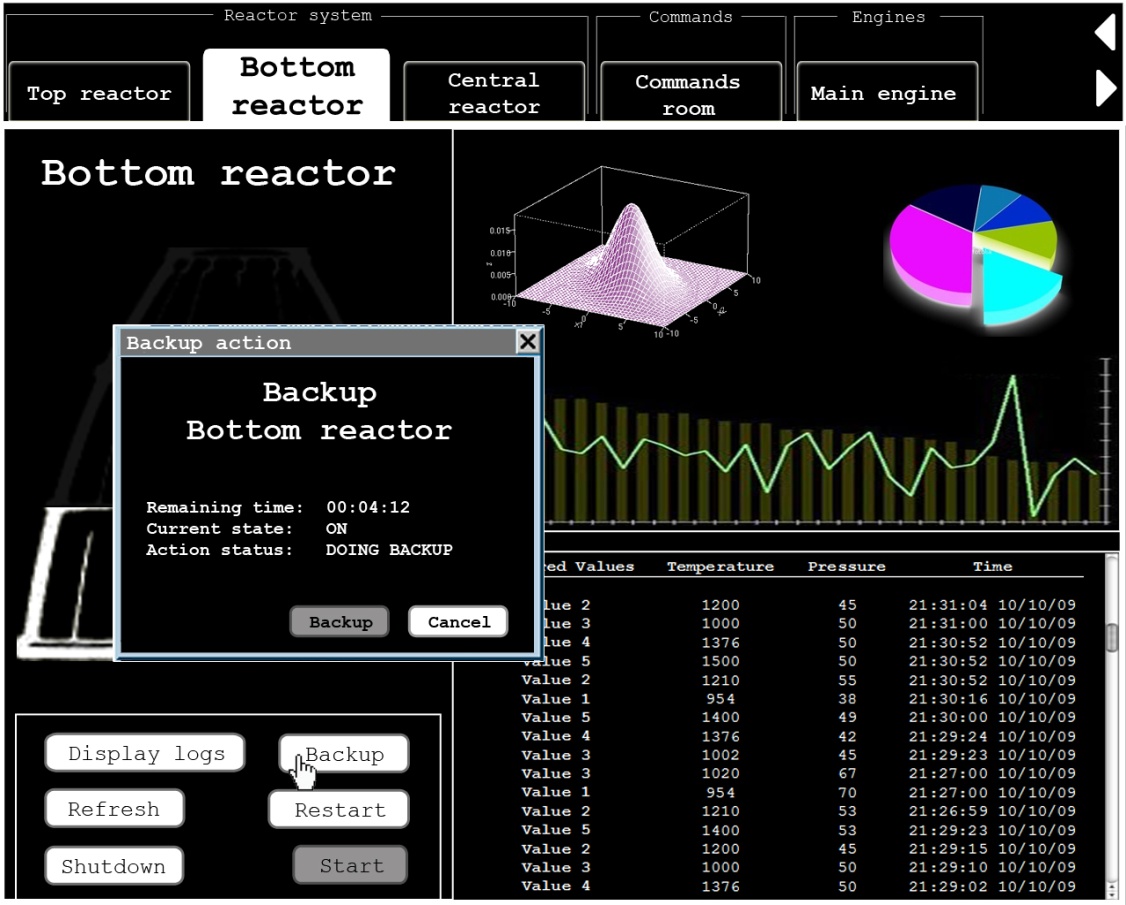
Locate the fault 🡪 displaying left panel with spacecraft scheme. List of the systems appearing on the tree should appear on the scheme too. If an alert appears on one system, we should be able to localize it geographically on the scheme.

Keep the control of the spacecraft with safety, observability & commandability 🡪 bottom panel with buttons (shutdown, restart, backup and recovery data)









* 1. Discussion on usability scenarios

The FDIR system is very important for the success of the mission. It provides information to the crew, can do a lot of specific actions on specific components of the spacecraft. Besides it has to be very reliable and efficient. In this purpose, we will describe some usability scenarios essential for a critical system like FDIR. We will use the general usability scenarios studied a couple of weeks ago, and find some more essential for this type of system.

* + 1. Scenario 13

Besides that the FDIR should provide views that have to be consistent from one view to another, in order to leverage the human knowledge (Scenario 13). User has to understand very quickly a new view and doesn’t have to lose his marks. So with similar interfaces, it will not take much time to user to achieve it.

This is the reason why we tried to harmonize the three views of the three FDIR screens. Every item is implicit and intuitive for user. He does not really need a training to understand the panels, actions or information. Moreover, the consistency from one view to another is really present as we are displaying different actions between screen 1 and screen 2. Indeed, buttons in screen 1 allow user to do several actions for the whole system/device selection, whereas buttons in screen 2 are just performing in the only displayed system/device.

When manipulating a system in screen 1 & 2, we can find out that we always have access to the same action at any time. It is never changing as we respect the consistency.

* + 1. Scenario 15

Moreover the system has to support concurrent activities (Scenario 15). The crew could test one device and do another action, for example restart a second device, during the same time. The crew doesn’t have to wait that one action is over to start another one. Besides this point the ground control could do an action directly from Earth while another function is running. This case must not be a problem for the FDIR.

Respecting this usability scenario, we decided to design three different screens that can be used in the same time in order to do concurrent actions. Indeed, crew can restart a system in the screen 2 while selecting another on screen 1 and displaying logs of this selection on screen 3. Moreover, tab system on screen 2 is also related to this scenario: allowing user to easily switch the view from a system/device to another allows him to do an action in a system (push the *Backup* button) and another action in another system (push the *Shutdown* button) a half of second after just in switching of tab.

* + 1. Scenario 16

Next, FDIR could provide an interface where user can see the entire spacecraft, and at the same time, can navigate in another view through this spacecraft (Scenario 16). User doesn’t have to change several times of menus and different screens to see the entire spacecraft system. It will be a huge loss of time for all users.

As for scenario 15, the choice of multiple screens is related to this usability scenario. User can see different information without hiding anything: systems/devices list, spacecraft view, monitoring values, logs, etc. Moreover, we combine the idea of multiple screens with the one of spreading the screens in several parts to strengthen the one and unique system view. Indeed, screen 1 always displays the entire view of the spacecraft, in order to easily see the problems or the selected systems/devices.

* + 1. Scenario 3

First of all, the FDIR system must be able to allow user canceling commands (Scenario 3). Crew could have mistaken one command to another for example, especially during the launch of the spacecraft where pressure is high and movements very hard to control. Let us take another example: if the crew decide to backup a system but received a notice a couple of second after saying that this same system is crashing, he would like to interrupt the backup to restart it. In this case, a cancel button is needed.

We expected these kinds of cases putting a *Cancel* button while action is processing. It is related to the Figure x.

* + 1. Scenario 8

Then, the FDIR could recover from failure as much as it cans (Scenario 8). When the systems crashed, FDIR system allows user to recover the last data. It will reduce the time the user will spend to do his last actions again.

This scenario is related to the *Backup* button we have in screen 1 & 2. This one allows user to switch to an alternate system/device.

* + 1. Scenario 19

Finally, the system could display the expected duration of any action before the user launches it (Scenario 19). It could be really useful for user in order to estimate the time he need to do several action or to help him to decide for priorities or fastness.

Moreover, FDIR system also display the remaining duration of an action when this one is already started. This can really help crew or ground control estimating the next actions.

**Conclusion**

The first project was very interesting. This project is based about an interesting topic we are not used to study every day: a spacecraft. After the careful reading of the subject we try to identify the needs of the clients, the problems of the actual system or the problems we will encounter during the design part.

In this purpose, we learned how to represent problems using a new type of diagram: the problems frames. These diagrams will be useful for the second project, the design of the FDIR system. Besides we use the UML to represent the functional requirements of the system, by describing them by using use case diagram.

Moreover, we tried to identify non-functional requirements in a specific system like a spacecraft. That’s why we find non-functional needs in the paper of course, but we also think about additional requirements by ourselves. We improvised and put the most important we found.

Finally, we explain some usability scenarios essential for us in this kind of system. We based our scenarios according to the paper we read two weeks ago. After that, we drew some interfaces to represent our idea of this system, including our usability scenarios.

**References**

**Web Sites**

1. <http://www.bredemeyer.com/pdf_files/NonFunctReq.PDF>,

about non-function requirements

**Articles**

**Books**

1. [Eas98] **Steve Easterbrook, and et al**., *Experiences Using Lightweight Formal Methods for Requirements Modeling*” IEEE Transactions on Software Engineering, Vol. 24, No. 1, January 1998.
2. [Jac05] **Michael Jackson**, *Problem frames and software engineering*, Information and Software Technology, Special Issue: 1st Int Workshop on Advances and Applications of Problem Frames, K. Cox, et al. eds, Vol. 47 No. 14, pp. 903-912, Nov. 2005.