KAIST

FDIR

Spacecraft fault protection system

Euro Team

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

CS554 -Design for Software & Systems

October17th, 2009

Table of contents

Illustration table

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 global 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.

When performing use-case modeling and problem frame analysis, we have to analyze the requirements as described in [Eas98], the direct **user needs**. Then, we are expected to identify the **non-functional requirements** of the system as well. 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.

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 and provide some screen shots of our interface.

Thanks to our work on the usability, we revised a lot our previous work, starting from the UML design and non-functional requirement. Moreover, our team has been involved in refactoring this project 1 considering the professor & TA feedbacks.

1. Problem understanding

1.1. Business case & system context

In the proposed study, efficiency of formal methods has been demonstrated by overriding initial specification of an already existing system: the **F**ault **D**etection, **I**solation and **R**ecovery system (**FDIR**). We extracted, as much as we could, the requirements of this project.

This system is specially designed for a spacecraft and provides specific functions needed and requested by the client which make this product unique. Customer presents two main needs about this system: the guarantee of the completion of any time critical activities, and the control of the spacecraft with safety, observability and commandability. Indeed each spacecraft device has a predefined set of operating parameters that have a normal operating range. Values beyond this range are called *"out-of-tolerance"*. Besides that, an out of-tolerance condition may come from any possible causes. That is why information from multiple sources must be combined in order to locate the fault.

The FDIR system is unique and specially designed to generate appropriate responses when out-of-tolerance conditions are detected in hardware and software components. It means that it will be able to recover the data, locate the fault with precision and to fix them as well by a restart of a system for example. Moreover this system is designed to answer all actions launched by the crew or the flight controller on Earth. Both actors are able to interact together and launch different actions at the same time.

In this purpose, the requirements of the system are the followings:

1. **Guarantee the completion of any time critical activities of the spaceship**. Even if a device has a problem, the system must be able to analyze, locate the fault, and restart the action again, even if it has to fix by anyway the problem before (recover the data, or restart the device for example).
2. **The system provides a manual control for the user**. The crew or the flight controller must be able at any time to restart, shutdown, or switch to a spare system.
3. To control the spacecraft and the FDIR system, the crew should know information about each device. That is why **the system will display information continuously to the both actors**.
4. **The system will be able to collect data to data storage**. All data during the journey will be store in a safe place on the spacecraft. After that, the crew must be able during the complete mission to **retrieve old information about a device**, to compare with actual data for example.
5. **System must display the failure localization**. In this goal, system has specific tools to detect and locate failure in a part of the ship, and display information with precision to the crew
6. FDIR Storage System contains the collected values or data from devices as said before. FDIR checks the inputs from the storage system, and analyses these inputs to determine if irresolvable conditions has been reached. **Information about irresolvable conditions is written into a report sent as a notification to the crew.**
7. **System presents an automatic recovery to failure.** If a system is failing, FDIR is required to act on its own to recover the crash. FDIR provides responses automatically for a lot of casual issues, by shutting down or restarting the faulty part. FDIR is also able to provide responses under specific or more critical context (hazardous conditions or unresolved problems).
8. **Keep the control of the spacecraft with safety, observability & commandability.**

1.2. Problem frames

1.2.1. DOMAINS IDENTIFICATION

We identified the FDIR system domains as depicted on the context diagram (Figure 1). Crew is identified as a biddable domain. The crew consists of the people on board of the spacecraft if any. In case of a satellite there might not be any crew. Ground Control is a biddable domain as well. It consists of the people on earth monitoring the progress of space missions. They have to possibility to issue commands to the system in addition of the crew.

Information display is a causal domain and it's essentially displays installed in the spacecraft, but it can also be a display in the ground control premises on earth. Other kinds of displays could be envisioned as well, for example PDA's. The FDIR storage system is a lexical domain and contains the historical data from different systems plugged in to the FDIR as well as the data from FDIR operation.

Report is a causal domain. It contains information as specified by a search usually performed on the FDIR Storage System. Systems are a causal domain. It consists of the different devices and systems plugged in to the control of the FDIR system.

1.2.2. CONTEXT DIAGRAM

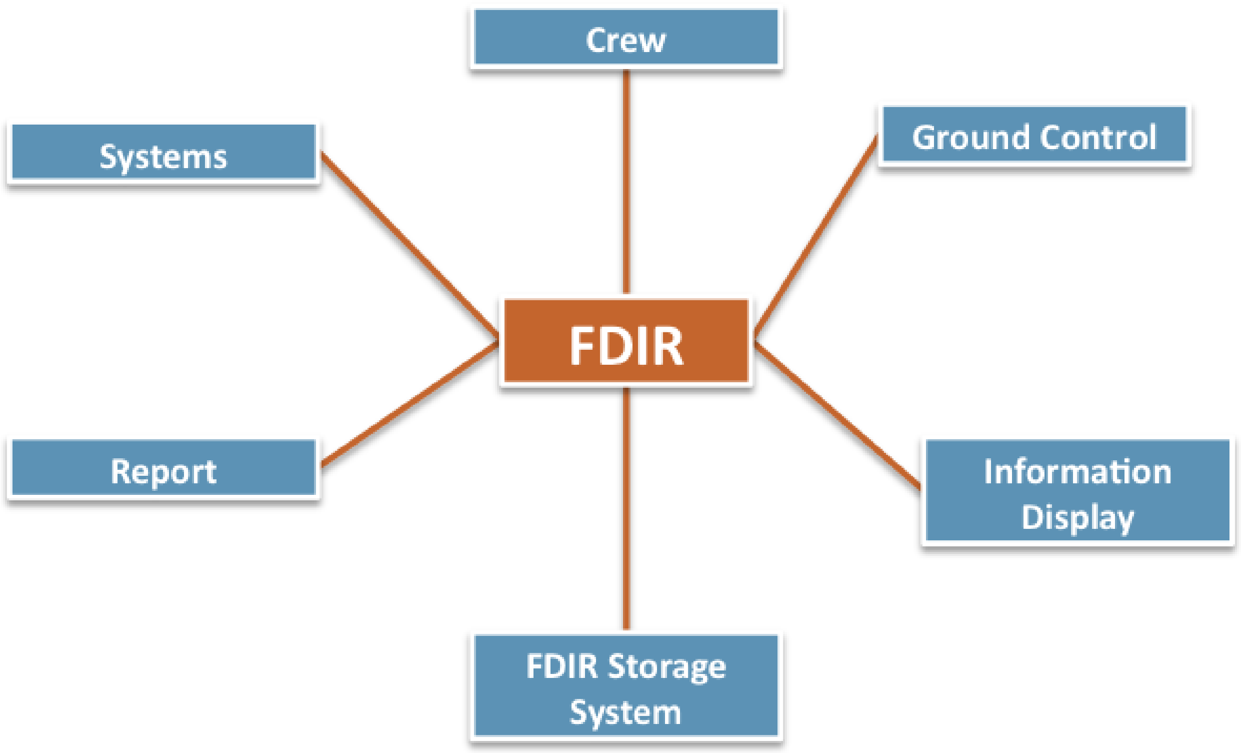
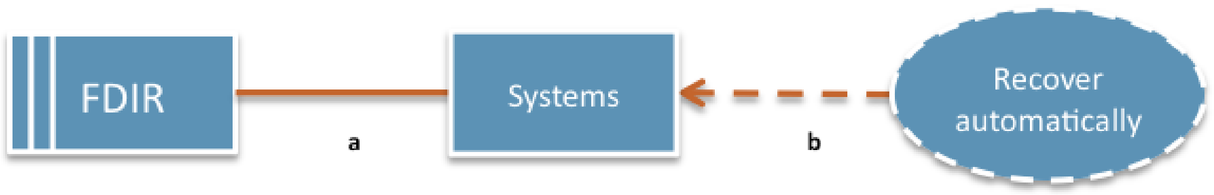


Figure 1: Problem Frames - Context diagram

1.2.3. Problem Diagrams

We identified distinct problem frames depicting certain scenarios in the FDIR systems. These frames are in order of appearance: Automatic recovery from failure, Manual control of FDIR, Displaying information continuously, Collecting system data to data storage, Information retrieval, Providing failure localization, and Response in case of irresolvable failure.

Automatic Recovery From Failure



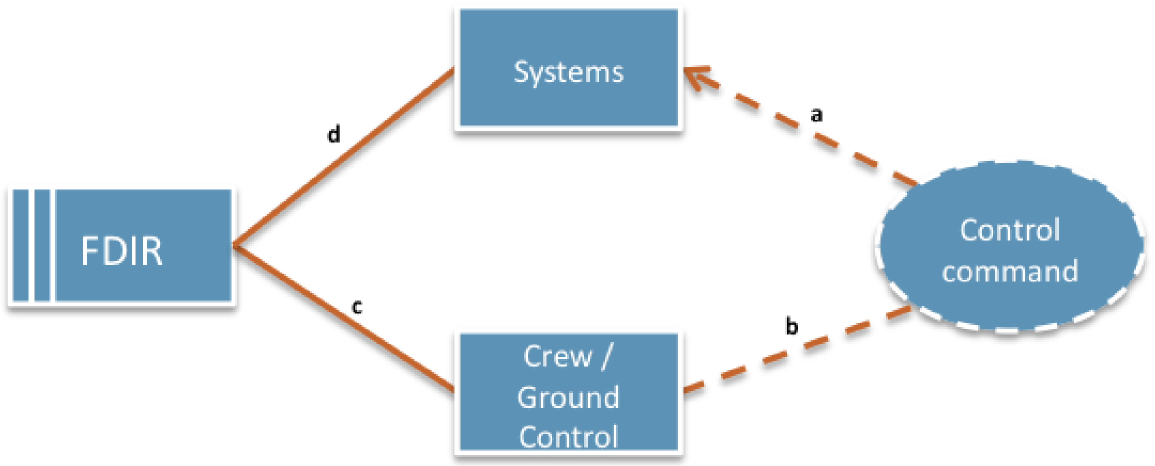
a: FDIR! {start, shutdown, restart, switch to alternate system}
Systems!{return command status}

b: {Functional, non functional, broken}

Figure 2: Problem Diagram - Automatic recovery from failure

Automatic recovery from failure (Figure 2) uses the required behavior problem frame. If FDIR deems that a failure has occurred it will try to remedy the situation by a series of actions. It can try to start, shutdown, restart the system, or try to switch to an alternate system. The systems can be, at any point in time, be functional, non-functional or broken. After issuing commands it will receive the command status from the systems.

Manual Control of FDIR



c: C/GC! {Do start, Do shutdown, Do restart, Do switch to alternate system}
FDIR!{return command status}
d: FDIR! {Issue start, Issue Shutdown,Issue Restart, Issue Switch to alternate system}
Systems! {Return command status, No return}

b: C/GC! {start, shutdown, restart, switch to alternate system}

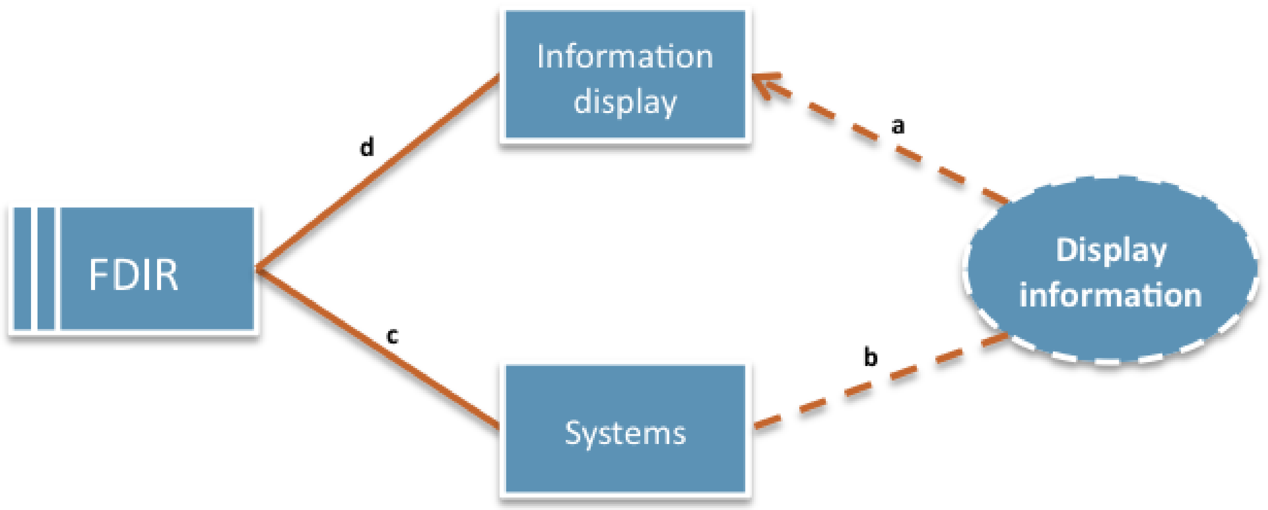
a: System! {Functional, malfunctioning, broken}

Figure 3: Problem Diagram - Manual control of FDIR

Manual control of FDIR uses the commanded behavior problem frame. The FDIR provides an interface for issuing manual commands for the crew and ground control. They might want to e.g. restart a system that is not operating in tolerance, or switch that function to use an alternate or backup system.

Commands issued by the crew or ground control are relayed to the systems by FDIR and the command status is returned to FDIR. In case the system is broken the system might not respond. The system can be Functional, malfunctioning or broken.

Displaying Information Continuously



c: Systems! {send value/no value}

b: Systems!{functionnal/not funct. proper./broken}

d: FDIR! {display in tol/out-of-tol/no resp}

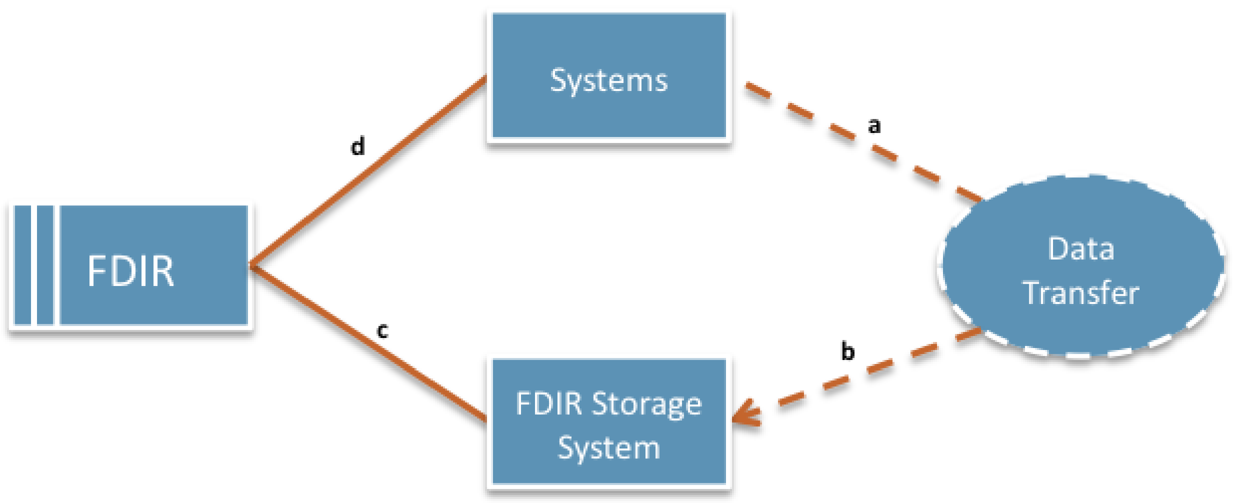
a: Information display!{console}

Figure 4: Problem Diagram - Displaying information continuously

Display information continuously (Figure 4) uses the display problem frame. The FDIR constantly relays information from the systems to be displayed on screen. The crew and flight control can make deductions about the systems based on this data.

Systems send the values to FDIR and no value if they are broken. FDIR on the other hand displays the value and interprets whether it's in tolerance. The information is displayed on the console depending on what kind of view is selected in the user interface.

Collect systems data to data storage



c: FDIR! {System ID, state value, time}

b: {System data}

d: Systems! {value, no value}

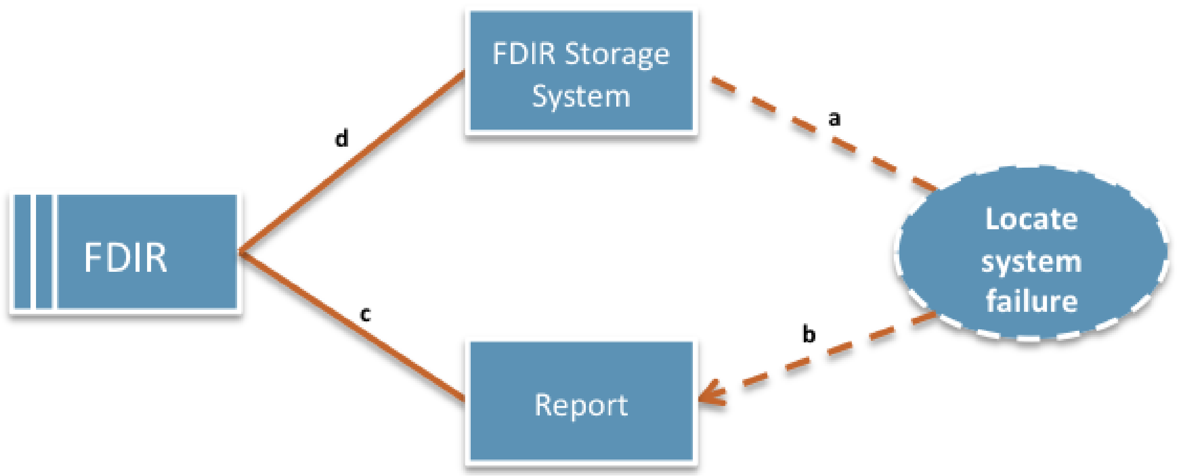
a: Systems! {System ID, state}

Figure 5: Problem Diagram - Collect systems data to data storage

Collecting systems data to data storage (Figure 5) uses the transformation problem frame. The systems send measurement values to FDIR at regular intervals. This data is stored with a timestamp to the FDIR Storage System. This approach has a link to non-functional requirements as well. If the data is stored in a centralized place with proper backup in place it doesn't matter if some parts of the system go down and the data can still be accessed. This contributes to better availability of data used for critical decision making.

All systems send some kind of data. The FDIR knows the system ID based on where the system is located. The state value is then stored with a timestamp.

Providing failure localization



c: FDIR! {write failure location, write type of failure}

b: Report! {failure data}

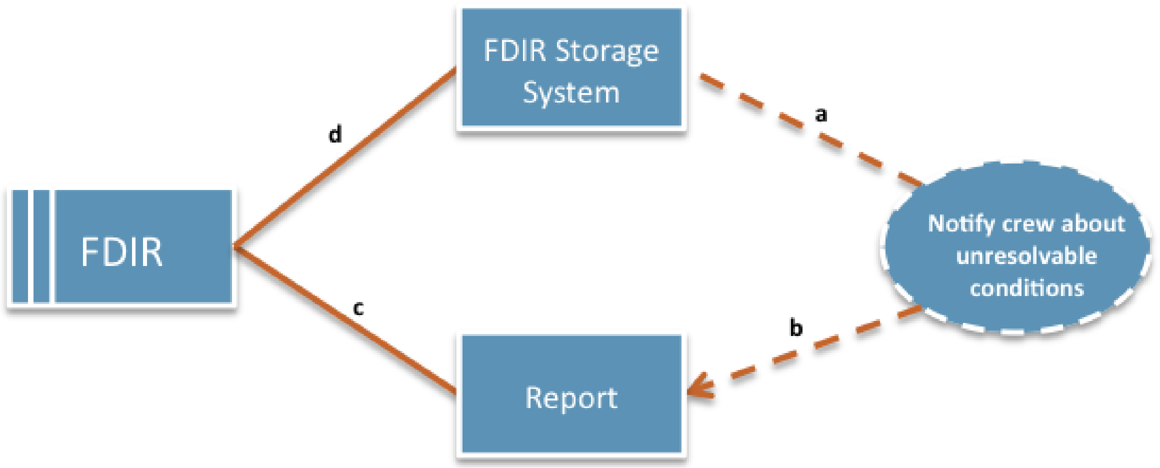
d: FDIR Storage System! {send device, send value}

a: FDIR Storage System! {device, value, time}

Figure 6: Problem Diagram - Providing failure localization

Providing failure localization (Figure 6) uses the transformation problem frame. Data is retrieved from the FDIR Storage System and then FDIR determines based on this data where the failure originates. This information is then written on a report that is later displayed based on which user interface view is open. The type of the failure is recorded in the report in addition to the localization data.

Response in case of hazardous conditions



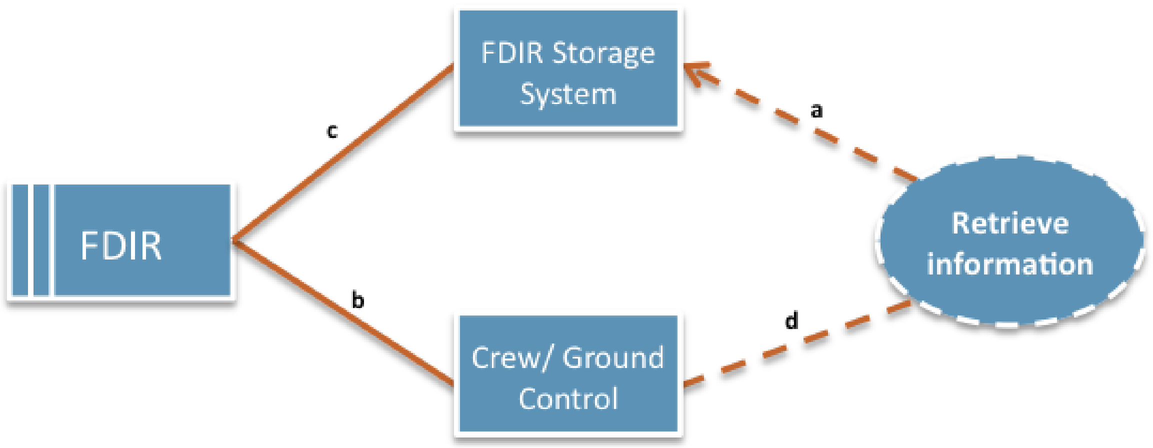
c: FDIR! {write notification, write unresolvable conditions} **b: Report! {notification}**

d: FDIR Storage System! {send device, send value} **a: FDIR Storage System! {device, value, time}**

Figure 7: Problem Diagram - Response in case of unresolvable conditions

Response in case of irresolvable conditions (Figure 7) uses the transformation problem frame. When automatic recovery fails and the issue cannot be resolved by the FDIR by itself a report is written that is then actionable by the crew or ground control. This report is based on the data collected to the FDIR Storage System.

Information retrieval



c:FDIR! {query}
FDIR SS! {return data}

a: FDIR SS!{status data}

b:Crew!{search}
FDIR!{display}

d: Crew!{search}

Figure 8: Problem Diagram - Information retrieval

Information retrieval (Figure 8) uses the commanded behavior problem frame. The crew can search information from the system logs. Each log item is stored separately in the FDIR Storage System. By specifying search criteria specific information can be retrieved. This data can then be used as basis for important decision regarding the use or repair of the systems. This data is can be more specific then general information displayed on the screen via the user interface.

The query goes through the FDIR to the storage system. Data is then sent back to FDIR for processing. Data is then sent to the crew and ground control for viewing.

1.2.4. Problem Frames conclusion

Problem frames technique provides patterns that allow us to identify each decomposed problem as a singular problem belonging to a standardized type. It is bringing us a new perspective around analysis.

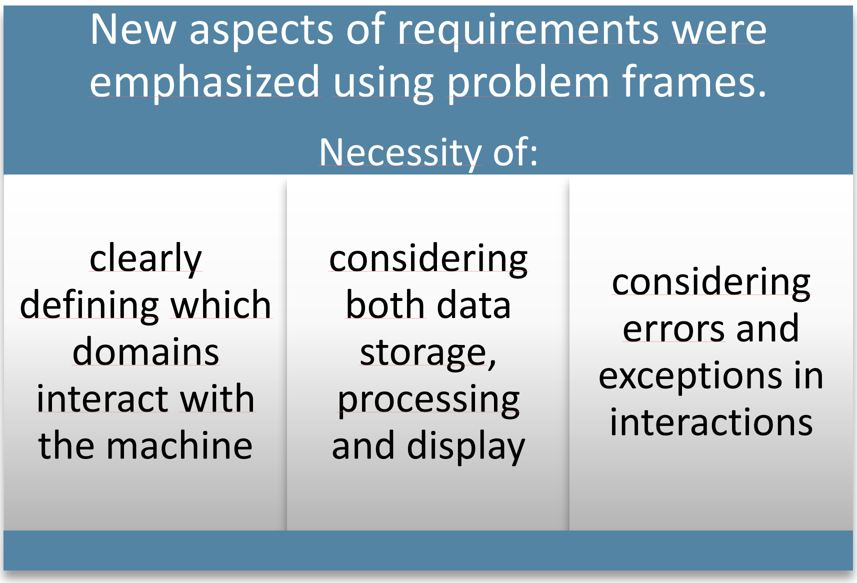

New aspects of requirements were emphasized using problem frames.
Necessity of:
clearly defining which domains interact with the machine
considering both data storage, processing and display
considering errors and exceptions in interactions

Figure 9: Problem frames method conclusion

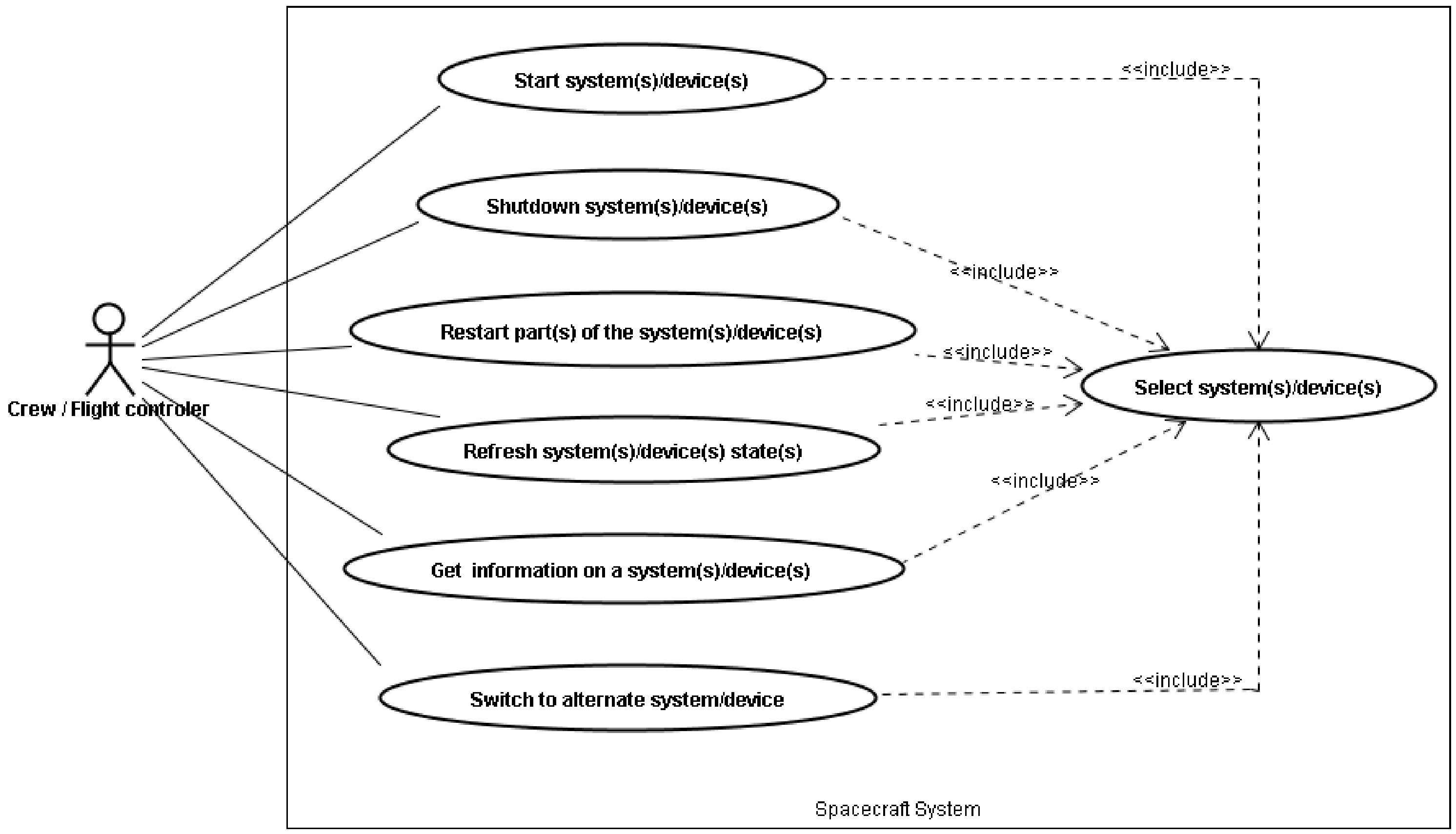
2. Functional requirements (use-case model)

2.1. Use-case diagram

2.1.1. Actors description

The FDIR system is designed to automatically control systems of the spacecraft. Moreover, a manual mode can be launch in order to let the **crew** acting on the system during their missions. Moreover the spacecraft and the crew are helped by the **flight controller** on Earth. Both actors can do the same action on the system and interact with it.

2.1.2. Use-case diagram



Start system(s)/device(s)
Shutdown system(s)/device(s)
Restart part(s) of the system(s)/device(s)
Refresh system(s)/device(s) state(s)
Get information on a system(s)/device(s)
Switch to alternate system/device
Select system(s)/device(s)
<<include>>
Crew / Flight controler
Spacecraft System

Figure 10: Use case diagram

2.1.3. 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. Confirm the 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 2. Confirm the shutdown
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	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	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 are 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 2. Confirm the restart
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	**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	- **One and only one** system/device 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 *Backup* button 2. Select a spare system 3. Confirm the backup
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 2.1. Click on the *Display in Current/New view* button on screen 1 2.2. Control the detailed information on screen 2 3.1. Click on *Logs* button on screen 1 3.2. Watch and filter logs on screen 3
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.

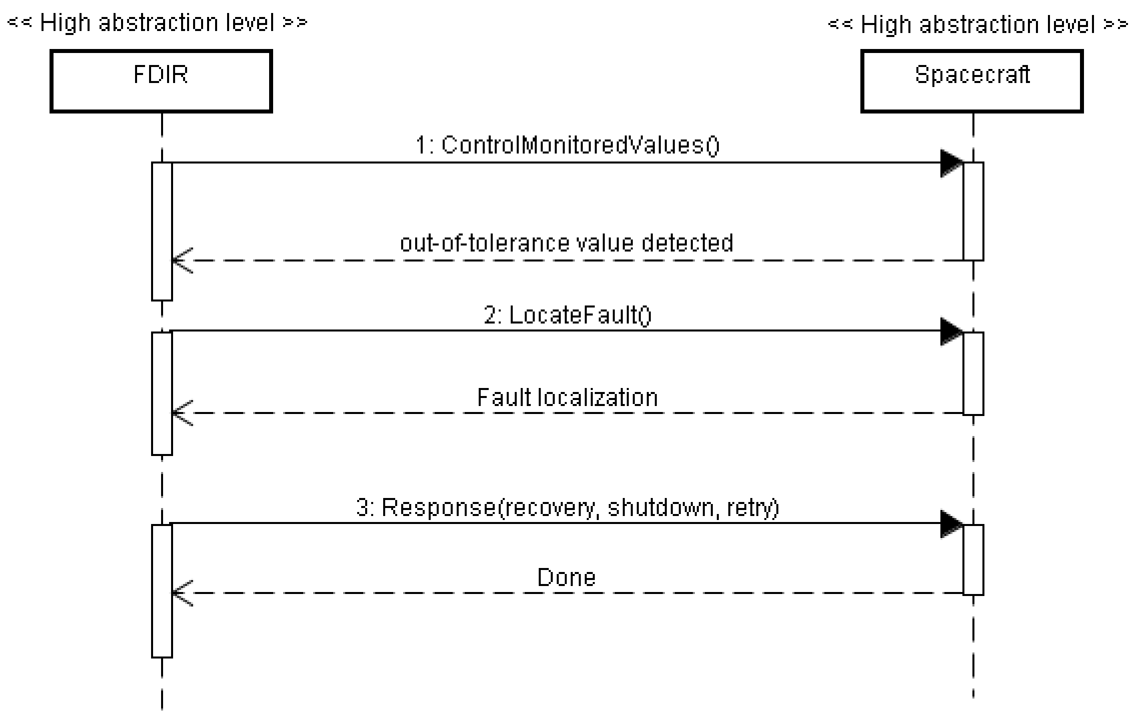
2.2. Sequence diagrams

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*.

2.2.1. 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*.



<< High abstraction level >>
FDIR
<< High abstraction level >>
Spacecraft
1: ControlMonitoredValues()
out-of-tolerance value detected
2: LocateFault()
Fault localization
3: Response(recovery, shutdown, retry)
Done

Figure 11: Sequence diagram - Fault recovery

2.2.2. 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.

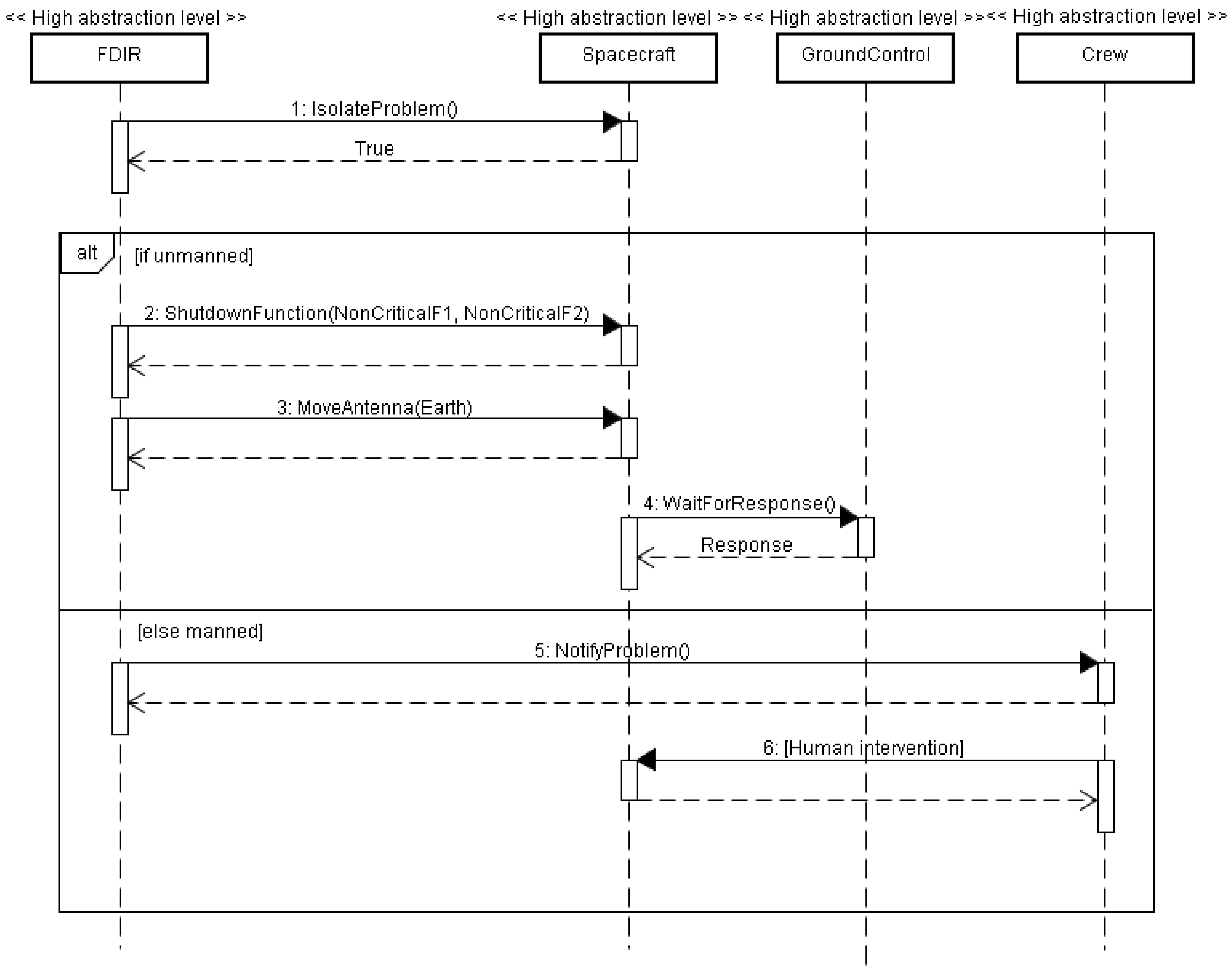
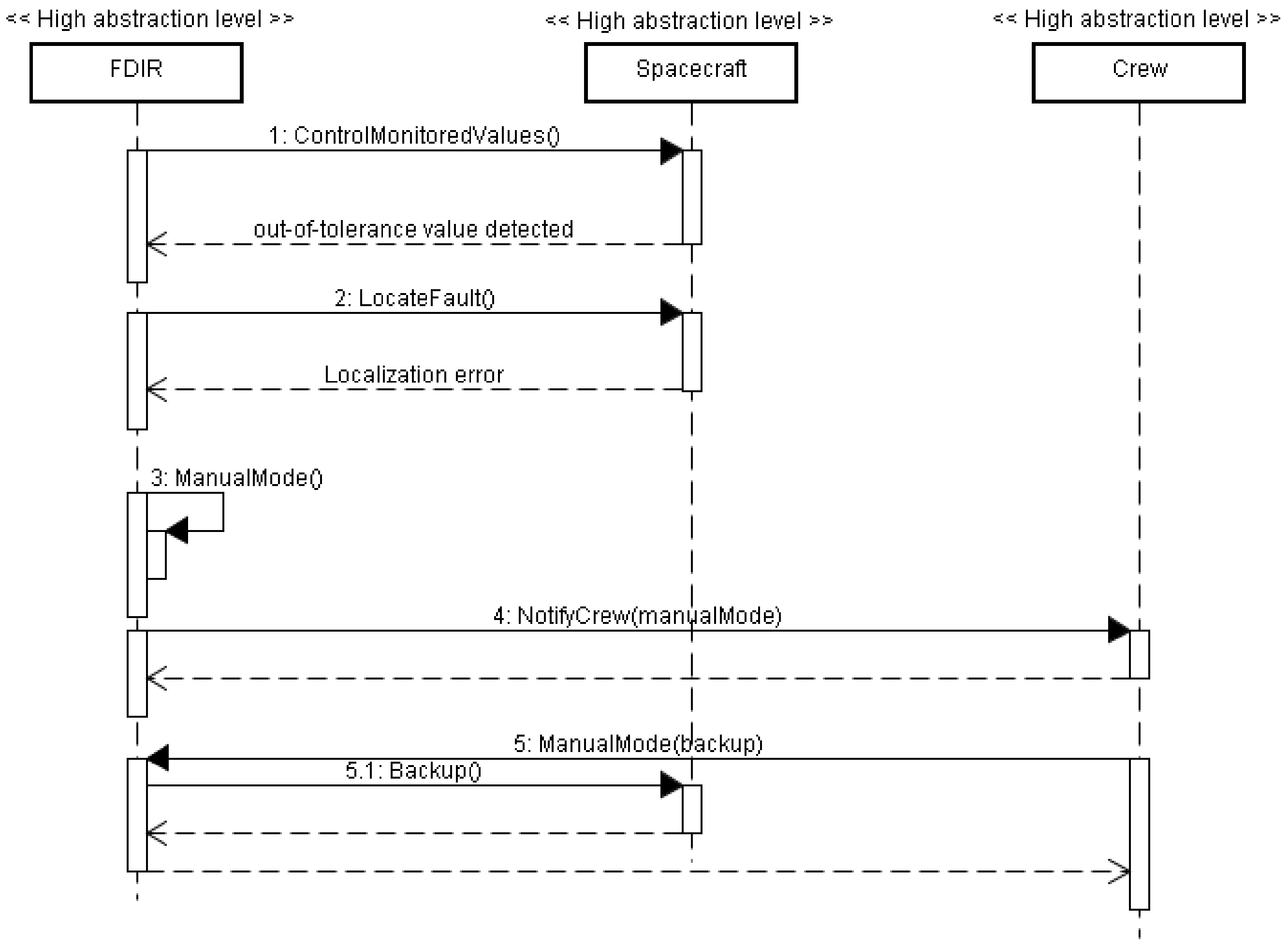


Figure 12: Sequence diagram - Safe response in case of hazardous conditions

2.2.3. 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*.



<< High abstraction level >>
FDIR
<< High abstraction level >>
Spacecraft
<< High abstraction level >>
Crew
1: ControlMonitoredValues()
out-of-tolerance value detected
2: LocateFault()
Localization error
3: ManualMode()
4: NotifyCrew(manualMode)
5: ManualMode(backup)
5.1: Backup()

Figure 13: Sequence diagram - Critical failure

2.2.4. Select system & get information

The three 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.

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 several of its information. 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*. Moreover, user can get several information about the system/device pushing the button Logs. In this case, the third screen called *Logs screen* is called (*LogUI*).

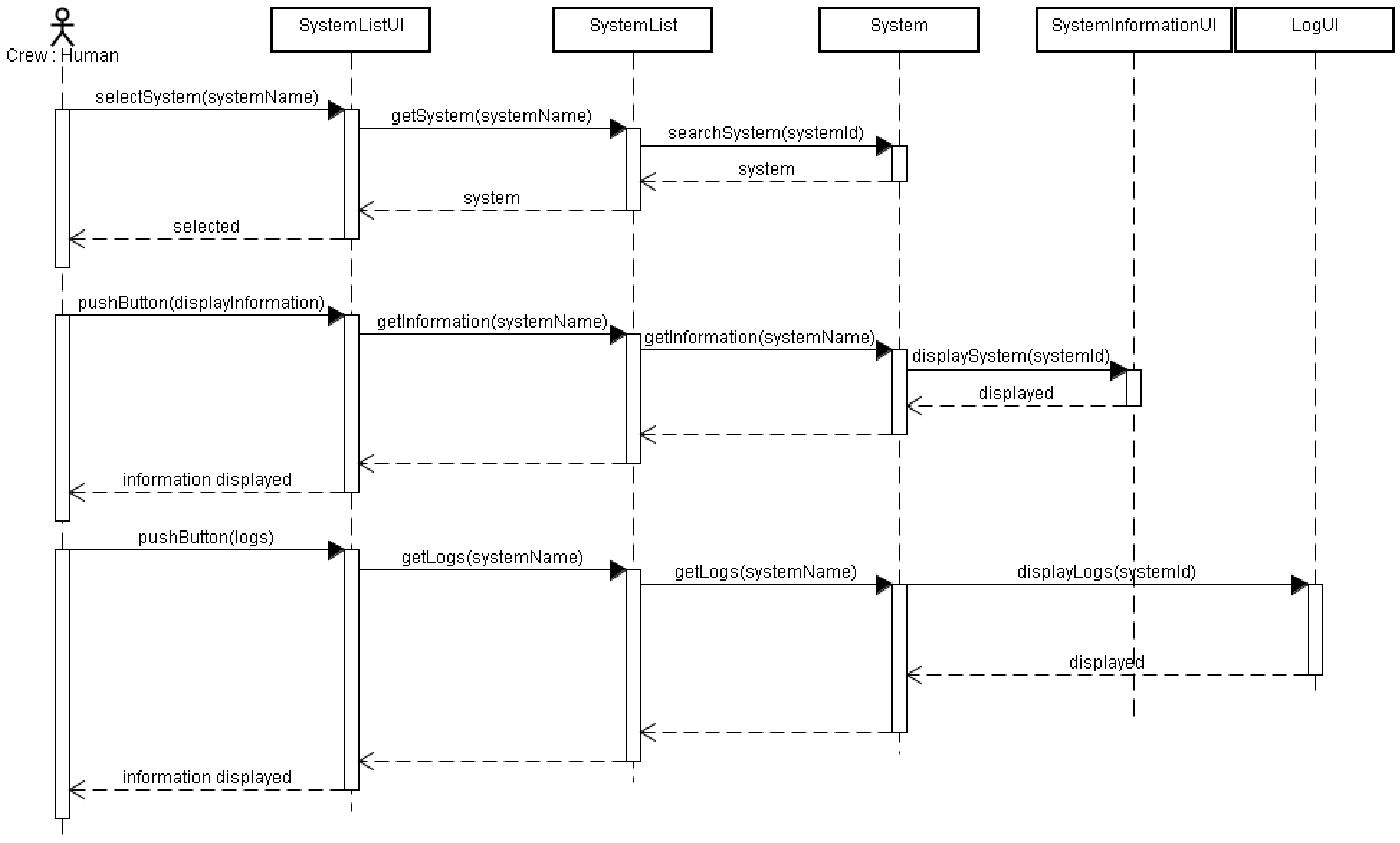


Figure 14: Sequence diagram - select system & get information

2.2.5. 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.

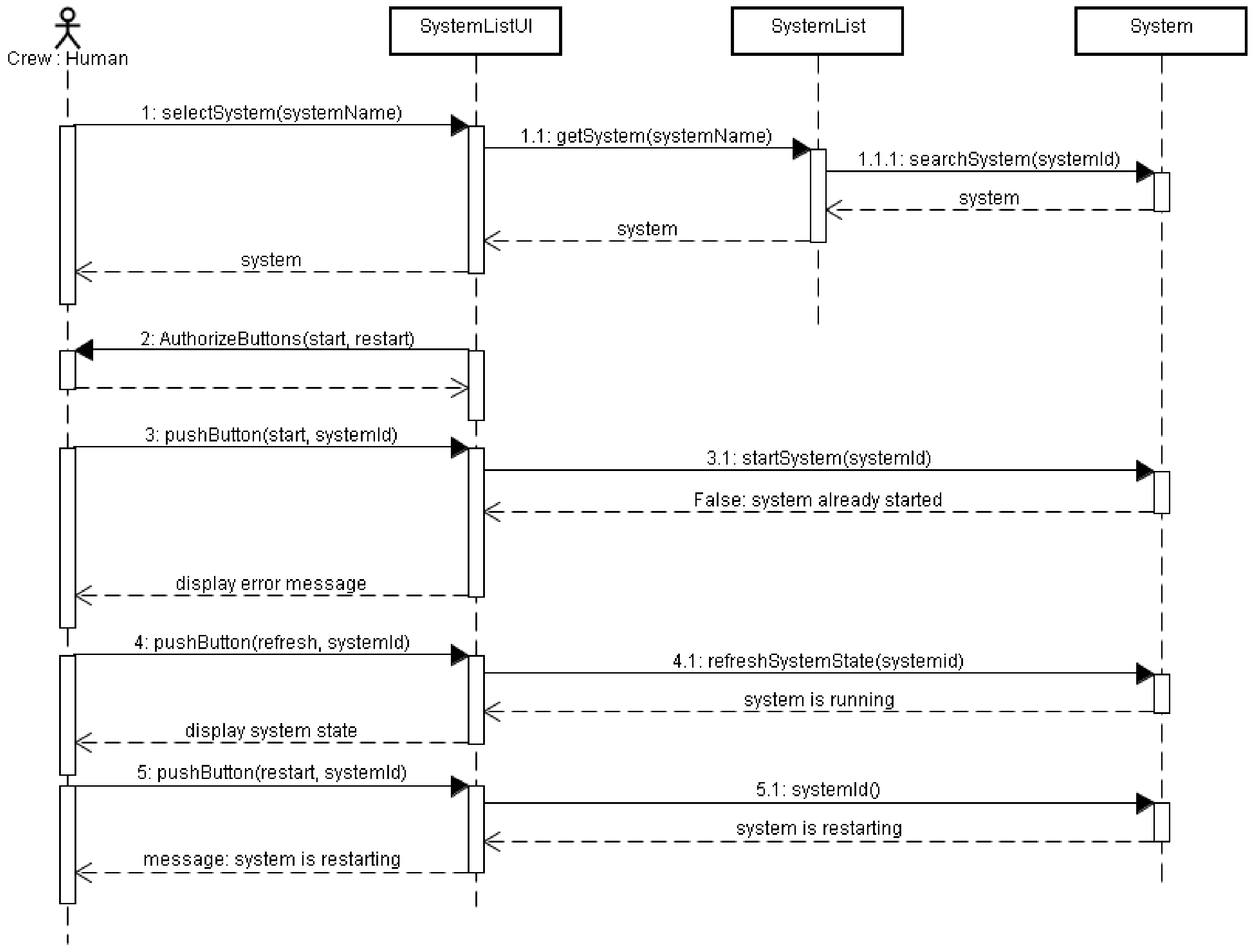

Crew : Human
SystemListUI
SystemList
System
1: selectSystem(systemName)
1.1: getSystem(systemName)
1.1.1: searchSystem(systemId)
system
system
system
2: AuthorizeButtons(start, restart)
3: pushButton(start, systemId)
3.1: startSystem(systemId)
False: system already started
display error message
4: pushButton(refresh, systemId)
4.1: refreshSystemState(systemid)
system is running
display system state
5: pushButton(restart, systemId)
5.1: systemId()
system is restarting
message: system is restarting

Figure 15: Sequence diagram - Start system, restart system & refresh system state

2.2.6. Switch to alternate system device

This scenario is one of the most interesting because this is not an intuitive crew action. Behind the sentence *Switch to Alternate System Device* there is a concept that we need to explain. First of all, this use case is represented by a button called Backup in our prototypes. Indeed this action is more or less corresponding to a backup event. First, user chooses a system to backup. Second, system is displaying him a popup with a list of systems where he has to choose one in order to replace the system to backup. Once he has chosen the system, he just needs to confirm or cancel the backup through a new popup.

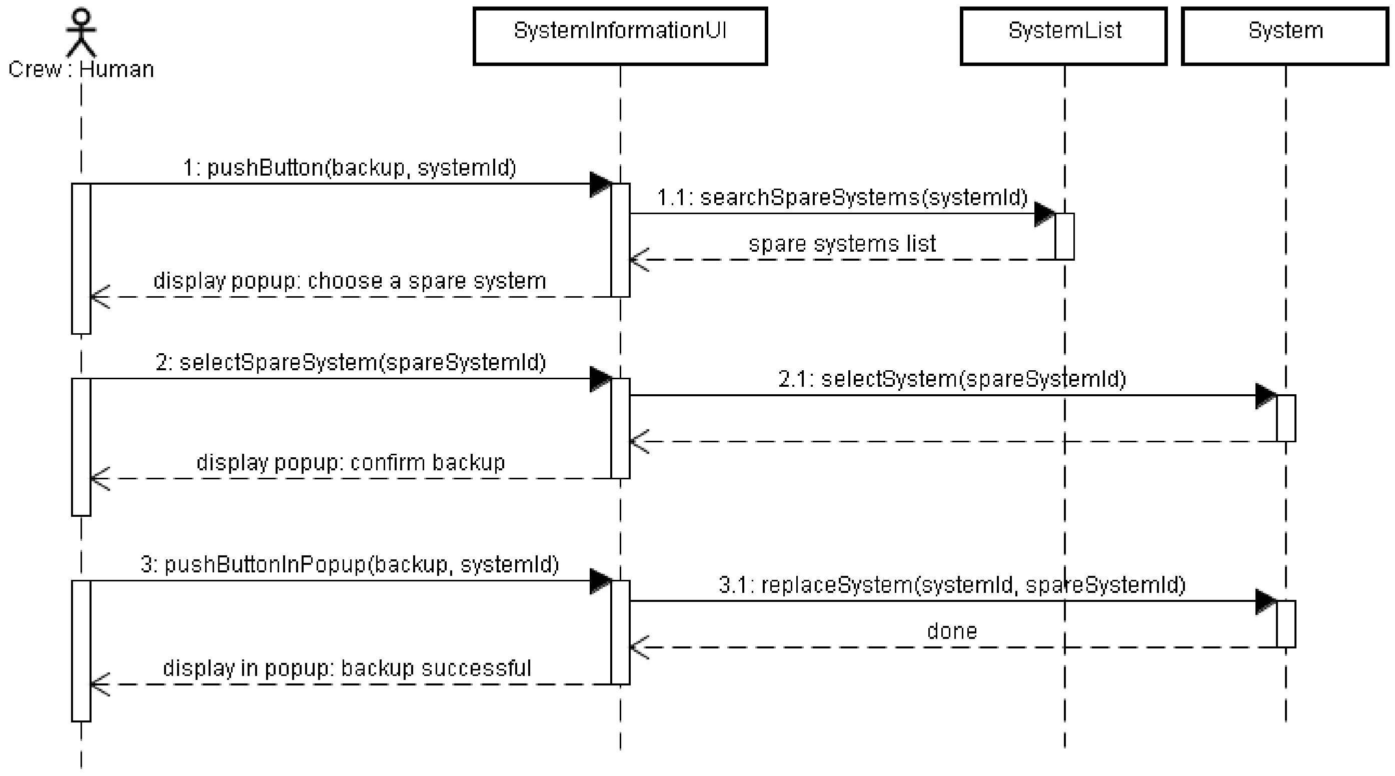

Crew : Human
SystemInformationUI
SystemList
System
1: pushButton(backup, systemId)
1.1: searchSpareSystems(systemId)
spare systems list
display popup: choose a spare system
2: selectSpareSystem(spareSystemId)
2.1: selectSystem(spareSystemId)
display popup: confirm backup
3: pushButtonInPopup(backup, systemId)
3.1: replaceSystem(systemId, spareSystemId)
done
display in popup: backup successful

Figure 16: Sequence diagram - Switch to alternate system device

3. Non-functional requirements

3.1. Identified quality attributes

Attributes	Explanations	Priority
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]	★★★
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]	★★★
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]	★★★

3.2. Improvised quality attributes

Attributes	Explanations	Priority
Availability	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.	★★☆
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.	★★★
Resilience	The system must be able to maintain an acceptable level of service in spite failures in parts of the FDIR system.	★★☆
Performance: **Response Time**	The system must respond in timely manner so that problematic systems can be shut down before any damage is done.	★★☆
Performance: **Thoughput**	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.	★★☆
Usability: **Recoverability**	FDIR has to limit errors of manipulation from users, considering the criticism of operations on a spacecraft. No operation should be irreversible, and confirmation should be asked to user each time he wants to do an action. Flow of information should be filtered to allow user to only focus on interesting parts, diminishing so the probability of mistakes to happen.	★★☆
Usability: **Learnability**	FDIR is a professional application that treats a lot of information and needs a good efficiency from the users. However, system must provide easy-to-learn features, by displaying confirmation and error pop-ups, and by providing easy-to-understand views.	★☆☆
Usability: **Consistency**	The FDIR interface should provide several views with consistent naming conventions and features from one view to another.	★☆☆

4. Usability analysis & design

4.1. Preliminary user interface design

Interface for the FDIR system has to be accurate, efficient and exhaustive. As a professional software, FDIR doesn't need to be particularly easy to learn or user friendly, that is why our design is more focused on interface performance and on displaying the more information possible while keeping it logic and accessible.

FDIR system has to display continuous information about monitored values, systems state, interventions (by both human and computer), errors and warnings. That is why we don't want to hide any information as possible. We are using 3 main screens, supposed to be displayed on three different screens. We can focus on a specific part by using tree architecture containing a list of every systems, sub systems, and devices. Selecting a part affects the state of every screen, it is updating the allowed actions, it filters information on the log screen, and it is updating the spacecraft scheme in order to provide geo localization. User is also allowed to select the entire spacecraft. Then, any information about it will be displayed.

Another feature not described in the following prototypes is the case of a critical issue happening, FDIR will so have to provide an alert (like a pop-up system) so that the crew will be directly informed.

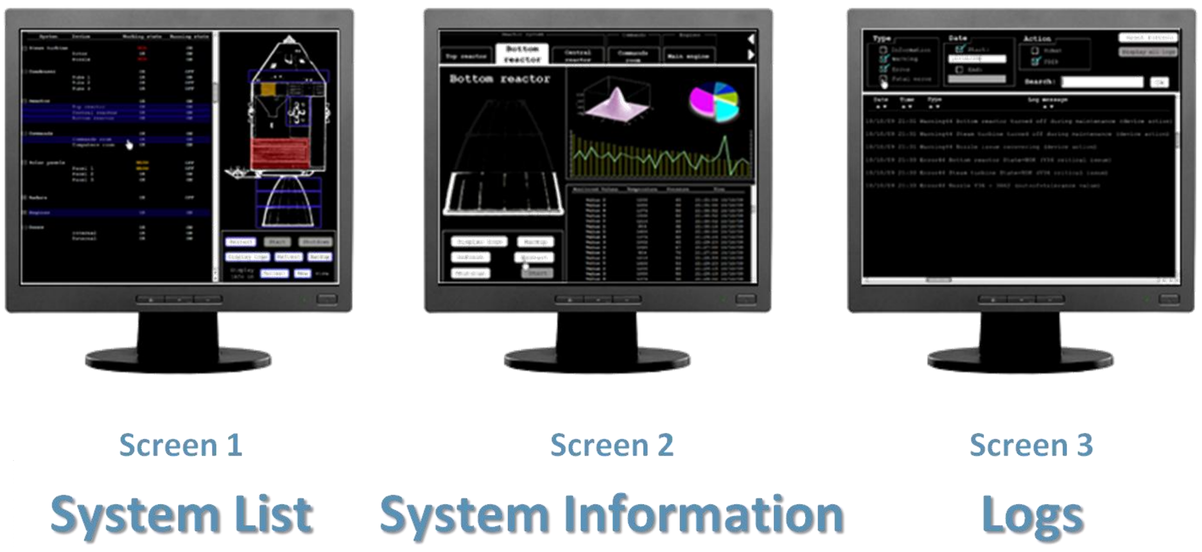
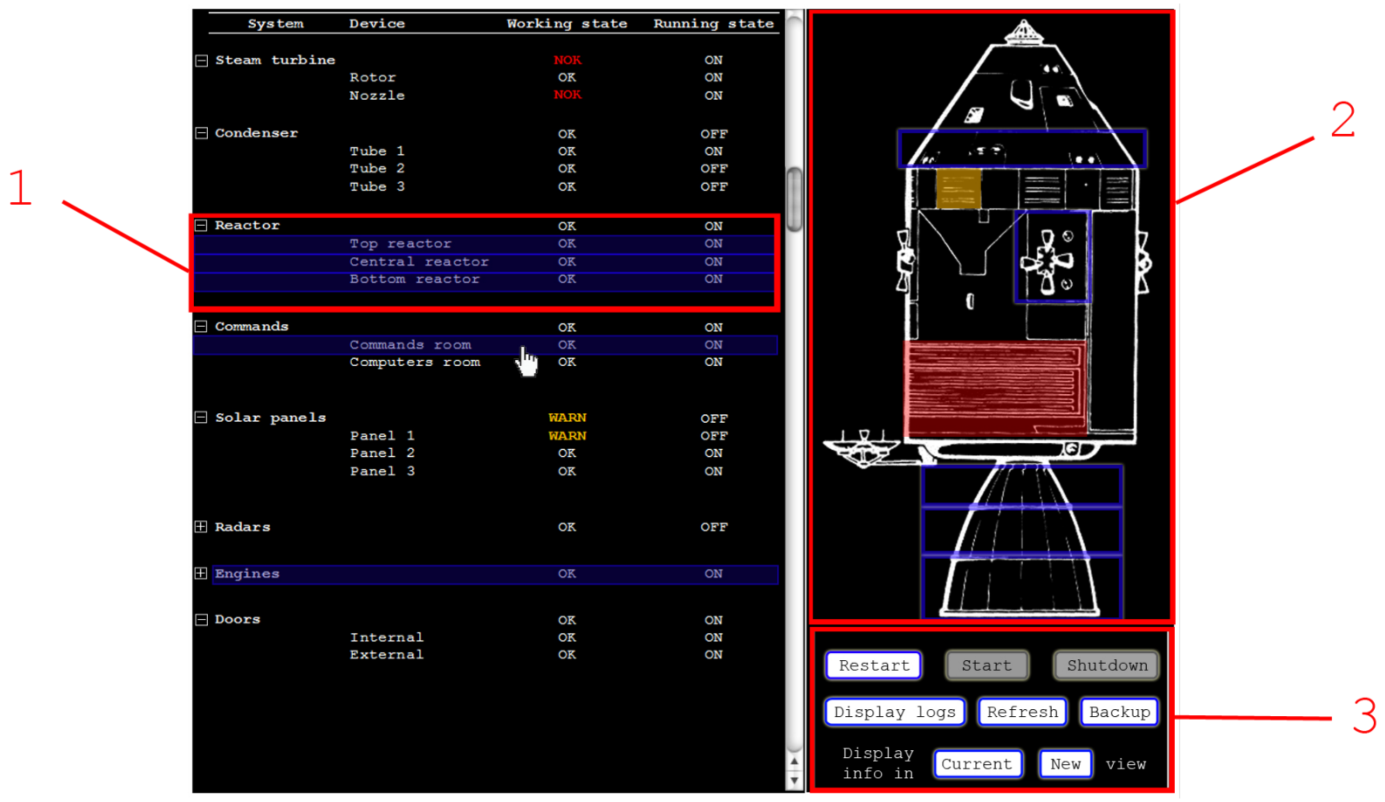


Figure 17: User Interface architecture

4.1.1. Screen 1: System list



System	Device	Working state	Running state
⊟ Steam turbine		NOK	ON
	Rotor	OK	ON
	Nozzle	NOK	ON
⊟ Condenser		OK	OFF
	Tube 1	OK	ON
	Tube 2	OK	OFF
	Tube 3	OK	OFF
⊟ Reactor		OK	ON
	Top reactor	OK	ON
	Central reactor	OK	ON
	Bottom reactor	OK	ON
⊟ Commands		OK	ON
	Commands room	OK	ON
	Computers room	OK	ON
⊟ Solar panels		WARN	OFF
	Panel 1	WARN	OFF
	Panel 2	OK	ON
	Panel 3	OK	ON
⊞ Radars		OK	OFF
⊞ Engines		OK	ON
⊟ Doors		OK	ON
	Internal	OK	ON
	External	OK	ON

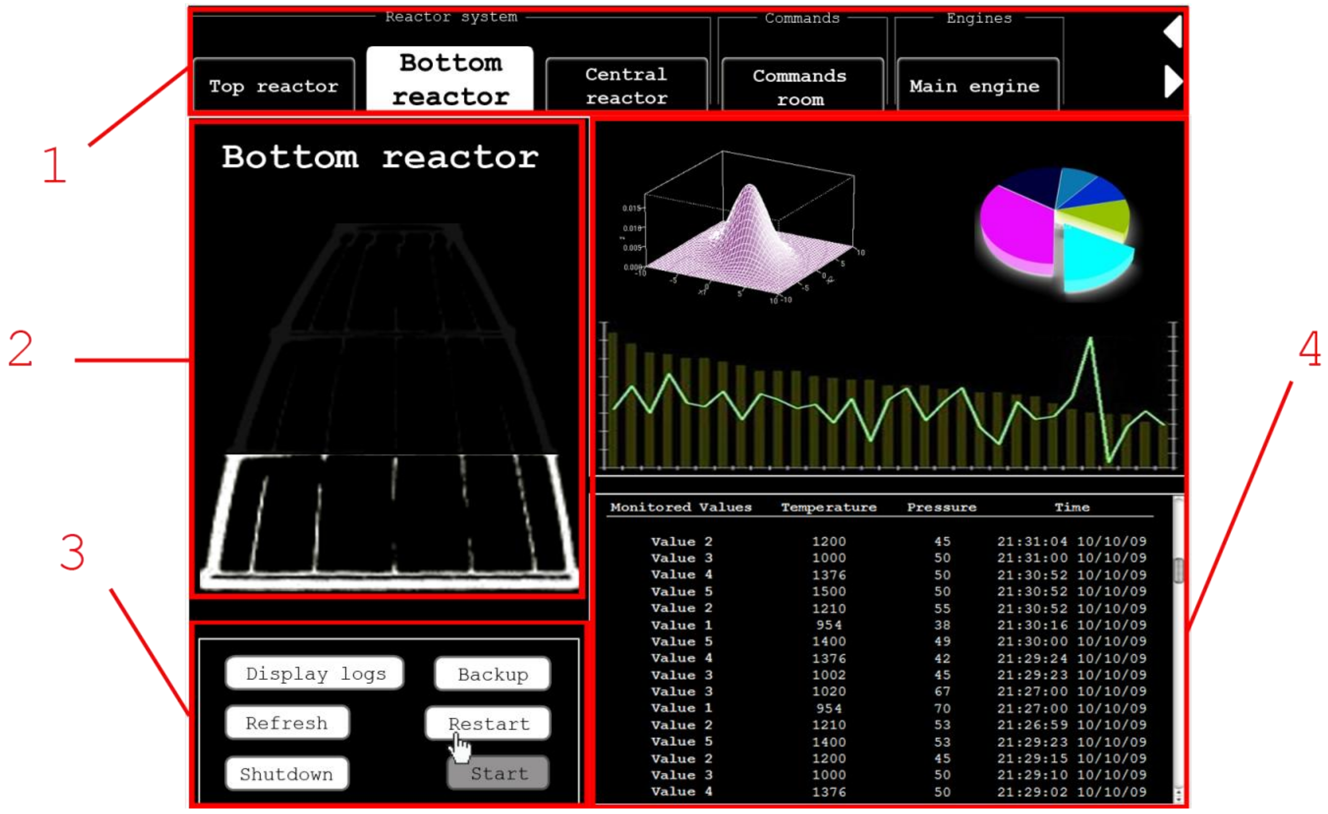
Figure 18: Systems list window prototype

1 In this tree panel we display every system, divided into several devices. Working states and running states will always be displayed in this tree. The user can make a multi-selection between several systems and devices, affecting the other two panels.

2 This panel gives a geographical localization of the selected parts to the user. This is useful in the case of a need of a fast human intervention on some part of the spacecraft.

3 This action panel displays every allowed action considering the selected parts. User is so able to start, restart, or shutdown a part or several parts of the system at any time. He can also ask for a manual and direct refresh of the states, he can operate a change on the current used backups (a pop-up window will so appears to permit him to select which backups are currently working on the selected parts), he can make a filter on the log screen by considering only components selected on this view, and he can display monitored values corresponding to these components on the appropriate screen (on the current tab or opening a new one).

4.1.2. Screen 2: System Information



Monitored Values	Temperature	Pressure	Time
Value 2	1200	45	21:31:04 10/10/09
Value 3	1000	50	21:31:00 10/10/09
Value 4	1376	50	21:30:52 10/10/09
Value 5	1500	50	21:30:52 10/10/09
Value 2	1210	55	21:30:52 10/10/09
Value 1	954	38	21:30:16 10/10/09
Value 5	1400	49	21:30:00 10/10/09
Value 4	1376	42	21:29:24 10/10/09
Value 3	1002	45	21:29:23 10/10/09
Value 3	1020	67	21:27:00 10/10/09
Value 1	954	70	21:27:00 10/10/09
Value 2	1210	53	21:26:59 10/10/09
Value 5	1400	53	21:29:23 10/10/09
Value 2	1200	45	21:29:15 10/10/09
Value 3	1000	50	21:29:10 10/10/09
Value 4	1376	50	21:29:02 10/10/09

Figure 19: System Information window prototype

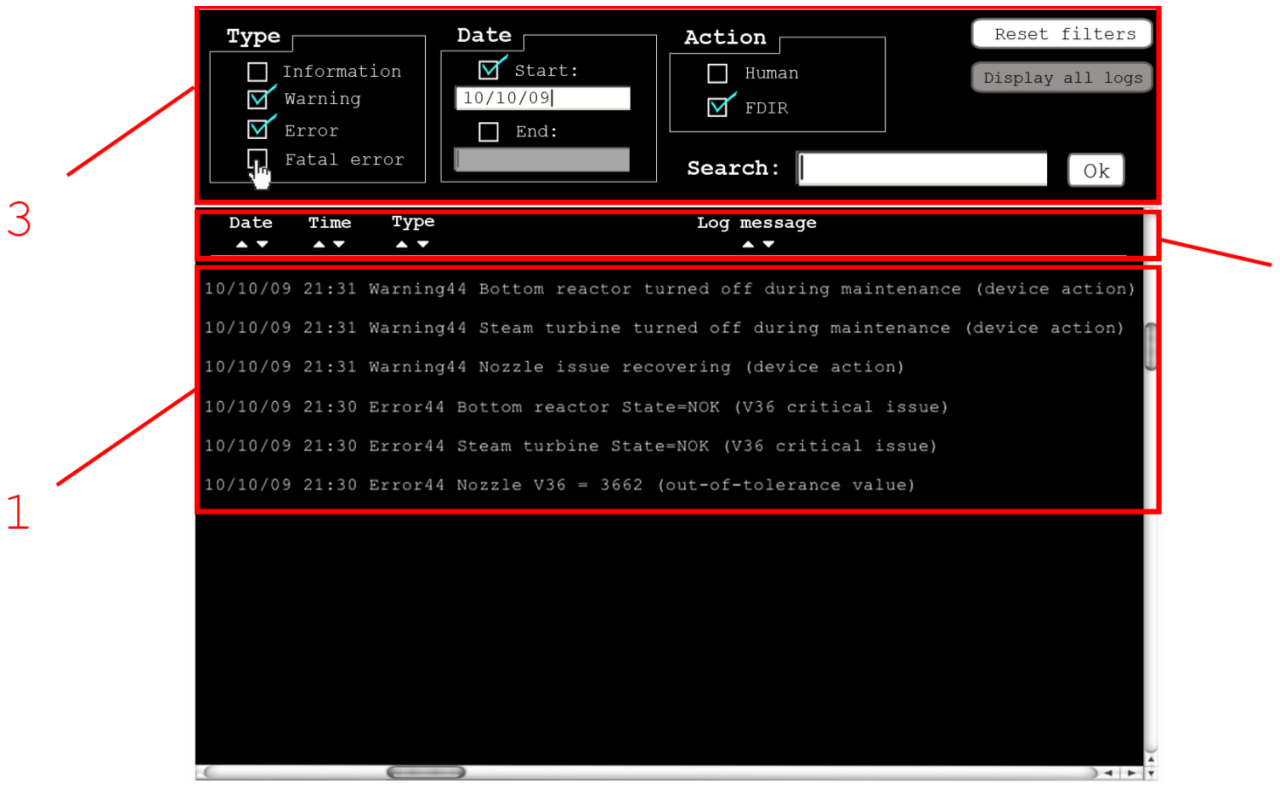
1 This tab system allows the user to manage this view by making groups of monitored systems/devices. We add new tab or new systems/devices to the current tab thanks to the system-displaying window, and we can remove tabs or groups of tabs on this window (not represented on the prototype).

2 This view offers a more accurate geographical view on the selected part, while the general view is always kept on the other window.

3 This action panel is basically the same that what was presented in the previously screen. The difference is that here we are considering a single part of the global system, no multi-selection are allowed.

4 Every monitored value is displayed here. We can sort them by period, increasing values etc… FDIR is also automatically generating stats, charts, and miscellaneous information, based on the monitored values.

4.1.3. Screen 3: Logs



```
10/10/09 21:31 Warning44 Bottom reactor turned off during maintenance (device action)
10/10/09 21:31 Warning44 Steam turbine turned off during maintenance (device action)
10/10/09 21:31 Warning44 Nozzle issue recovering (device action)
10/10/09 21:30 Error44 Bottom reactor State=NOK (V36 critical issue)
10/10/09 21:30 Error44 Steam turbine State=NOK (V36 critical issue)
10/10/09 21:30 Error44 Nozzle V36 = 3662 (out-of-tolerance value)
```

Figure 20: Logs window prototype

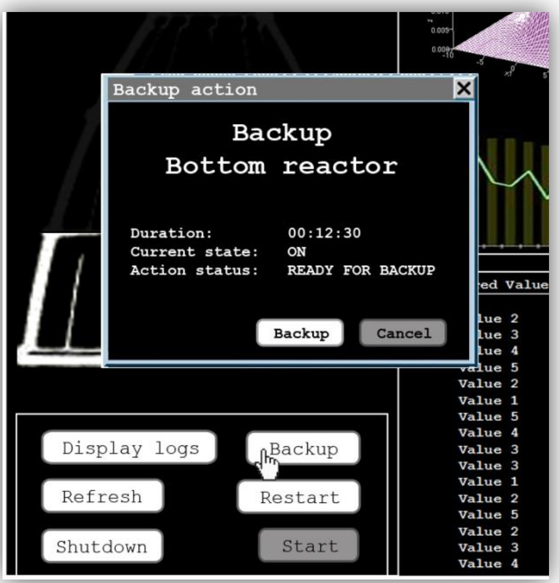
1 In this panel we display every log information produced by FDIR, by specifying date, time, id of message, concerned component, monitored value or action, and type of issue or type of action. If we don't apply any filter on its, that mean every information is accessible on this tab. We wanted to use a log system for this purpose, because this kind of tool is powerful and adapted to professional applications.

2 With this interface, user can sort displayed lines by date, time, type of information, and by name of message.

3 User is able to do filter operations in margin of the systems/devices filter operated on the system displaying window. He can choose between information type to display (information, warning, error, fatal error), and if displayed information is concerned by FDIR automatic intervention or human intervention. He can also make a filter by specifying dates of "start" and "end". The button 'Display all logs' cancel the effect of systems/devices filters, and the button 'Reset filters' cancel the effect of other filters. A search allow user to perform a research through the log, for example if he types the string "panel solar", only lines containing this string will appears on the first panel.

4.1.4. Screen 3: Backups management pop-up

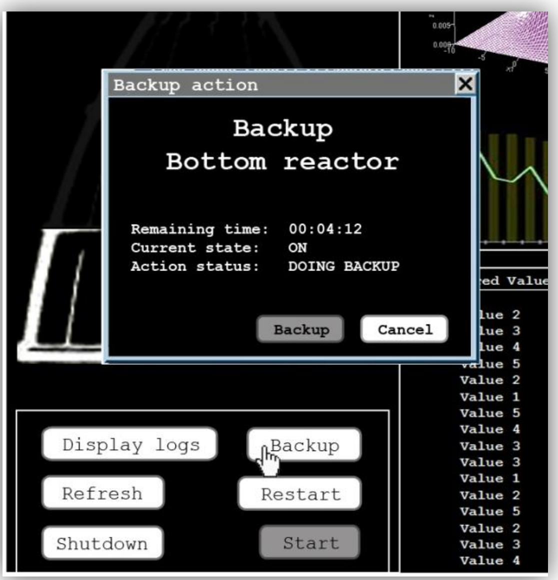
When the action "Backup" is selected, a window appears allowing user to select a spare system. This pop-up then appears, informing the user of the expected duration of the operation, and allowing him to cancel the operation if this one is too long or for some other reason.



```
Duration:        00:12:30
Current state:   ON
Action status:   READY FOR BACKUP
```

Figure 21: Backup management pop-up before backup

During the backup operation, this pop-up is still displayed, informing user of the remaining the action's achievement. User is allowed to cancel the operation for some reason, and then the system recovers automatically the original configuration.



```
Remaining time:  00:04:12
Current state:   ON
Action status:   DOING BACKUP
```

Figure 22: Backup management pop-up during backup

4.2. 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.

4.2.1. Scenario 24

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 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.

In screen 1 (*System List screen*) & 2(*Information system screen*), we can find out that we always have access to the same type of actions, as long as we are handling systems. The difference is that on the first screen we apply the action on every selected systems, while, on the second screen, it is only applied on the current displayed system.

4.2.2. Scenario 25

According to this scenario, we decided to create real screen interactions. At any moment, views are accessible and available. Besides, we never hide any information from user thanks to multiple screens feature. Moreover spreading the screens strengthen the real screen interaction and accessibility of the FDIR system.

In screen 1 (*System List screen*) & 2(*Information system screen*), we allow user to do several actions that affect the others screens. For example, clicking in Logs button on screen 1 or 2 will instantly change the view of screen 3 (*Logs screen*). Another example is the *display information in current or new view* buttons: pushing one of them will affect the second screen displaying several tabs about selected systems/devices on first screen. The system of tabs is never disabled in order to switch very quickly between views at any time.Tabs allow system to display all the information without hiding anything.

4.2.3. 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 Earthwhile 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.

4.2.4. Scenario 16

In order to be efficient and fast, the system should provide functions to navigate within a single view. That's why we create the tree with the list of systems and all devices in one screen, to access every system in a fast way. Besides, we provide some filters in the log screen, in order to accelerate the research of the both actors.

4.2.5. Scenario 26

Our user interface is relating to this scenario thanks to the multiple FDIR screens. Indeed, FDIR system is providing different viewpoints and several ways in order to display system information and give the user a better insight.

Within the screen 1 (System list screen), state and location (in the spacecraft) of a system/device are displayed. Within the screen 2 (Information system screen), monitoring values and statistics of this one are shown. Within the screen 3 (Logs screen), logs of this same system are available.

To conclude we see that information concerning the same system/device is displayed in different ways (images, statistics, texts) and different views (screen 1, 2 and 3).

4.2.6. 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 21.

4.2.7. 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.

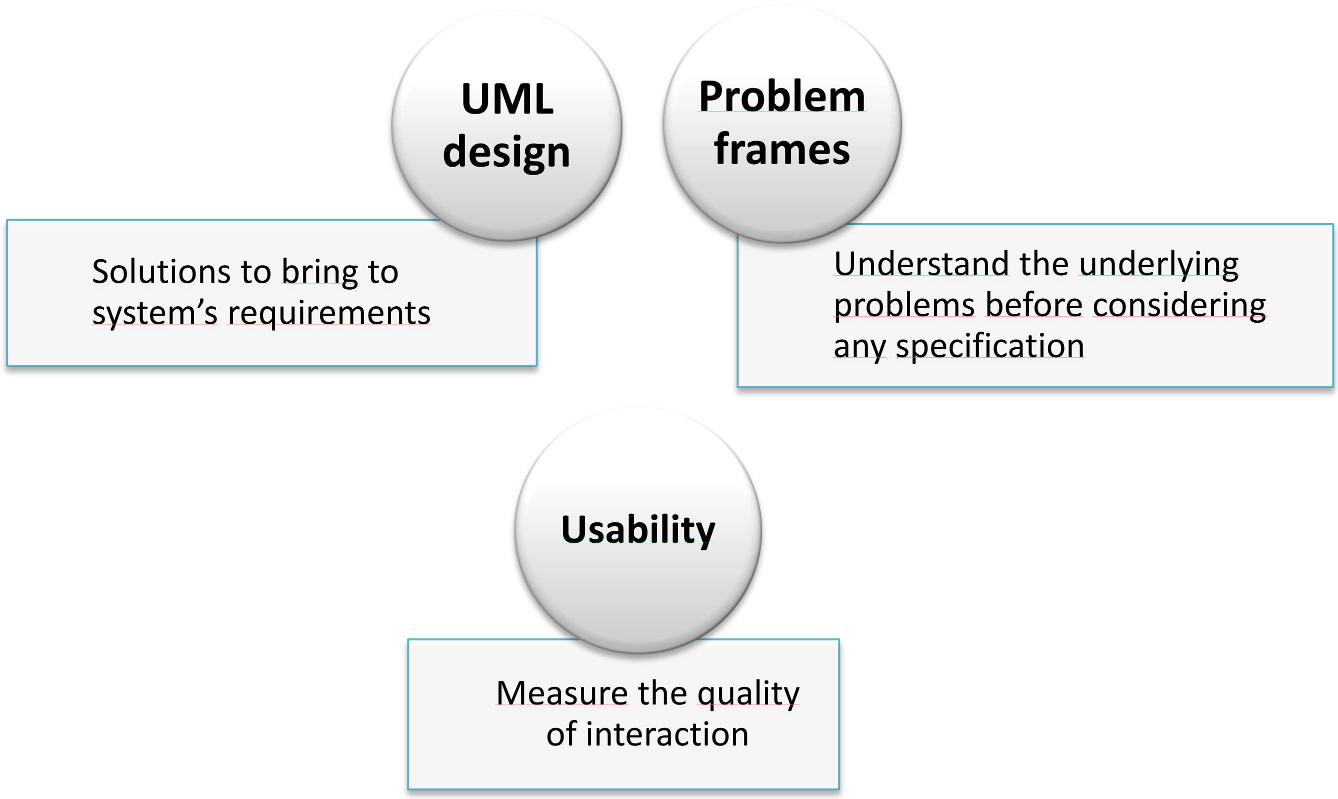

UML design
Solutions to bring to system's requirements
Problem frames
Understand the underlying problems before considering any specification
Usability
Measure the quality of interaction

Figure 23: Comparison about UML and problem frames methods

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-functional requirements
2. http://en.wikipedia.org/wiki/Problem_Frames_Approach, about problem frames

Books

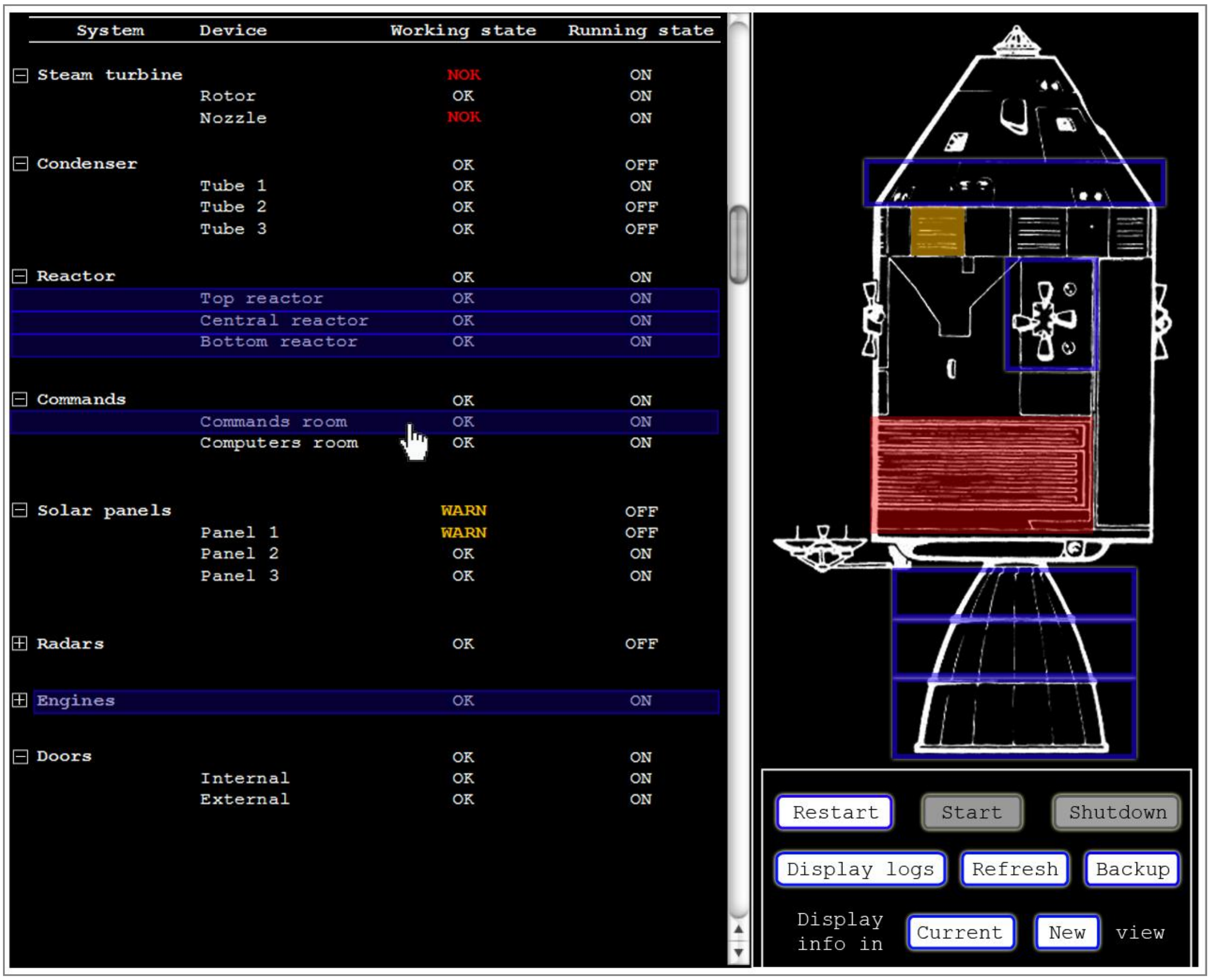
1. [Eas98] **Steve Easterbrook, and et al**., *Experiences Using Lightweight Formal Methods for Requirements Modeling* IEEE Transactions on Software Engineering, Vol. 24, No. 1, January 1998.

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.

3. [BJK01] **Len Bass**, **Bonnie John**, **Jesse Kates**, *Achieving Usability Through Software Architecture*, Technical Report CMU/SEI-2001-TR-005, Carnegie Mellon Software Engineering Institute, March 2001.

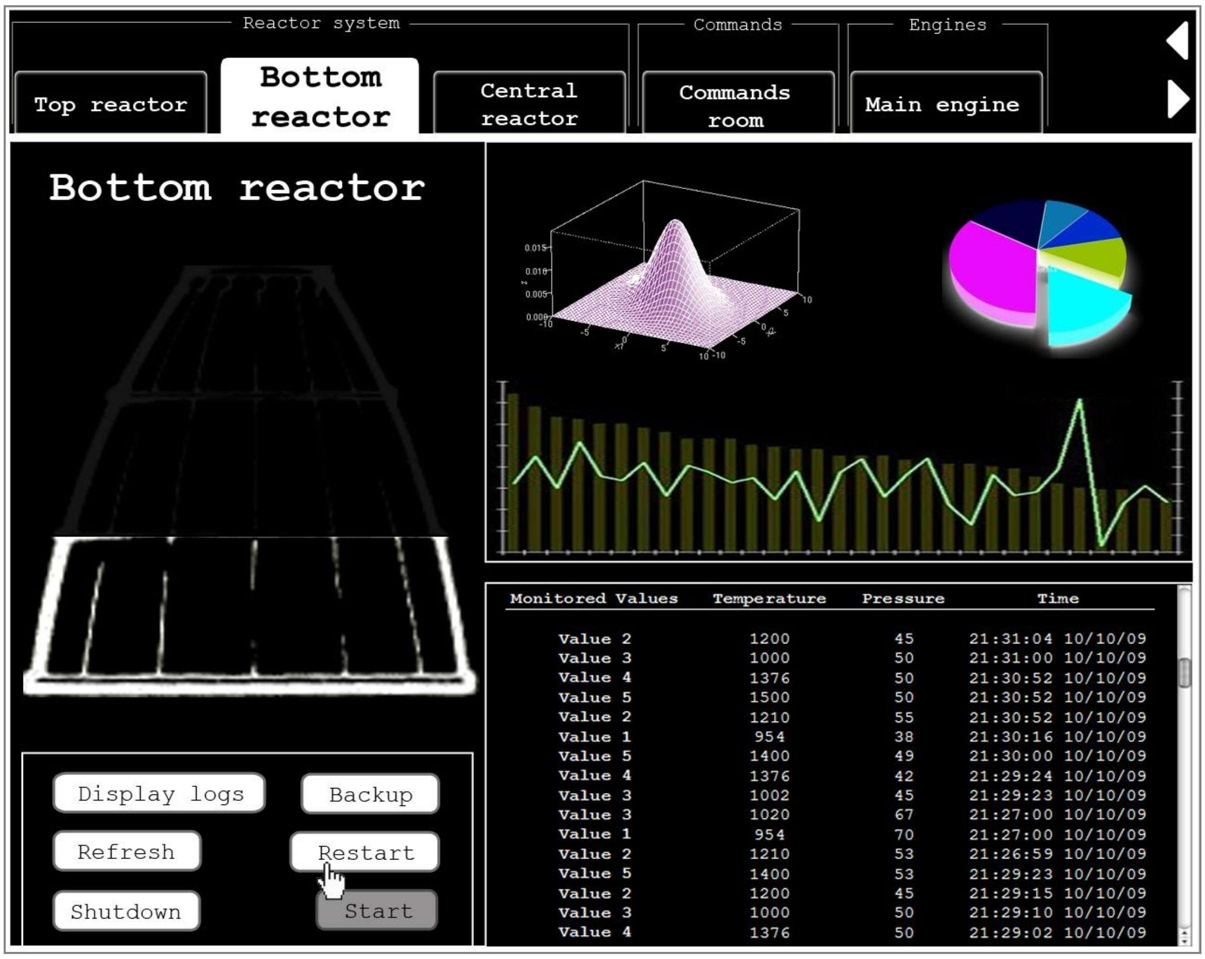
Annexes

1. **Annexe 1**: Screen 1 - System List



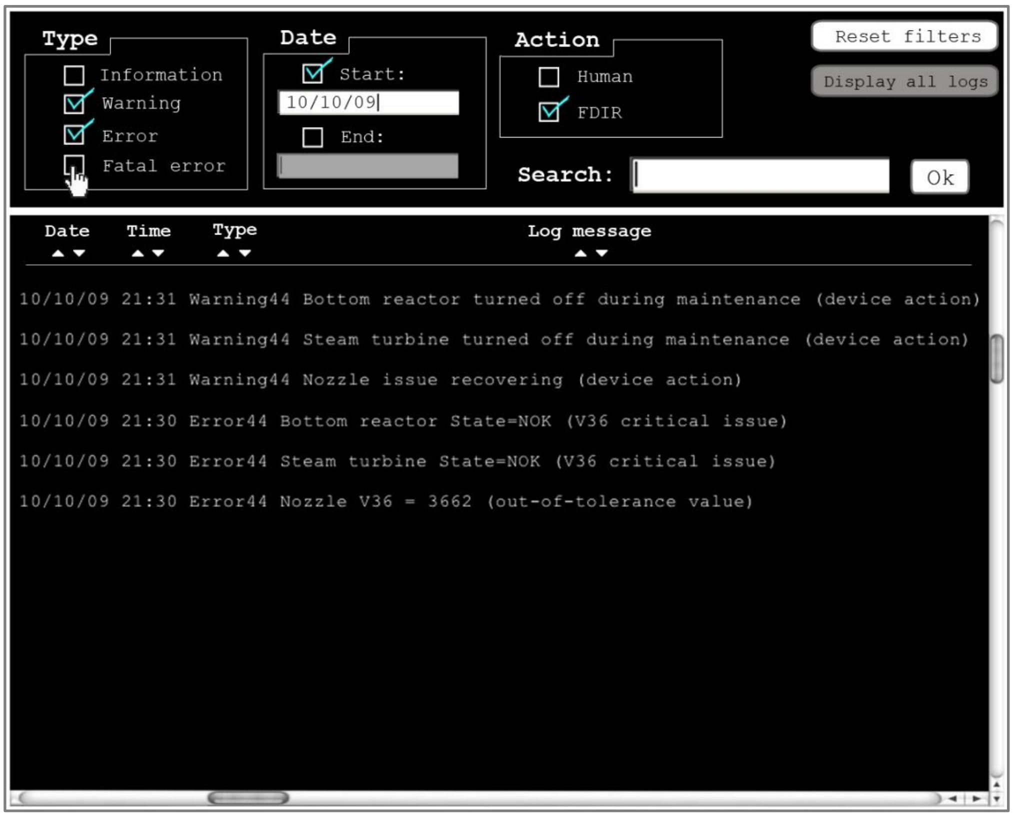
System	Device	Working state	Running state
⊟ Steam turbine		NOK	ON
	Rotor	OK	ON
	Nozzle	NOK	ON
⊟ Condenser		OK	OFF
	Tube 1	OK	ON
	Tube 2	OK	OFF
	Tube 3	OK	OFF
⊟ Reactor		OK	ON
	Top reactor	OK	ON
	Central reactor	OK	ON
	Bottom reactor	OK	ON
⊟ Commands		OK	ON
	Commands room	OK	ON
	Computers room	OK	ON
⊟ Solar panels		WARN	OFF
	Panel 1	WARN	OFF
	Panel 2	OK	ON
	Panel 3	OK	ON
⊞ Radars		OK	OFF
⊞ Engines		OK	ON
⊟ Doors		OK	ON
	Internal	OK	ON
	External	OK	ON

2. **Annexe 2**: Screen 2 - System Information



Monitored Values	Temperature	Pressure	Time
Value 2	1200	45	21:31:04 10/10/09
Value 3	1000	50	21:31:00 10/10/09
Value 4	1376	50	21:30:52 10/10/09
Value 5	1500	50	21:30:52 10/10/09
Value 2	1210	55	21:30:52 10/10/09
Value 1	954	38	21:30:16 10/10/09
Value 5	1400	49	21:30:00 10/10/09
Value 4	1376	42	21:29:24 10/10/09
Value 3	1002	45	21:29:23 10/10/09
Value 3	1020	67	21:27:00 10/10/09
Value 1	954	70	21:27:00 10/10/09
Value 2	1210	53	21:26:59 10/10/09
Value 5	1400	53	21:29:23 10/10/09
Value 2	1200	45	21:29:15 10/10/09
Value 3	1000	50	21:29:10 10/10/09
Value 4	1376	50	21:29:02 10/10/09

3. **Annexe 3**: Screen 3 - Logs



Type: [ ] Information [✓ Warning] [✓ Error] [ ] Fatal error

Date: [✓ Start:] 10/10/09 [ ] End:

Action: [ ] Human [✓ FDIR]

```
Date     Time   Type                                Log message

10/10/09 21:31 Warning44 Bottom reactor turned off during maintenance (device action)

10/10/09 21:31 Warning44 Steam turbine turned off during maintenance (device action)

10/10/09 21:31 Warning44 Nozzle issue recovering (device action)

10/10/09 21:30 Error44 Bottom reactor State=NOK (V36 critical issue)

10/10/09 21:30 Error44 Steam turbine State=NOK (V36 critical issue)

10/10/09 21:30 Error44 Nozzle V36 = 3662 (out-of-tolerance value)
```