

Politecnico Di Torino

Mechatronics Engineering

Master's Degree Thesis



Development of a demonstrator for automatic test execution using LabVIEW and XML-RPC server

Student: Joshua COUTURIER

Academic supervisor: Marco GHIRARDI

Company tutor: Olivier ALQUIER

September 2018

Credits

I wish to take this opportunity to thank all those who contributed to the success of my thesis, and who helped me in the realization of my project.

I would like to thank my tutor Olivier ALQUIER, head of the "automatic test" activity within Assystem Technologies, for his hospitality, his pedagogy and his investment in the realization of my thesis, as well as his advice on personal and professional plans.

Then, I thank David CORAIN, technical manager, and Ivan LEFRANÇOIS, director of the agency Systems Engineering, for having welcomed me and allowed me to realize this thesis.

Without forgetting to thank the whole « automatic test » team, including Pierre CHAUMONT, Jonathan MASSON, Franck LEGAL and Frédéric FACCHETTI, for training me to the job of tester, and for taking time to explain their activities with pedagogy and attention.

Finally, I would like to thank all the Assystem Technologies employees with whom I have worked, including Sébastien ANSCIEAU, for their expertise, their team spirit, their patience and their knowledge.

Content

Introduction	1
I) Company presentation	2
1) History	2
2) Organization	4
3) Key numbers and values	5
II) Automatic testing context	6
1) Need to automate tests	6
2) Solution	7
III) Realization of a demonstrator	10
1) Needs Analysis and Specifications	10
2) Selected tool	11
3) Development and integration of a new brick	13
a) Structure	13
b) Operations	14
c) XML-RPC server	18
d) Configurable	19
4) Deployment and results	20
5) Challenges and difficulties	24
IV) Complementary activities	25
1) Validation of GTA	25
2) Training	25
3) Writing of documentation	26
V) Conclusion	27
Appendixes	28
A. Xml configuration file	28
B. Gantt diagram	29
C. Poster	29
D. Developer guide	31

Introduction

As part of my mechatronics engineering master's degree at Politecnico di Torino, I realized my 6 month thesis by integrating the company Assystem Technologies in Toulouse, France. As an international engineering and innovation consulting group, Assystem Technologies assists major industrialists in the various stages of their projects.

During this thesis, I joined the "automatic test" team, which is in charge of the deployment and monitoring of automatic tests. Their main customer, Airbus, adopted a few years ago a test automation policy that reflects the progress of technology and industrialization.

Airbus uses a software suite for automatic testing that allows you to write and execute ground test procedures. The benefits of tests automation are the exploitation of downtimes and the increase of the reproducibility of tests, thus a reduction of cost thanks to a better efficiency.

GTA (Generic Tool for Automation) is one of the software used on Airbus test benches for writing and executing test procedures, as well as generating a report. It makes it possible to communicate with the different tools of the test benches (robot arm, image recognition ...). It is developed by Airbus India, which relies on the support and expertise of Assystem Technologies. Indeed, the team that I integrated had previously developed an automation software, since 2014, called GENESTIC (GENERIC Software for Test Automation).

Today, Assystem Technologies does not have the intellectual property of this automation project, and therefore can only use this software suite (GTA) in connection with Airbus, although having many skills and knowledge on it.

My mission was therefore to develop an automated test demonstrator, relying on the skills of Assystem Technologies, which will export the capabilities of the company to other customers than Airbus.

The first part of the report is dedicated to Assystem Technologies: its history, organization and key figures.

The second part presents the definition and the stakes of the automatic tests, in particular the system tests and the functionalities of the GTA software.

The third part details the realization of my mission, that is to say the realization of the demonstrator: from the analysis of needs to the results obtained.

Finally, the last part describes the other activities carried out during the thesis, namely the introduction to the job of tester, the various trainings carried out, and the writing of documentation.

I) Company presentation

1) History

"Assystem Technologies specializes in product engineering and post-development services, innovation and digital transformation consulting, and quality assurance. Assystem Technologies is distinguished by its high level of technical know-how and proven expertise in complex and critical systems, on behalf of industrial customers operating in the aerospace, defense, automotive and transportation sectors"- Generic presentation template from Assystem Technologies.

Historically focused on nuclear power, Assystem Technologies has diversified in favor of the aerospace, automotive and new technologies sectors. Gradually, it has grown internationally. The history of the society can be divided into five main periods:

- From 1966 to 1995, the nuclear years:

In 1966 the company ATEM was created, specialized in the organization of the commissioning of industrial units (nuclear, iron and steel ...).

A few years later, in 1989, Alphatem, a subsidiary of Cogema dedicated to nuclear power, was also created.

Then in 1995, the two companies ATEM and Alphatem merged to give birth to the company Assystem. At this time, Assystem's business is nuclear-centric and the company is listed on the stock market.

- From 1996 to 2003, diversification:

In this period, Assystem is evolving and expanding its business sectors through takeovers and mergers of companies.

Indeed, in 1996, Assystem bought the company Studia aerospace and automobile. Then, in 2003, the company enters the sectors of new technologies by merging with the company Brime Technologies.

- From 2005 to 2010, international development:

After having diversified, Assystem very quickly understood the need to open up to new countries and therefore started an expansion abroad.

In 2005, Assystem acquired the British engineering group Inbis, SKI and Atena (a subsidiary of the German company MTU Aero Engines).

In February 2008, Atena and Silver Software merged to create Silver Atena (a company specializing in the design of critical-security electronic and computer systems).

In 2010, the strategic alliance "n.triple.a" was set up with the British company Atkins to serve the market of emerging countries in nuclear engineering.

- From 2011 to 2017, new growth in energy and embedded systems:

In recent years, Assystem focused more on energy, embedded systems and digitalization sectors, acquiring numerous companies.

In 2011, it acquired the German company Berner & Mattner, specialized in embedded systems.



In 2012, Assystem acquired the MPH family group, which specializes in petroleum and natural gas engineering.

Finally, in 2016, Assystem acquired ENVY (a Turkish company specializing in the energy and transport sectors), Onyx Promavi (a French company specializing in the environment, transport and energy sectors, and defense), Aerotec Concept (a player in the aircraft and helicopter modification and adaptation market) and Batir Group (nuclear civil engineering).

The figure 1 summarizes the history of Assystem from 1996 to 2016 (in French):

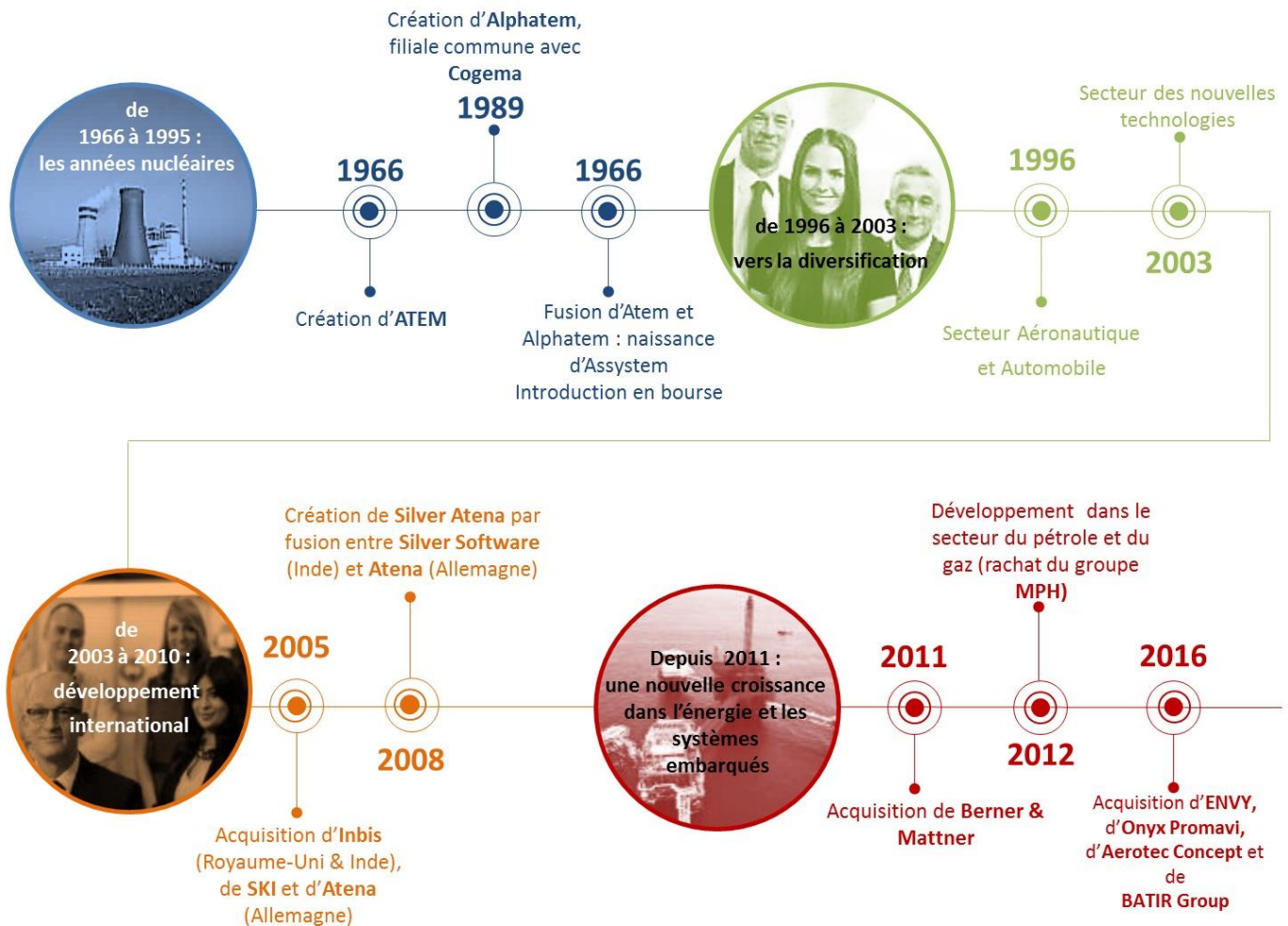


Figure 1: Assystem Technologies' past

- Since 2017, Assystem Technologies has become independent:

In September 2017, Assystem sells to private investment firm Ardian 60% of its outsourced R&D division Global Product Solutions.

In February 2018, Assystem Technologies successfully completes its takeover bid of the SQS Group, a leading player in digital transformation and quality assurance, with more than 8,000 employees.

This acquisition led to the creation of Assystem Technologies, consisting of a part of Assystem, and SQS. The other part of Assystem became Assystem energy & infrastructure.

In April 2018, Assystem Technologies successfully completed the acquisition of Stirling Dynamics Limited ("Stirling") and consolidated its position as a leading international partner for aerospace and marine industry players.

Today, Assystem Technologies' teams assist major industrialists from different sectors to reduce the costs and deadlines of their projects, to optimize their development, manufacturing and marketing processes, and to make possible design and production of innovations all over the world.

2) Organization

After refocusing the company on Assystem's nuclear side and selling its GPS division to Ardian investment company, Olivier Aldrin took over the presidency of the new Assystem Technologies group.

Figure 2 below details the overall organization of the company.

My internship took place in the France Sud-Ouest Region, in Toulouse, within the Systems Business Unit, in the digitalization part. I was part of the team of the automatic test, composed of 5 people when I arrived.

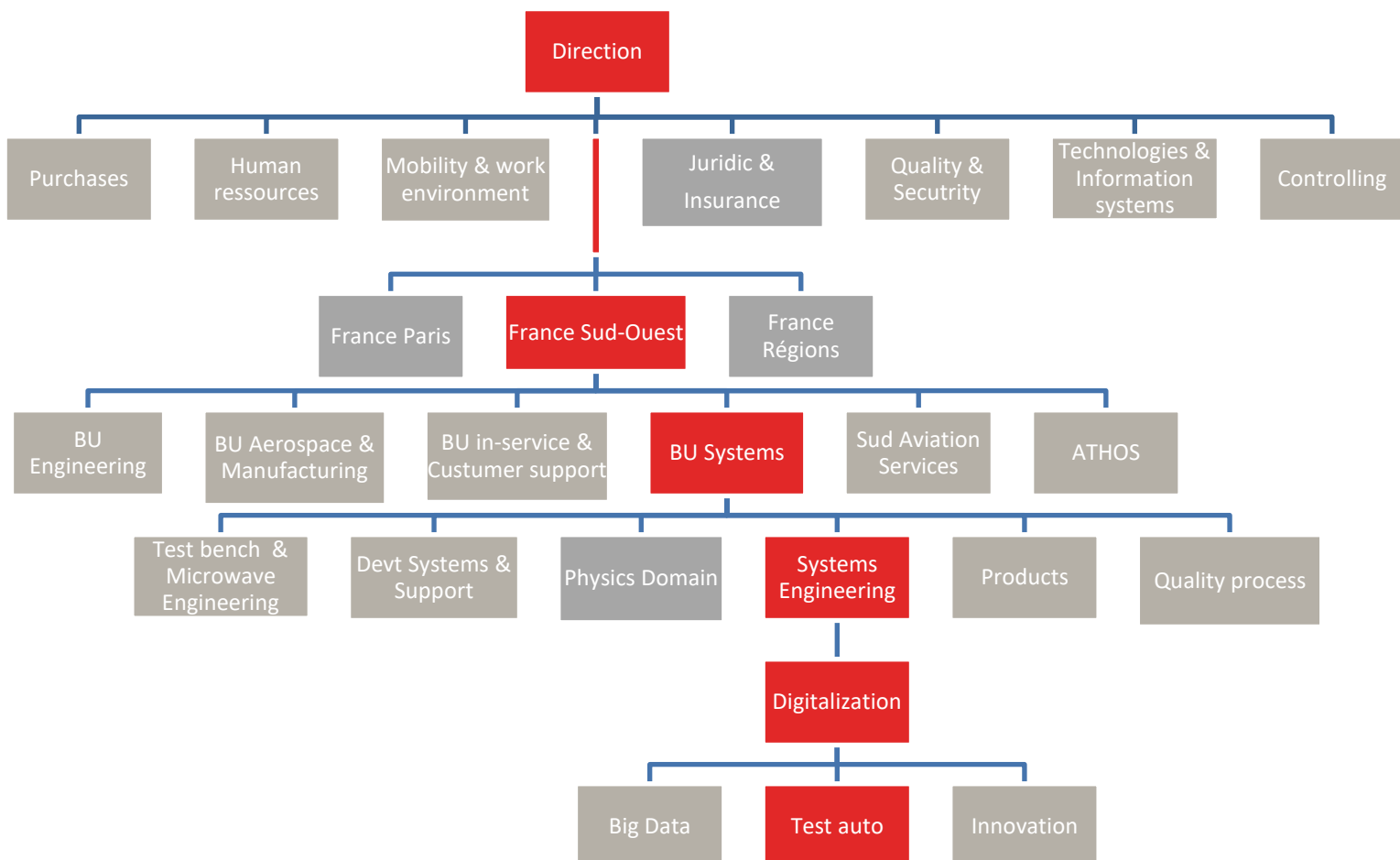


Figure 2 : Assystem Technologies organizational chart

3) Key numbers and values

Although originally nuclear-centric, Assystem Technologies has diversified over time and currently has a broad range of skills. Today, it is independent of the historical energy section, but remains focused on many areas, namely:

- **Aerospace**
- **Automotive**
- **Defense**
- **Rail**
- **Industry**
- **Naval**
- **Space**

For major domains, Figure 3 details the main clients:



Figure 3 : Main clients

In 2016, Assystem had close to 12,500 employees (including 4,500 in the current Assystem Technologies) in 20 countries, with a turnover of 950 million euros. In 2017, Assystem Technologies had more than 14,000 employees in 25 countries and a turnover of more than 1 billion euros.

In terms of values, the company is committed to recruiting at least as many women as school promotions allow, and a significant number of people with disabilities. This is why Assystem Technologies is part of the "women of energy" network and has set up the "handicap" mission. With these commitments, the company reaches a rate of 22% of women in new hires and more than 3% of its employees are disabled. Finally, the values of Assystem Technologies are Respect, Resilience, Perennial Performance, Responsibility, and the Spirit of Entrepreneurship. Its vision is growth afterwards, that is to say, development driven by innovation and global cooperation, articulated around strong human values.

II) Automatic testing context

1) Need to automate tests

In today's industry, performance measurements are made throughout the life cycle of a product through testing means. These can be used to fine-tune certain settings, to ensure compliance with specifications, to carry out maintenance operations or to understand possible malfunctions.

For the aerospace industry, with every new standard delivery, testers often have to repeat the same test procedures. As long as they are systematic, repetitive and automatable, it is preferable to use the automation of tests. The stakes of this new activity are important: they make it possible to gain in efficiency, in coverage of tests, in quality and in reproducibility. And let's not forget that automated tests can also be launched at night, saving time and reducing delivery times.

Quite simply, during the automation of the tests, a sequence of commands is executed to control various tools / instruments of a test bench, and a check of the status of each one is done, which indicates its good behavior or not.

The Airbus group has been updating its methods for about 5 years, to test its avionics and aircraft systems. We are mainly talking about "integration" tests, which consist in integrating the equipment to be tested into a "complete airplane" environment and observing its behavior. To test avionics systems, the Airbus Group performs two types of tests: flight tests (40%) and ground tests (60%). Concerning the ground tests, depending on the number of real / simulated elements, different test methods are used:

- **Test benches:** these are the test facilities that contain the most simulated elements. They can only test a few avionics systems
- **Simulators:** they consist of a real cockpit, and several real avionics systems. They also present a simulated part (aerodynamic engines, APU, air sampling, landing gear ...)
- **Iron bird:** it is a steel tubular structure receiving all the electrical and hydraulic systems and subsystems of the aircraft at scale 1, which allows in particular to represent the actual loss of loads of the aircraft

Figure 4 illustrates these different test means.



Figure 4 : testing facilities (top FIB2 NEO test bench, lower left A350 simulator and right A350 iron bird)

To obtain a major gain in productivity, this automation requires a certain rigor to anticipate each step of the flight and the possible reactions of the avionics systems concerned. Each instruction must be detailed to be transmitted by computer. The initial preparation is therefore slightly longer than for manual tests, for which only the main steps are written.

Despite these shortcomings, the advantages of automating tests are very important and this modernization represents a major challenge for aeronautical testing.

2) Solution

Since 2013, the test automation policy at Airbus has evolved. During its first steps, in 2013, Airbus relied on Assystem which was developing GENESTA (GENERIC Software for Test Automation), still present on some test benches. This software enables the writing of test procedures, the compilation of executable scripts on Airbus test benches, as well as the automatic generation of a test report.

Faced with the success of GENESTA, Airbus decided to develop its own automatic test software in its Indian subsidiary, GTA (Generic Tool for Automation), and delegated its follow-up to Assystem Technologies.

To date, GTA is used on twenty benches, by a community of more than 100 testers. Its history is presented in figure 5 which follows (in French).

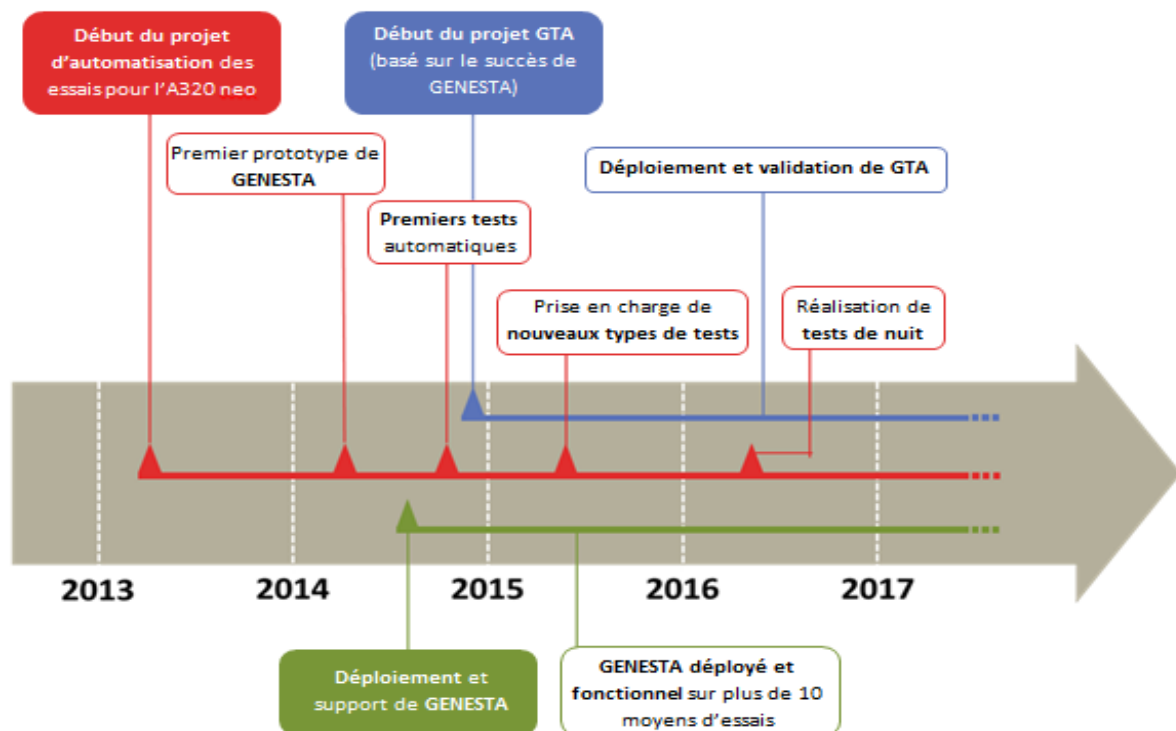


Figure 5 : Beginnings and development of GTA from 2013 to 2017

Today, GTA allows:

- ✓ The writing of test procedures, in a simple and understandable way by the users
- ✓ The generation of an SCXML file which is a language understandable by the various tools of the test bench, describing the instructions written in the procedure
- ✓ Writing a log file that lists the different stages of the test and the tool returns
- ✓ The generation of a test report in Word format, readable and understandable, to determine the correct functioning or not of the test, according to a status OK or KO

The tools are driven by various XML-RPC servers. That is, they use a Remote Procedure Call (RPC) protocol, which makes it possible to call a function on a remote server from any system connected to the network. The operation of this type of server will be detailed later. Among the controllable tools, we find:

- **The BIS-G** (Generic System Integration Benchmark) server that serves as a communication interface with the bench. It centralizes the data of real and simulated elements, connects equipment and manages communications
- The **BITE Logger** and the **Flight Warning Computer (FWC) Logger** which are software for generating log files containing messages from the simulated MCDU and FWC
- **OneClick** which is a tool whose role is to start the machines, to launch the various software, to load a test configuration and to launch a simulation
- The **Interactive Controller** that manages the user interactions (display messages on the screen)
- **Image recognition (IRT)** which aims to check if a message is displayed on the cockpit screens

- The **robot arm**, which is a tool for replacing manual actions (pressing a button, moving the levers, adjusting the rotary knobs, etc.) by automated actions

Figure 6 details the communication links between GTA and the various tools:

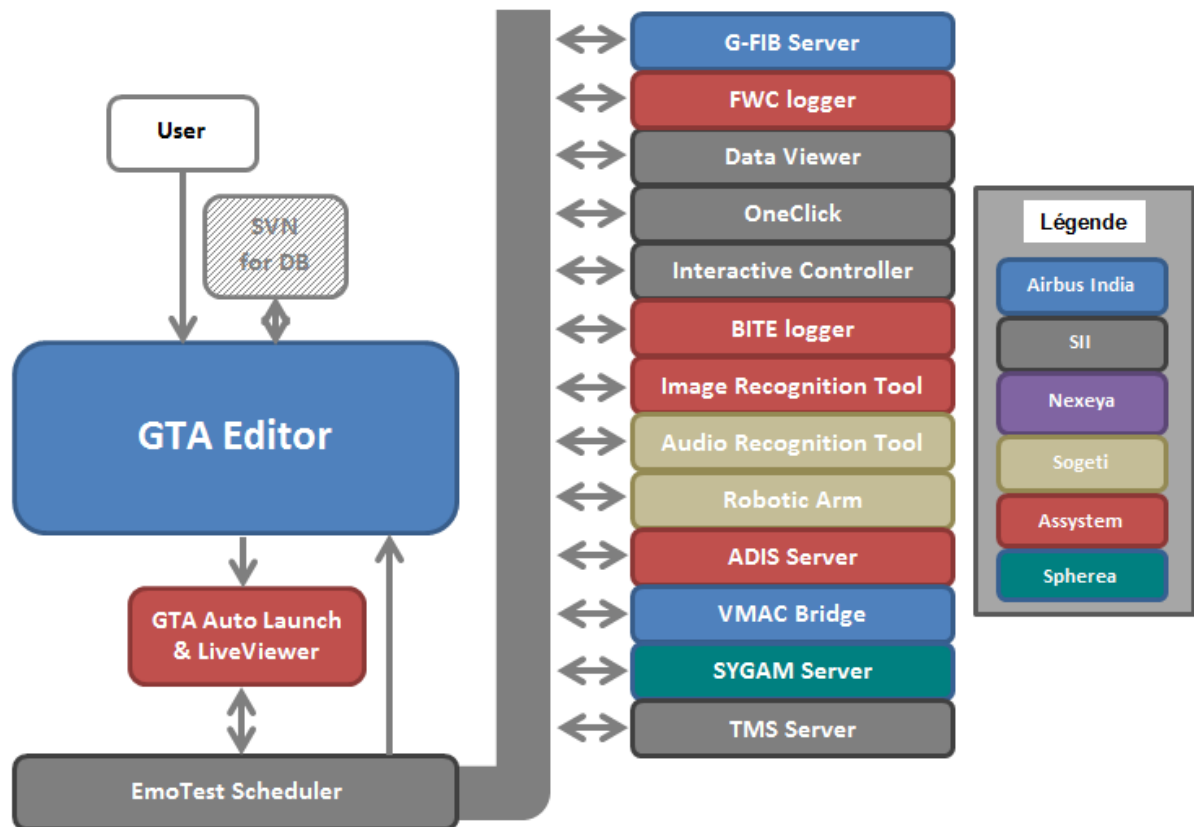


Figure 6 : Global architecture of the software suit

The growing demand to integrate GTA on new benches requires the installation of GTA and the training of new users, which are mostly conducted by the automated testing team. In addition, it is in charge of the validation of the good behavior of GTA, and the continuous improvement of the software (technically realized by Airbus India).

As we can see in Figure 6, the communication between the software and the tools via the XML-RPC server is a strong point that allows a great efficiency and ease of development which reflects the success of GTA. This is why we will retain this communication solution via XML-RPC server for later.

III) Realization of a demonstrator

1) Needs Analysis and Specifications

As seen in Figure 6 above, Assystem Technologies is only partially involved in the GTA project because it does not hold the intellectual property, but it allows the export of skills in the project at Airbus.

The idea of my internship is to be able to detach ourselves from the GTA solution of Airbus, to create a sequencer which Assystem Technologies would be the owner. The GANTT diagram in **Appendix B** details the major steps that were necessary for the development of this sequencer.

Firstly, I had to analyze the GTA software in details, in order to highlight the various integrated functions, and the needs it responds to. I then spent a few days reading the documentation on the software, understanding the context, and getting trained on its use by the team. I had access to both user-driven technical documentation and internship reports from previous years.

Then, my first initiative was to list the different possibilities of the software in order to compare them with the possible solutions of a sequencer. I then realized a table listing all the possible functionalities of GTA, simplified in figure 7.

Element concerned	Functionalities
General	<ul style="list-style-type: none"> ✓ Importing databases and tools ✓ Exporting a report in Word format
Header	<ul style="list-style-type: none"> ✓ Customization of the header ✓ Adding or removing fields
Document Viewer	<ul style="list-style-type: none"> ✓ Tree view of different types of data ✓ Editing and quick search of elements
Editor	<ul style="list-style-type: none"> ✓ Visualization of the procedure being edited ✓ Add an action, a check or a comment ✓ Features built into Action are: <ul style="list-style-type: none"> • [set] a parameter • [print] a message • [title] to structure the procedure • [condition] of the action • [manual action] to indicate to the operator to act • [call] an already existing element • [wait] • [external tool] to control tools ✓ Features built into Check are: <ul style="list-style-type: none"> • [value] for a parameter • [FWC warning] • [BITE message]
Result	<ul style="list-style-type: none"> ✓ Find a log, a procedure or a sequence ✓ Visualize the log
Editor Tool Bar	<ul style="list-style-type: none"> ✓ Quick navigation bar ✓ Includes most functions in shortcuts

Figure 7: Simplified version of the main functionalities of GTA

It was clear then that the main axes of realization of the demonstrator had to focus on a solution integrating:

- Importing a database of tools and functions
- Writing a procedure / test sequence in a structured way
- The ability to save created items for reuse
- Execution of the test by a sequencer
- Communication via an XML-RPC server with tools
- A reading of the returns and measurements on each tool
- The generation of a report readable by a lambda user

2) Selected tool

Once the test needs analysis and the demonstrator specifications were done, I turned to existing software solutions that would make my project possible. I first studied a sequencer from National Instruments, called TestStand; but then I realized that it did not necessarily correspond to our needs, especially due to its complexity.

In the end, after talking about my project around me, I realized that another team at Assystem Technologies was working on a sequencer that already integrated some of our needs. Their software was coded under LabVIEW, so modular and deployable without a license. The C.E.D.M (Design / Studies / Development / Methods) team in question is working on the development of test benches and test facilities under LabVIEW among others. Their software already allowed:

- ✓ Writing a procedure / test sequence in a structured way
- ✓ The ability to save created items for reuse
- ✓ Execution of the test by a sequencer

Although it lacked the integration of an XML-RPC server, making it possible to send commands to each tool, and the generation of a clear report, the software suited our primary needs. The major advantage of this solution is that Assystem Technologies is the owner of this software, and masters its development. Figure 8 summarizes in schematic form the objectives and the development choices described.

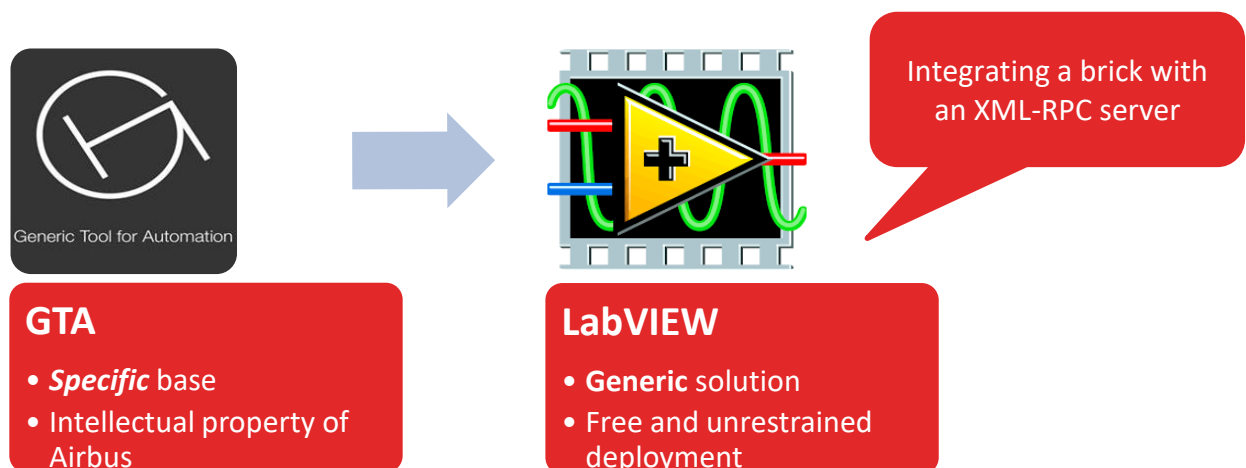


Figure 8: Summary of the choices of development

During my internship I could have support and training of the C.E.D.M. team on their solution, which allowed me to integrate my own needs in their software. It was also necessary to strengthen my knowledge of LabVIEW programming.

The first version of the software I was able to access was an old project for a certain client. In Figure 9, we can see the main menu in question, where each element is then detailed.

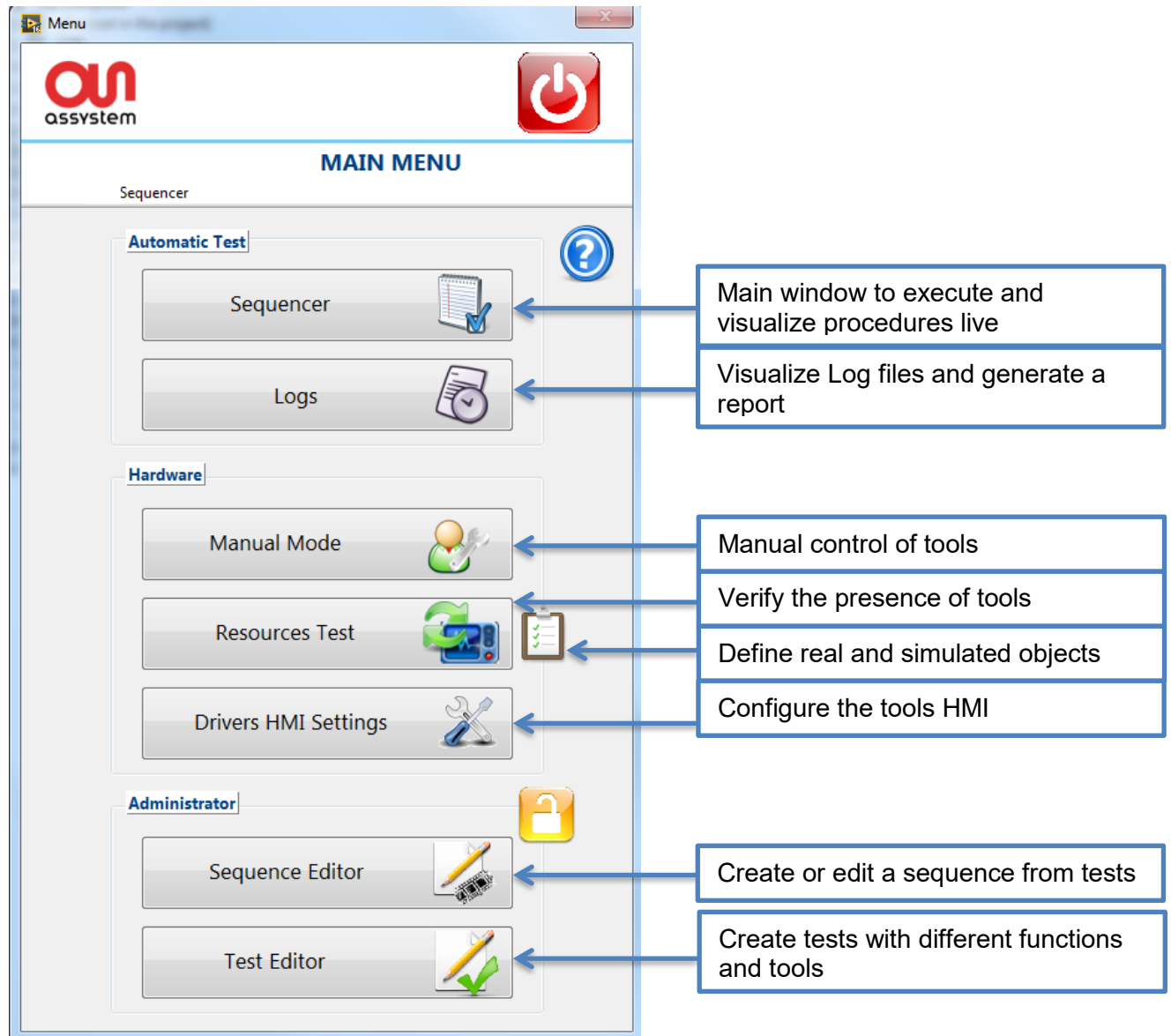


Figure 8: Main menu of the Assystem technologies software

From then on, it was a question of taking in hand the technological bricks which constituted this software, and to integrate a new XML-RPC brick in order to be able to communicate with our tools. Thanks to my table of the functionalities of GTA, I was able to determine the points to work on and those already present.

3) Development and integration of a new brick

a) Structure

The software was fully developed under LabVIEW 2016 initially, and then updated to version 2017. LabVIEW has the advantage of being a platform for designing measurement and control systems based on a National Instruments graphical development environment, relatively easy to handle, and well documented.

The main structural features of the "Assystem Technologies software" are:

- The many subVIs are loaded dynamically at launch, and communicate with each other
- Each brick is defined as a driver of the main program. The name of files and folders can be important to tell the program where to look for relevant VIs
- Data exchange is done through LabVIEW classes (lvclass), or controls (ctl)
- Drivers are considered either as a command or as a measure
- The HMI of each driver (VI) communicates with the main program through an ATP

The general diagram in Figure 10 shows all the drivers present in the software (for now). We will only focus on the XML-RPC brick that is being developed (in the middle).

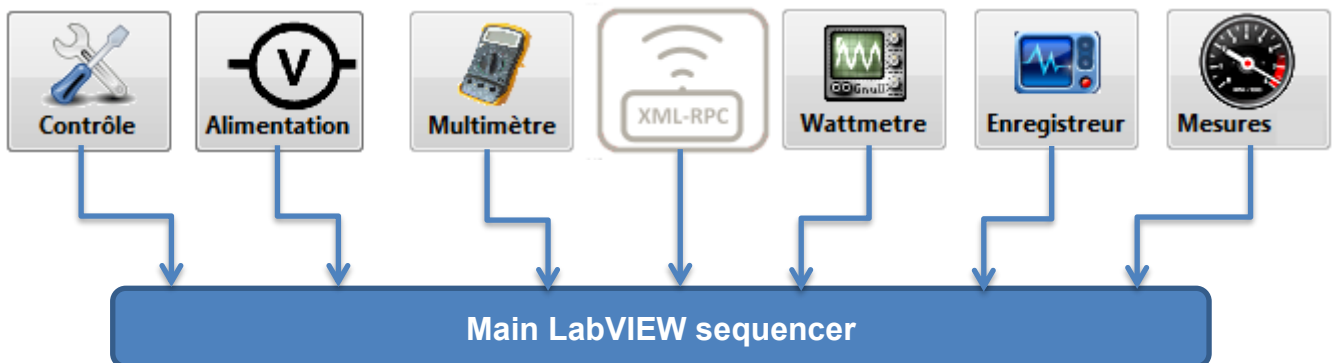


Figure 10: Main structure with the other drivers

As for my XML-RPC brick, once completed, it had the structure visible in Figure 11.

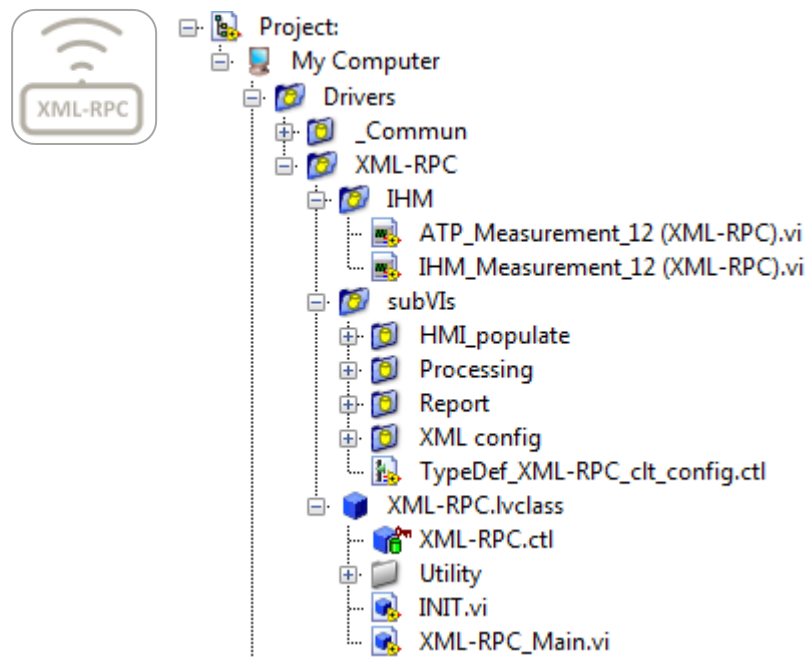


Figure 11: Structure of the XML-RPC brick

The HMI folder contains the two main VIs that allow communication with the main program, and the display of the HMI.


The subVIs folder contains all subVIs that are useful for brick operation, categorized by features.

The last folder contains the information and VIs of the XML-RPC object class.

b) Operations

The operation of each VI is detailed in the XML-RPC brick developer guide (in order to continue the development of the software), of which the summary is presented in **Appendix C**.

Here we will detail the structure of the main VI of the tool, that is to say the "HMI_Measurement_12 (XML-RPC) .vi". Here is a table that summarizes the inputs and outputs of the VI:

	Input	Output
	Configuration	Configuration out
	IHM control?	TAG
	Editor mode?	Value
		Output results

The following figure shows the simplified structure of the VI, which we will then detail.

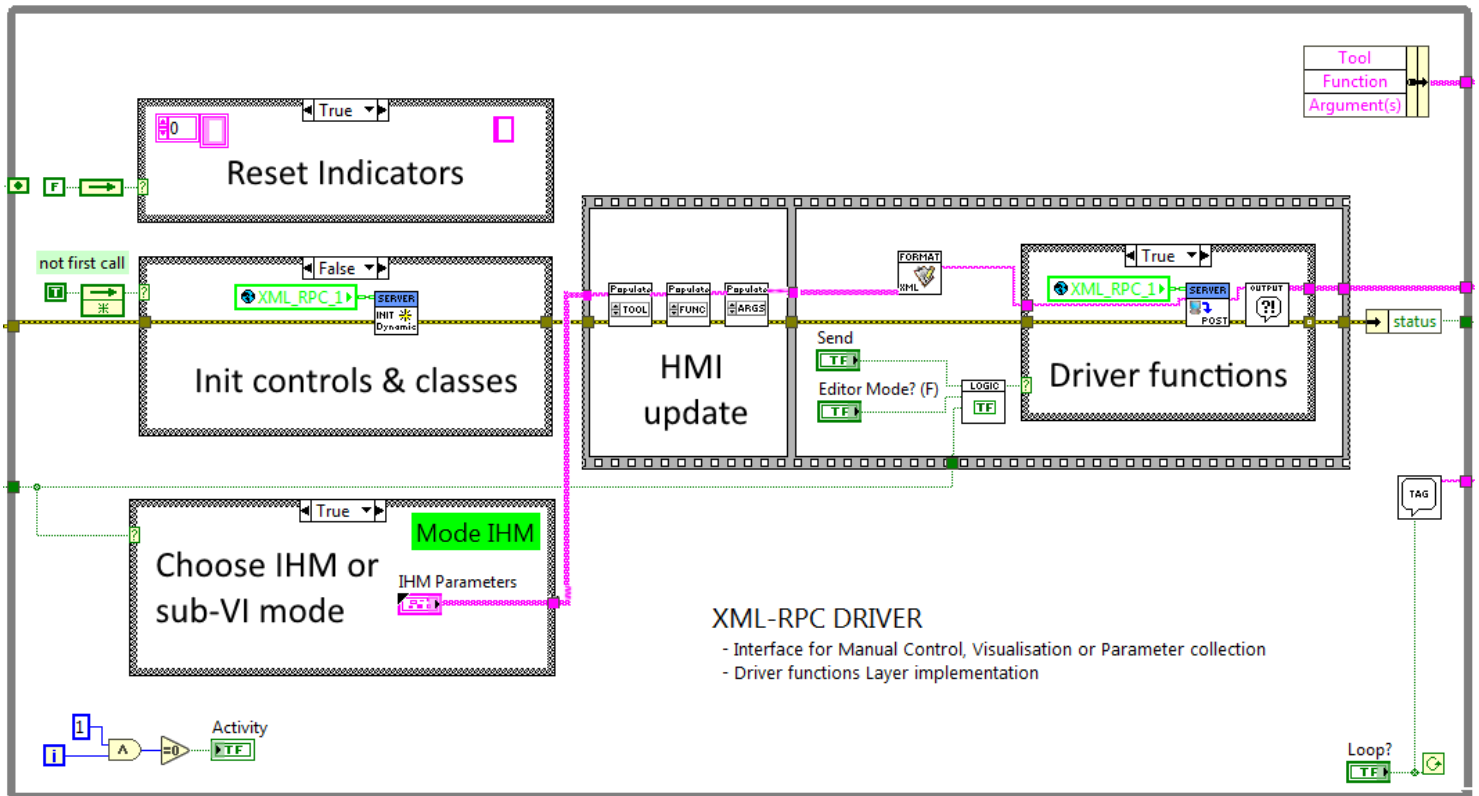


Figure 12: Simplified HMI block diagram

The case structure box at the bottom left controls the information to be sent (Configuration or IHM parameters) according to the selected subVI or HMI mode. HMI mode is used in Manual mode and in Test editor mode, while subVI mode is used when executing sequences. In subVI mode, as can be seen in figure 13, the (previously registered) configuration is sent to the program.

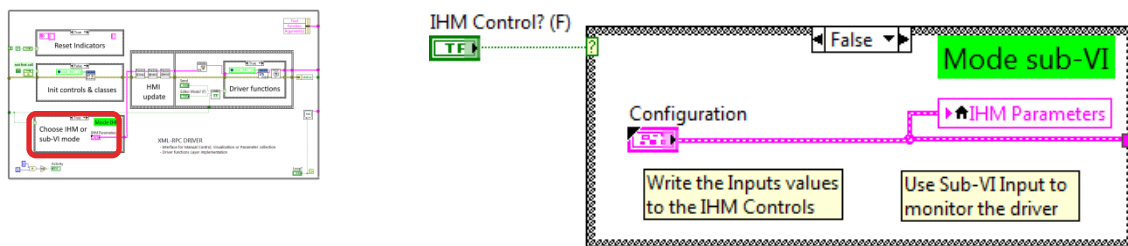


Figure 13: SubVI mode case structure

The case structure in the middle left allows us to initialize the parameters and controls useful to the execution of the VI: it is the INIT step. It is called only once in the first call of the VI. As can be seen in Figure 14, it is composed of four sub-VIs which allow respectively to reset the origin of the window, to activate the controls, and to initialize the XML-RPC class to extract the configuration array.

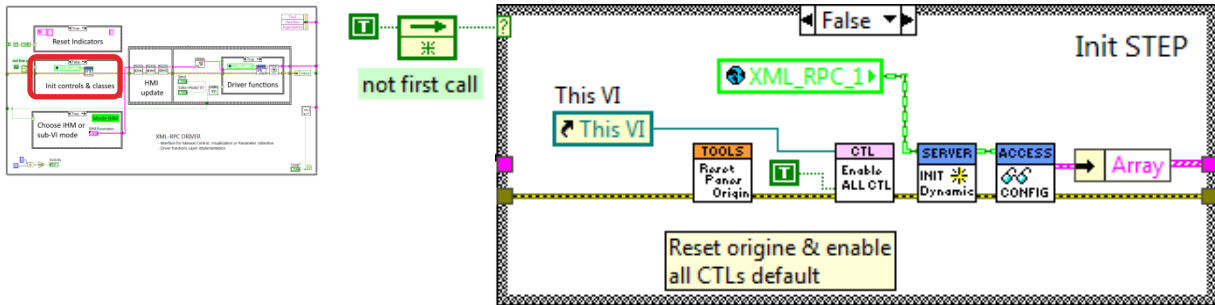


Figure 14: Init step case structure

The structure box at the top left allows us to reset all HMI indicators every time we open the window, except in editor mode, because we might need to reopen the VI to modify the entered values. In Figure 15, we see that the different elements are reset thanks to Property Nodes, when we are not in Editor Mode.

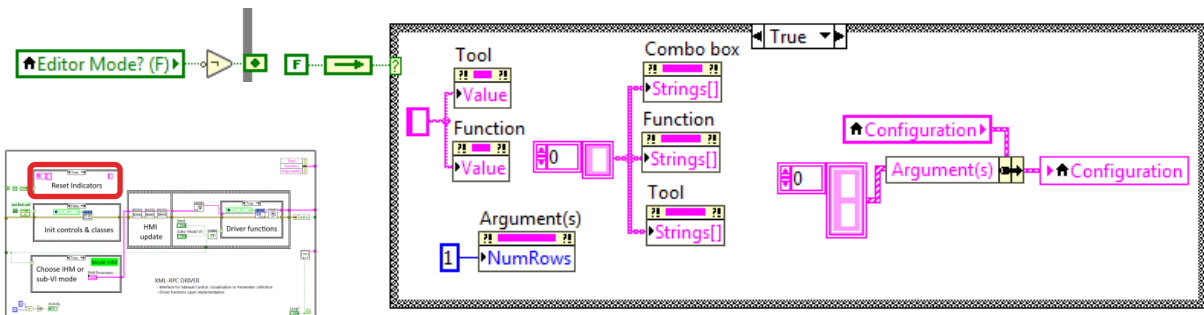


Figure 15: Reset case structure

Then, the left part of the flat sequence is relative to the update of the HMI. It fills the strings and combo box of the indicator "IHM parameters", with the good values it finds in the configuration array, and which are updated dynamically. We can see in figure 16 an extract of this part of the flat sequence.

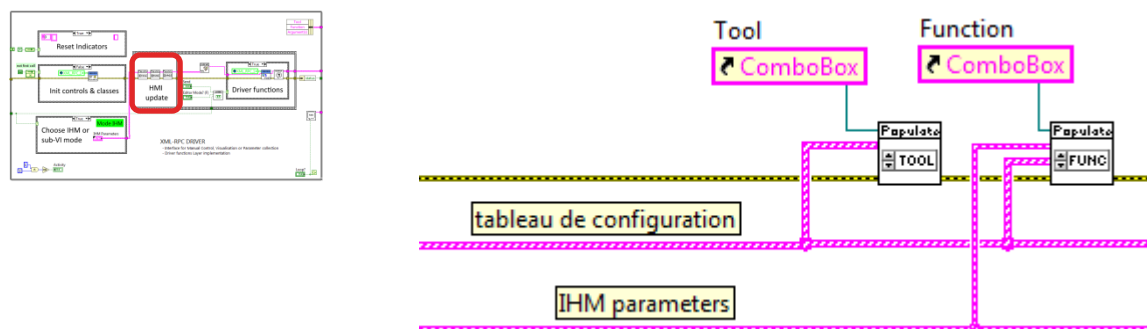


Figure 16: Extract of the HMI update

In Figure 17 we see the details of the previous sub-VI to fill the combo box with existing functions, specific to the chosen tool. When "Tool" changes its value in "IHM parameters", the "Input Array" configuration array is scanned to extract the functions, when the tool corresponds. Then we extract the functions to the correct column (2), and we eliminate the empty boxes. Finally we fill the combo Box "Function" with the Property Node, passing by the argument "Strings []" (possible values of the combo box).

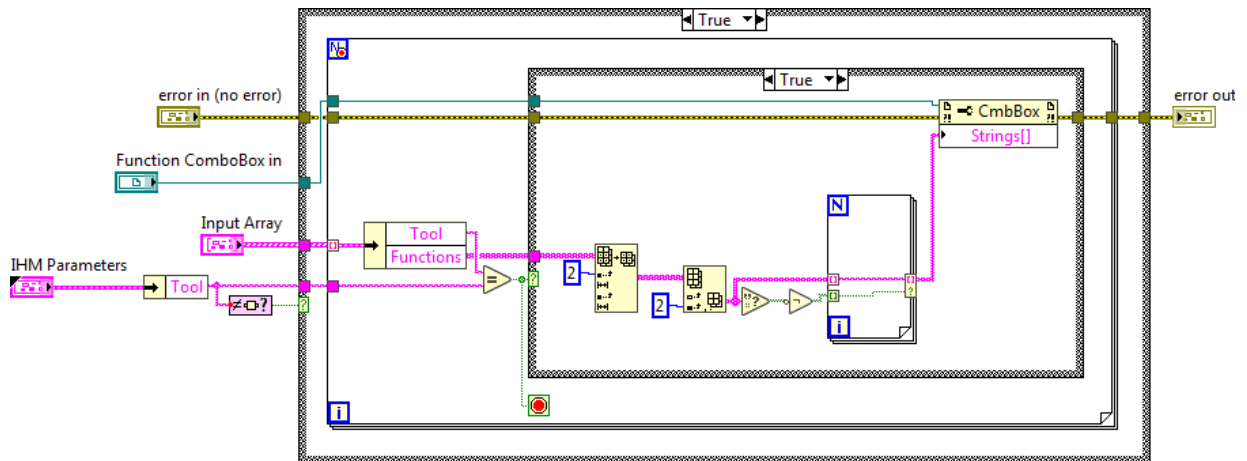


Figure 17: Populate functions subVI

Finally, the right part of the flat sequence allows us to send the data to the tools thanks to the XML-RPC server. This is used when you press Send in manual mode, or in sequencer mode (shown in logic at the bottom of Figure 18). When we are in the right mode, we see that we enter the True structure box, where we send the previously restructured information in XML format, via the "Server post" subVI, and we treat the information received with the "output message filter".

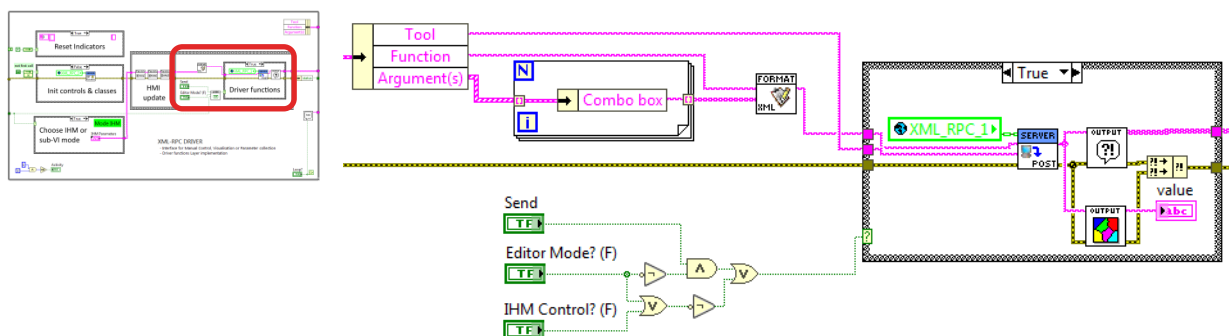


Figure 18: Driver Functions

At that moment, the information entered in the HMI is sent via the XML-RPC server.

c) XML-RPC server

The server is the main element allowing the communication between the software and the tools (of a test bench for example).

XML-RPC is a Remote Procedure Call (RPC) protocol, which allows you to call a function on a remote server from any system (Windows, Mac OS X, GNU / Linux), across a network, and with any programming language.

It allows a client to call functions, with their arguments, on a remote server (designated by a URI), and receive structured data in return. The data is transferred through the HTTP protocol and structured by the XML standard. That's why we use a "format to XML" subVI before sending.

XML-RPC is designed to allow complex data structures to be transmitted, executed, and returned very easily. This is why Airbus mainly uses this technology on its test benches, and we will also use it to control our tools.

This explains the logo on the button of the technological brick, which I realized in flat design:



Figure 19: Logo of the XML-RPC brick

Thus, for each tool that one wants to use, it is enough to assign it an XML-RPC server with the list of functions and arguments that it can execute, as well as the returns.

And on the client side, here the software, just send commands to the right tool. To distinguish them from each other, we use a specific URI (Uniform Resource Identifier), which consists of an IP (example: 127.0.0.1 in localhost) and a port number (example: 8003). See Figure 20 for the server summary.

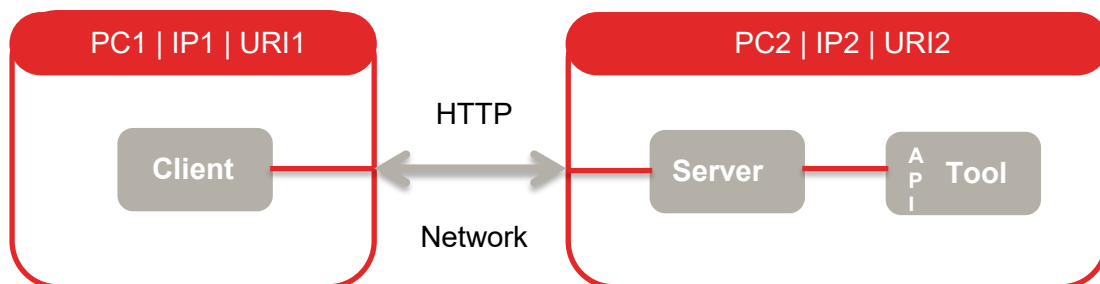


Figure 20: Server operation

d) Configurable

One of the great advantages of the developed brick is the ability to easily add new tools to pilot. By adding a configuration file and 3 lines to an INI file, we can pilot a new tool immediately. This allows a great flexibility and speed of deployment because it is not necessary to recode the software every time you want to add a tool.

First, the INI file is used to define the IP and the port of each tool, as previously described. Figure 21 shows an example of an INI file that allows to control a tool called "demonstrator".

```
[Configuration]

Model = Server XML RPC 1
Simulation = FALSE

[Parameters]

Demonstrator\ToolFileName = Demonstrator
Demonstrator\IP = 127.0.0.1
Demonstrator\Port = 8003
```

Figure 21: Classical initialization file

Then the configuration file in XML format defines the functions, arguments and returns of the tool. **Appendix A** shows an example of a configuration file for the demonstrator tool.

This tool is a LED with a color and a text display.

By reading the XML file, we can see the tool includes:

- A "change_color" function, with the arguments "black; white; red; green; blue"
- A "change_text" function that takes as argument a string
- Four different returns: ID, color, text and result

Since reading a file of this type is not native in LabVIEW, I have developed a brick for reading and saving data in an array. The following figure 22 gives us a global idea of the behavior and links between the developed VIs.

In terms of LabVIEW programming, XML configuration files are read at initialization, and the data is placed in an array.

All you have to do is have the XML configuration file path (automatically defined in the software) as an input, and as an output you have an array containing the data of all the files.

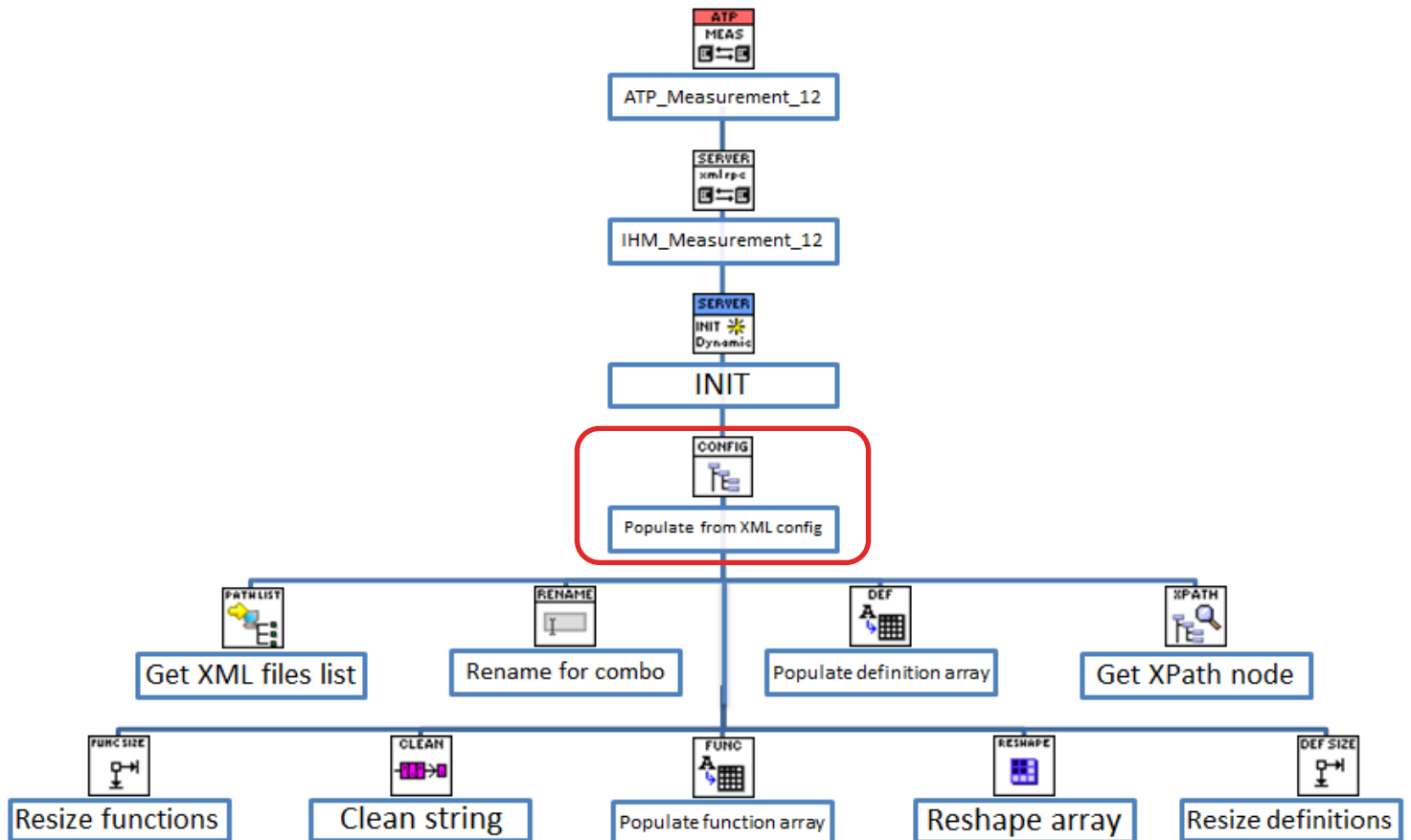


Figure 22: Structure of the VIs which read the configuration files

The main steps of the "Populate from XML config" VI are:

- Initialize arrays and variables, and fill in an array with the path names of each tool
- For each tool / path, initialize the parameters, before filling the sub-arrays with the functions and definitions
- Use Xpath to locate specific portions of data in the file
- Optimize arrays to be as small as possible, without empty rows or columns
- Some arguments are renamed to allow sorting later
- Send the data to the output configuration array

Thanks to this set of VIs, one can have access to the functions and arguments of the tools that one wants to control, by simply putting their configuration file, in XML format, in a specific folder. This is a very simple and generic way, which makes it possible not to have to recode the software for every new tool that one would like to add.

4) Deployment and results

Before being able to deploy the software, it had to be updated to the new version (done post-report) that is based on LabVIEW 2017, and then purified of names and references to old projects for clients, and finally modernize the brick in flat design.

For my XML-RPC brick, Figure 23 shows the pre-deployment HMI enhancement before testing: from LabVIEW 2016 to 2017 in flat design, for a more modern look.

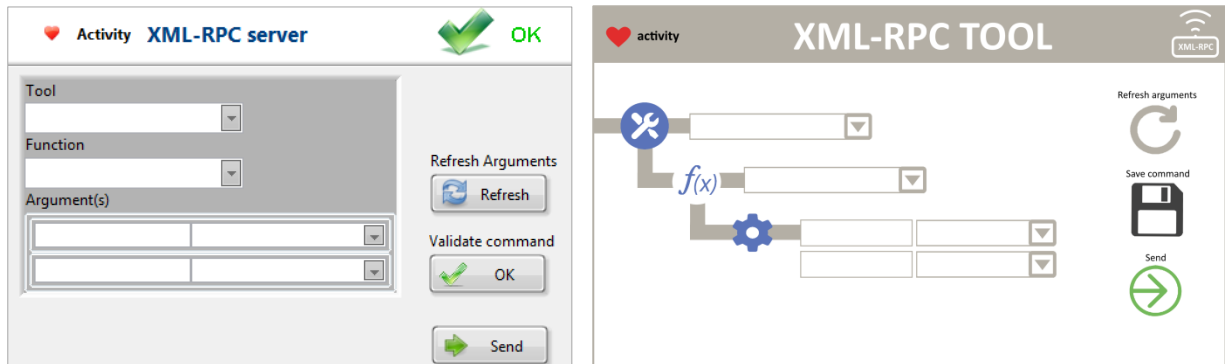


Figure 23: Old (left) and new (right) HMI

As can be seen in the previous figure 23, the XML-RPC brick is very easy to use:

1. We select the tool we want to control
2. We choose the function we want to send
3. Choose the parameters from the list that loads dynamically
4. Save the parameters for the test, or send them directly to the tool

Now, taking a step back, like the table in Figure 7, here is the list of major features of the software deployed:

Element concerned	Functionalities
Sequencer	<ul style="list-style-type: none"> ✓ Execution of a sequence ✓ Play, Pause, Stop button ✓ Detailed visualization of the execution for each step ✓ Shortcuts to some drivers ✓ Open the report
Logs	<ul style="list-style-type: none"> ✓ View log in "hist" format
Manual Mode	<ul style="list-style-type: none"> ✓ Quick access banner to certain drivers ✓ Manually control the tools
Resource Test	<ul style="list-style-type: none"> ✓ Check the correct operation of the drivers ✓ Indicate whether or not a device is simulated
Drivers HMI Settings	<ul style="list-style-type: none"> ✓ Restore the parameters of an HMI
Sequence Editor	<ul style="list-style-type: none"> ✓ Creating a sequence from the list of tests that exist ✓ Adding information about the sequence
Test Editor	<ul style="list-style-type: none"> ✓ Editing a test ✓ Possibility to add: <ul style="list-style-type: none"> • a description • an order • a measure • a wait • an audit • a label • a data storage / reading • a goto

Figure 21: Simplified list of the main functionalities of the Assystem Technologies sequencer

In order to test the good behavior of my software and the XML-RPC communication, I used at the beginning a LED tool presented in **III.3.d**. It was developed by a former trainee, and could simply change color and text.

I then decided to develop my own test tool in order to try my software and to be able to show the operation and the results in a visual way. So I choose to create a "game" developed with Pygame on Python.

The game's interface, visible in Figure 25, is a white grid of 20 pixels by 20 pixels. The idea is to be able to send commands to the interface, via the XML-RPC server, to be able to draw on it. My first example of drawing made with the sequencer is Mario:

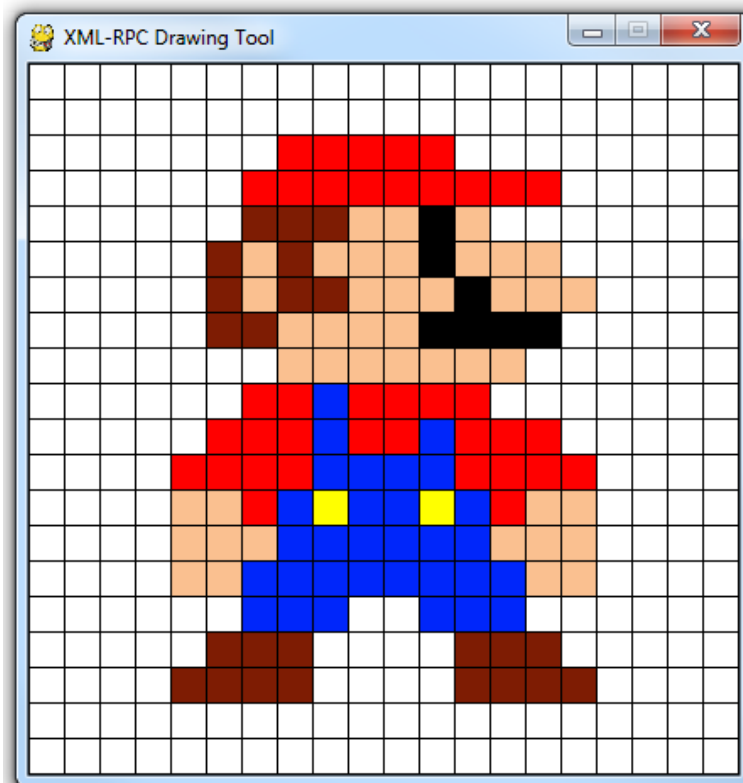



Figure 25: Example of a drawing with the Pixel Draw tool

This tool, developed with the Pygame module, has an XML-RPC server, whose client is the LabVIEW software. This server hosts a list of functions, detailed in the following table, that it executes when it receives the command:

	Function	Effect
	Clear (clear)	Reinitializes the grid to blank
	Draw (column, row, color)	Colors a pixel at the given position (column, row), and at the given color (chosen from a predefined list)
	Stop (stop)	Stops and closes the game

Thanks to my LabVIEW software, I can create a sequence composed of many commands, which allow to draw something, and to check if each pixel is well displayed. The following figure 26 gives an idea of part of the test I did to draw Mario.

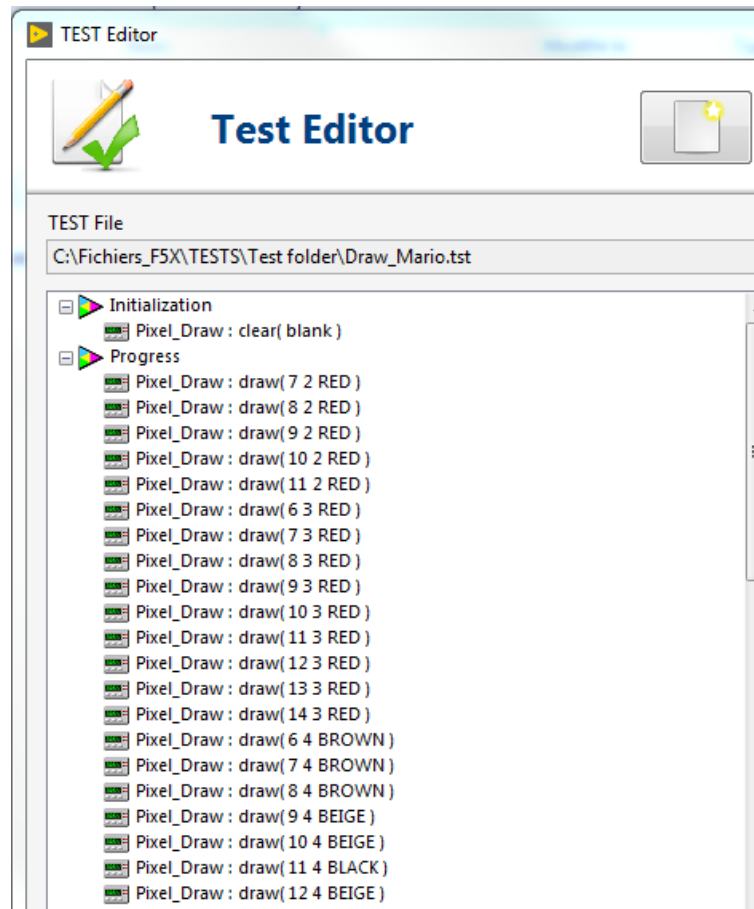


Figure 26: Example of a sequence for the Pixel draw tool

Following the execution of this test thanks to the sequencer, we obtained the result in figure 25, and a report, automatically generated, visible in the following figure 27:

Test	Action	Comment	Status
Initialization			OK
	Sequence Initialization	Sequence Initialization OK	OK
Draw_Mario			OK
	Pixel_Draw : clear(blank)	Server Return : Column= NONE ; Row= NONE ; Color= NONE; Result= OK	OK
	Pixel_Draw : draw(7 2 RED)	Server Return : Column= 7 ; Row= 2 ; Color= RED ; Result= OK ;	OK
	Pixel_Draw : draw(8 2 RED)	Server Return : Column= 8 ; Row= 2 ; Color= RED ; Result= OK ;	OK
	Pixel_Draw : draw(9 2 RED)	Server Return : Column= 9 ; Row= 2 ; Color= RED ; Result= OK ;	OK
	Pixel_Draw : draw(10 2 RED)	Server Return : Column= 10 ; Row= 2 ; Color= RED ; Result= OK ;	OK
	Pixel_Draw : draw(11 2 RED)	Server Return : Column= 11 ; Row= 2 ; Color= RED ; Result= OK ;	OK
	Pixel_Draw : draw(6 3 RED)	Server Return : Column= 6 ; Row= 3 ; Color= RED ; Result= OK ;	OK
	Pixel_Draw : draw(7 3 RED)	Server Return : Column= 7 ; Row= 3 ; Color= RED ; Result= OK ;	OK
	Pixel_Draw : draw(8 3 RED)	Server Return : Column= 8 ; Row= 3 ; Color= RED ; Result= OK ;	OK

Figure 27: Generated report from the previous sequence

The report is the key element for analyzing the results. It testifies the good behavior or not of the procedure, thanks to the global and individual statutes. It is also possible to check the status of each step live, during the execution of a sequence.

The report takes each step of the written sequence, and indicates the return of the server for the command sent, as well as the status. A sequence can be composed of several tests, and for each test we note its global status according to the OK or KO of each step in the test.

The user notices here that the sequence was successful (OK).

5) Challenges and difficulties

Before I started developing an XML-RPC brick, I had to identify the best software for the task. Initially, my internship was based on the development of a brick under TestStand. This software is a configurable and modular test sequencer, distributed by National Instruments since 1999.

Its recognition and the fact that it has enough online documentation makes TestStand an attractive software. I then spent about 2 weeks getting to know TestStand through many tutorials, from beginner to advanced, and then another two weeks trying to develop the XML-RPC brick. The GANTT chart in **Appendix B** details this planning.

Although the features of the sequencer were advanced and optimized, TestStand did not really suit our needs. The biggest negative points that emerged are:

- Software complexity hindering development
- The fact that the writing of a procedure is not very intuitive and is not made to be modified often
- To deploy the software it is necessary that the "client" also obtains a relatively expensive TestStand License

On the plus side, TestStand is very efficient at running sequences, even in parallel, and for writing very complete reports.

After taking all these parameters into account, and following a discussion with the team of Assystem Technologies who developed the home sequencer, it was decided to work with them to integrate an XML-RPC brick. The big advantages of this solution are:

- Support and simplicity of software and LabVIEW
- The fact that the software is owned by Assystem Technologies
- Simplicity and cost-free software deployment, coded in LabVIEW

Even if the software is not complete in terms of functionality, it is easy enough to add some, and allows a good basis for a demonstrator.

IV) Complementary activities

During my internship I was able to participate in several activities and trainings that allowed me to develop my knowledge and be aware of the professions that surround me. **Appendix B** gives all the details under a GANTT chart.

1) Validation of GTA

A major role of the automatic test team is the validation of the GTA software, so that the developers of Airbus India have a return on the development of their software. It is a question of making sure of the good behavior of the new functionalities and of the non-regression of the old functionalities of GTA. Validation is somehow part of the debug / test of the software, in which I was able to take part. Each point to be tested is listed, and distributed among the members of the team, who verify and attest to the good functioning (OK) or not (KO) of the points to be tested.

During my internship, I participated in the validation of the v22 version of GTA, delivered in May. I detected minor HMI issues in the menus, as well as more important issues related to new "ignore" and "undo / redo" functions that were not fully developed. The problems were present in conditional if or while loops. I then described my observations to the team and a member of Airbus India so that he could correct the software before delivery to the customer on the test benches.

These validation steps allowed me to better understand the job of tester, and to deepen my knowledge of automatic tests, in parallel with my training at GTA upon my arrival.

2) Training

During my experience, I benefited from several trainings, allowing me to better understand the professional world in which I worked, and the roles of innovation and quality in the engineering profession.

My first training was an introduction to the **cockpit and piloting on a flight simulator**. The simulator in question, visible in figure 28 was made by an Assystem Technologies team, mainly allowing training in flight. In fact, the testers are required to perform flight maneuvers on the Airbus test equipment, whose sessions are sometimes expensive and difficult to obtain. The Assystem Technologies flight simulator thus makes it possible to train to maneuvers of various aircraft (including A320 and A380), ensuring full efficiency of testers during test slots on the customer site. As part of my team, it helps to prepare for the writing of procedures for the tests to be performed. For my part, I learned about all the elements that constitute a cockpit, as well as their usefulness, then I took part in an actual practice on the simulator, where I was able to practice the take-off and landing of an A380, following specific instructions. I will remember the complexity of a cockpit in terms of instruments and buttons, but the simplicity in terms of maneuvers.



Figure 28 : Home-made flight simulator

Then, I had the opportunity to participate in a training called "**Successful innovation project**", presented by my tutor. It persuaded me that innovation is a major issue in the engineering profession, where we are all actors, and especially that it exists in different forms. I realized that today, innovation is not an option, but a necessity to be and above all to remain competitive on projects. It is pursued by everyone, so it is important to always keep a head start to avoid being swallowed by competitors. Also, market analysis and communication are major steps to promote and realize your project.

Finally, during my last "**Quality awareness 2018**" training we discussed the evolutions EN9100: 2016 / ISO 9001: 2015, and we made a few reminders on the processes of Sales and Realization.

3) Writing of documentation

A very important part when developing a program is the documentation. It allows new users to take ownership of the software to understand how to develop it, or just how to use it.

So I realized 3 different documentations:

- How to install the software? - Installation guide
- How to use the software? - User guide
- How to modify and develop the software? -Developer guide

Appendix D is the Developer Guide for the XML-RPC Brick, that I wrote, which details the structure and operation of each VI in the tool, as well as instructions for further development.

I also created a poster that sums up my whole thesis, visible in **Appendix C**, which was used to present my internship to my French engineering school.

V) Conclusion

These 6 months of thesis spent in the "automatic test" team of Assystem Technologies in Toulouse allowed me to carry out my project. My mission was to develop a generic demonstrator of automatic tests, in order to deviate from the specific solutions used, of which Assystem Technologies does not hold the intellectual property.

The software developed with the help of another team of the company, meets the objectives and needs defined. It is coded under LabVIEW, and thus allows a license-free deployment, without any intellectual property rights constraints.

The XML-RPC brick allows remote communication with any tool that has a server. And adding new tools to the software is very easy: just add a configuration file in XML format.

For example, I was able to create and configure my own tool, developed with Python.

In terms of technical enrichment, I was able to acquire a lot of knowledge in LabVIEW and Python programming, but also on the operation and the usefulness of a sequencer in the aeronautics world for example. It also allowed me to learn about XML and how an XML-RPC server works.

I also conducted training that allowed me to better understand the professional world around me, and the innovative aspects of the engineering profession, but also the job of tester.

From a personal point of view, this internship allowed me to gain autonomy and technical expertise, since I was working independently on a new project. I also developed my decision making and my teamwork. And above all, I was able to enrich my technological curiosity by working in a professional environment surrounded by interesting people, who brought me new ideas.

This professional experience led me to use and enhance the knowledge acquired during my years of training in engineering school. Assystem Technologies allowed me to assume the duties and responsibilities of a mechatronics engineer.

Appendixes

A. Xml configuration file

```
<?xml version="1.0"?>
<GENERIC_TOOL name="Demonstrator" toolDisplayName="LED Demonstrator">
  <HMI hasActionOnFail="0" hasTimeout="1" hasComment="1" hasDumpList="0"/>
  <functions type="STATIC">
    <function name="change_color" toolId="Demonstrator" argList=""
      functionDisplayName="Change the LED color" toolType="Tool">
      <arguments>
        <argument name="color" type="string"
          HMILabel="color" mandatory="yes" searchType=""
          defaultValue="" cond=""
          values="black;white;red;green;blue" constValue=""/>
      </arguments>
      <returns>
        <return name="ResultStruct" type="struct"/>
      </returns>
    </function>
    <function name="change_text" toolId="Demonstrator" argList=""
      functionDisplayName="Change text" toolType="Tool">
      <arguments>
        <argument name="text" type="string" HMILabel="Text"
          mandatory="yes" searchType="" defaultValue="" cond=""
          values="" constValue=""/>
      </arguments>
      <returns>
        <return name="ResultStruct" type="struct"/>
      </returns>
    </function>
  </functions>
  <definitions>
    <data name="ResultStruct" type="struct">
      <attributes>
        <attribute name="ID" type="int" trueCond="" returnCode="no"
          occurence="" waitUntil="no" falseCond=""/>
        <attribute name="color" type="string" trueCond=""
          returnCode="no" occurence="" waitUntil="no" falseCond=""/>
        <attribute name="text" type="string" trueCond=""
          returnCode="no" occurence="" waitUntil="no" falseCond=""/>
        <attribute name="result" type="string" trueCond="OK"
          returnCode="true" occurence="1" waitUntil="yes"
          falseCond="KO"/>
      </attributes>
    </data>
  </definitions>
</GENERIC_TOOL>
```


B. Gantt diagram

Here I present the Gantt diagram of the implementation activities of the software. It sums up the main tasks I planned and worked on during my thesis. As we can see, I had many meetings with different people to acquire new knowledge, and to explain my project. I also spent most of my time on the development of the XML-RPC brick and its HMI.

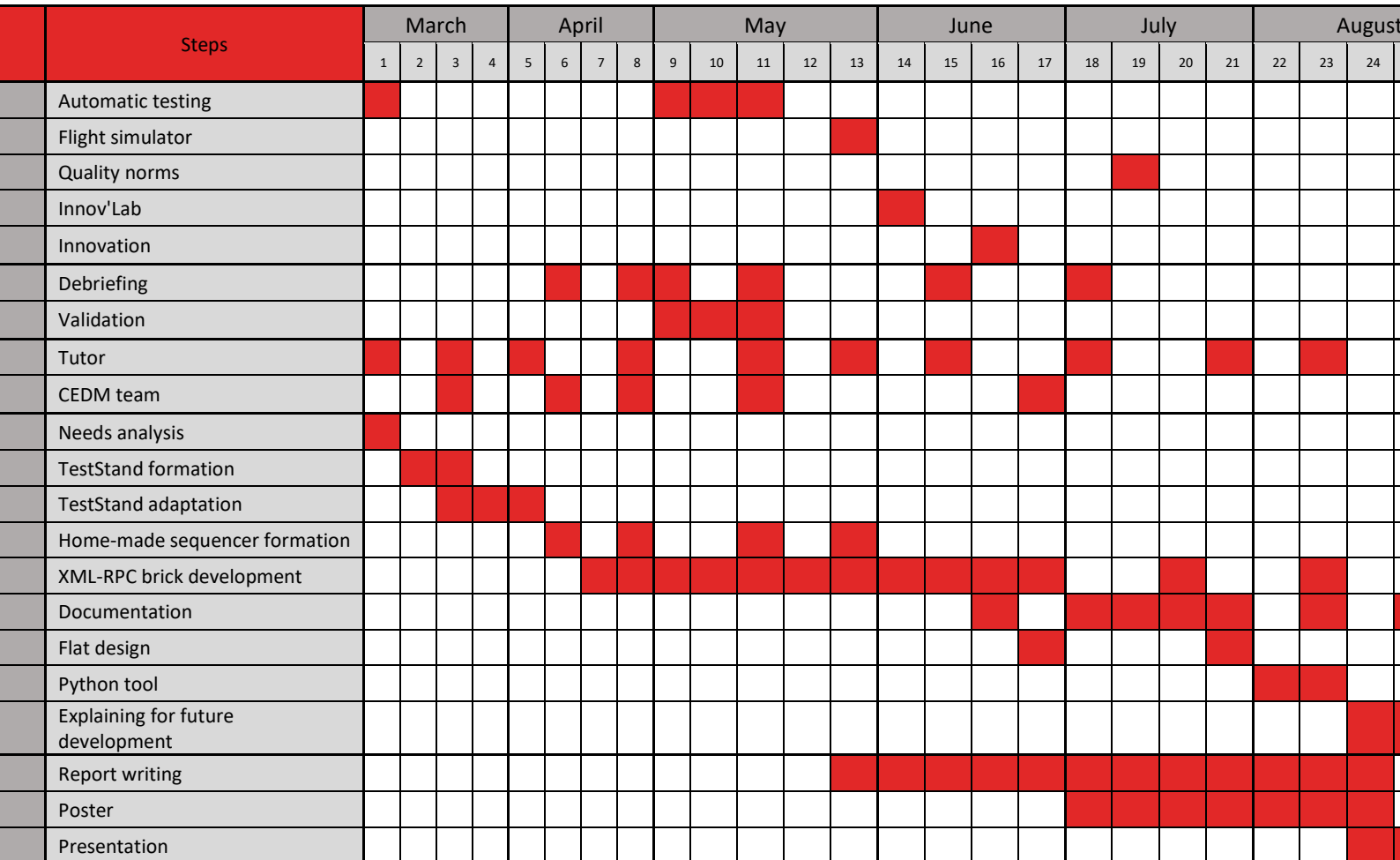
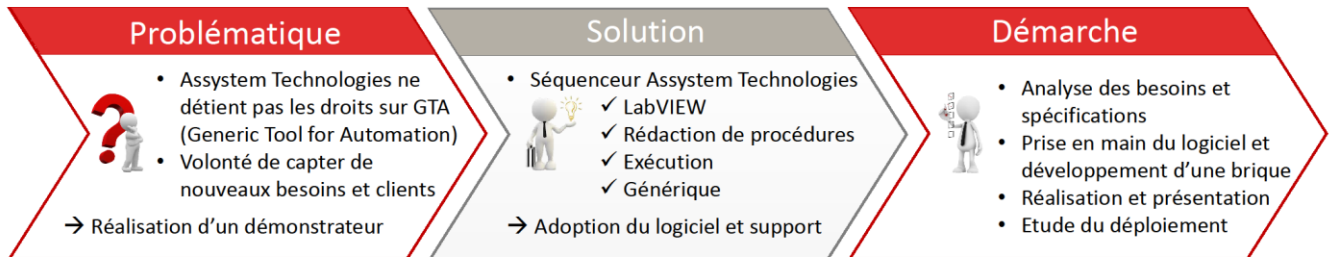


Figure 29: GANTT diagram

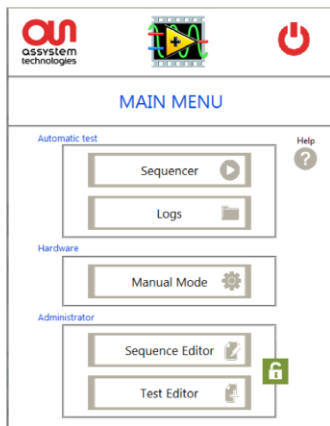
C. Poster

The following poster I designed sums up my whole thesis and was used to present my internship to my French engineering school. I presented it in front of two professors and two people from my company, as well as random students who were interested in my project. Everybody was very happy about the job I had accomplished.

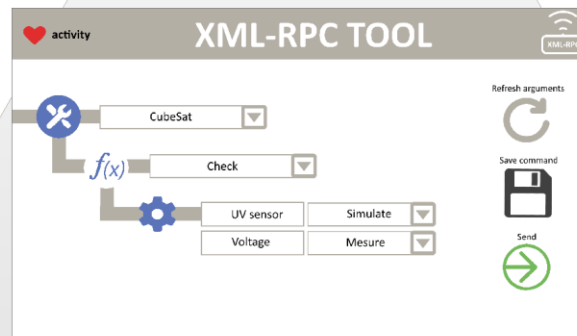
Mots-clefs : démonstrateur, séquenceur, serveur XML-RPC, LabVIEW, configurable, générique



1. Rédaction de la procédure



2. Exécution de la séquence



IHM de la brique développée, elle permet :

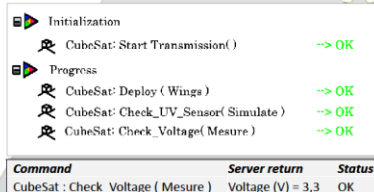
- ❖ D'ajouter des fonctions à la procédure
- ❖ D'envoyer des commandes via le serveur
- ❖ De recevoir un feedback après chaque request

❖ Banc d'essai CubeSat



request
feedback

3. Génération d'un rapport



Vérification

Pour chaque étape du test, détails :

- ✓ Commande
- ✓ Feedback
- ✓ Statut (OK-KO)

Serveur XML-RPC

- Appel de fonctions sur un serveur à distance avec le protocole RPC (Remote Procedure Call)
- Données structurées en XML et transférées via HTTP
- Structure Serveur/Client avec identification spécifique

Points forts

Configurable

- Fichier .INI : contient l'adresse du serveur sous forme d'une URI (Uniform Resource Identifier)
 - Fichiers .XML : liste les fonctions, arguments et retours de l'outil
- Logiciel générique et facilement modulable

Déploiement

- Facile : installation d'un exécutable
 - Gratuit : pas de licence LabVIEW nécessaire
 - Propriétaire : appartenance des droit de propriété intellectuelle
- Détachement de la solution Airbus

Assystem Technologies
13 rue Marie Louise Dissard
31024 Toulouse
05 34 39 60 00
<https://www.assystemtechnologies.com>




D. XML-RPC Developer guide


The Developer Guide for the XML-RPC Brick details the structure and operation of each VI in the tool, as well as instructions for further development.

I wrote it so that people can continue to work on the project with a detailed explanation of what has already been implemented.

2018

XML-RPC Tool Developer Guide

 Activity

XML-RPC Tool 

Tool

▼


Function

▼


Argument(s)

<input type="text"/>	<input type="text"/>	▼
<input type="text"/>	<input type="text"/>	▼
<input type="text"/>	<input type="text"/>	▼
<input type="text"/>	<input type="text"/>	▼


Refresh Arguments



Save command



Send



COUTURIER Joshua
Assystem France
31/08/2018

Table of content

Introduction	1
1. Structure.....	1
2. Configuration	1
2.1. .INI file.....	1
2.2. XML files	1
2.3. Populate from XML configuration files.....	4
2.3.1. Main VI	4
2.3.2. Get XML files list (SubVI).vi.....	5
2.3.3. Get XPath node (SubVI).vi.....	5
2.3.4. Clean string (SubVI).vi	6
2.3.5. Populate definition array (SubVI).vi	6
2.3.6. Populate function array (SubVI).vi	6
2.3.7. Reshape array (SubVI).vi	7
2.3.8. Resize functions (SubVI).vi	7
2.3.9. Resize definitions (SubVI).vi	7
2.3.10. Rename for combo (SubVI).vi	7
3. XML-RPC class.....	7
3.1. XML-RPC.ctf	7
3.2. INIT.vi	8
3.3. XML-RPC_main.vi	8
3.4. Read config cluster.vi.....	8
4. Tool setup.....	8
4.1. Global drivers	8
4.2. Manual mode drivers.....	9
5. HMI	9
5.1. TypeDef_XML-RPC_clt_config.ctf.....	9
5.2. ATP_Measurement_12.vi.....	10
5.3. IHM_Measurement_12.vi.....	10
5.3.1 HMI interface	10
5.3.2 HMI modes	11
5.3.3 Block diagram template.....	12
5.3.4 INIT step.....	13
5.3.4.1 TOOLS_Reset_All_Panes_Origin.vi.....	13
5.3.4.2 CTL_Enable_ALL_VI_CTLs.vi.....	13
5.3.5 HMI update	13
5.3.5.1 Populate_tools (SubVI).vi	13
5.3.5.2 Populate_functions (SubVI).vi.....	13

5.3.5.3	Populate_Argument(s) (SubVI).vi.....	14
5.3.5.4	Coordinates to Index.....	14
5.3.5.5	Populate_comboBoxes (SubVI).vi.....	14
5.3.6	Driver functions	14
5.3.6.1	Get type (SubVI).vi.....	14
5.3.6.2	Format to XML (SubVI).vi.....	15
5.3.6.3	Multi-output message filter (SubVI).vi.....	15
5.3.6.4	Values_for_tag (SubVI).vi.....	15
6	Report generation	16
6.1	Report_populate_LOG_array(SubVI).vi.....	17
6.2	Report_add_test_line(SubVI).vi	18
6.3	Report_Test_Status(SubVI).vi.....	18
6.4	Report_Create_XLS(SubVI).vi	18
7	More	18

Introduction

The XML-RPC Tool is a **driver** added to the main sequencer software. Combined, they allow us to **write** procedures for specific XML-RPC based tools, **execute** them, and **generate** a report to be read by the user. The software allows to send commands specific to each tool (defined in XML configuration files), which include functions and arguments. The software communicates with the tools via an **XML-RPC server**, for a given IP and port (defined in the initialization file).

1. Structure

To help situate the developer, the figure 1 presents the tree views of the XML-RPC tool folders and files. The main paths for working on the XML-RPC tool are:

- ➔ C:\WORK\Séquenceurdémo\MAIN_SW_SPECIFIC\Drivers\XML-RPC **for the main VIs**
- ➔ C:\Fichiers_Demo_sequenceur_LV2017\CONF_Drivers\XML_RPC **for the config files**
- ➔ C:\Fichiers_Demo_sequenceur_LV2017\Admin\XLS LOGs **for the generated report files**

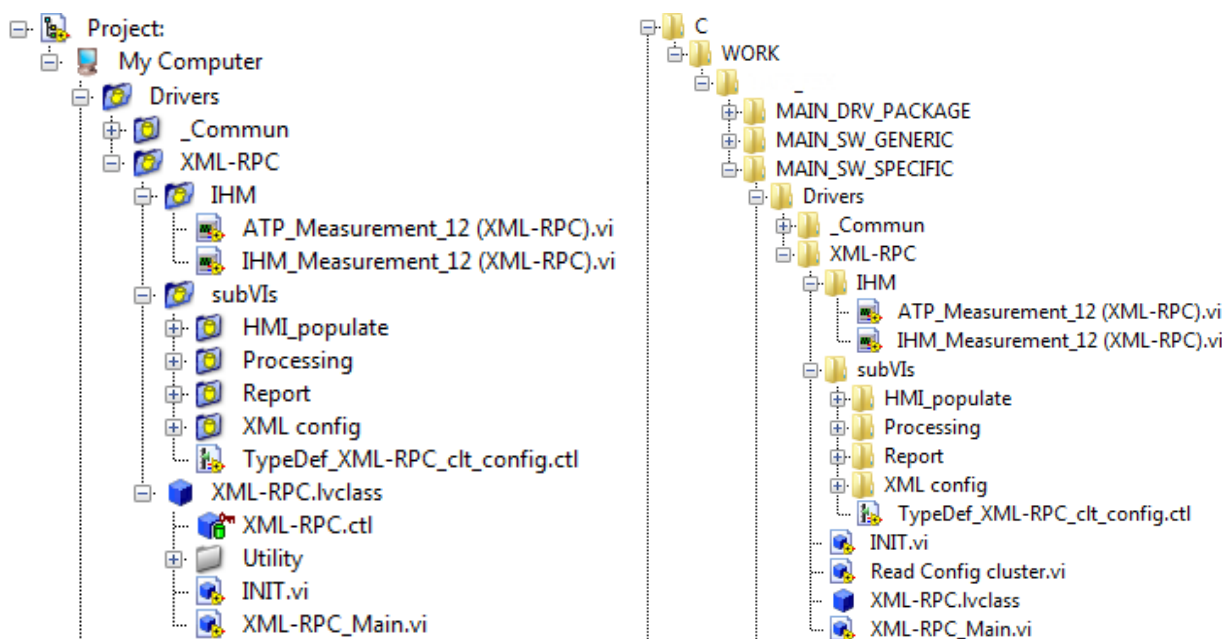


Figure 1: Structure of the project under LabVIEW and Windows

The main file/VI is **IHM_Measurement_12(XML-RPC).vi**. Most of the other VIs are subVIs of this one, where some of them are placed in the folder named as the specific function they serve.

2. Configuration

Configuration files are placed in “\CONF_Drivers\XML_RPC_1”. They include an **INI file** to define server properties, and **XML files** specific to each tool to be used.

2.1. .INI file

The INI file path is automatically read by the software. For it to be read correctly, [configuration] and [Parameters] have to be present for structure. This file is read by the INIT VI from the XML-RPC class.

For each new tool, we have to add 3 lines to the INI file:

- Tool\ToolFileName = ... the name of the tool as displayed in the name of the related xml file
- Tool\IP = ... the IP of the server
- Tool\Port = ... the port of the server

The figure 2 gives us the basic example of the XML-RPC initialization file.

```
[Configuration]

Model = Server XML RPC 1
Simulation = FALSE

[Parameters]

Demonstrator\ToolFileName = Demonstrator
Demonstrator\IP = 127.0.0.1
Demonstrator\Port = 8003
```

Figure 2: Typical initialization file

The important information is the **IP** and **port** numbers, which give the address of the server of each tool.

2.2. XML files

The XML files define every possible function and values understood by a tool. They allow a **generic initialization** of every tool in a configuration file, without having to modify the software code for every new tool added. The power of these files relies on the **simplicity** to access information, knowing the relevant **XML tag**.

The important tags, visible in figure 3, present in every XML for every tool, are:

- The main « GENERIC_TOOL » one,
- « HMI » which allows to define parameters specific to the conditions of the test (timeout, action on fail, comment, dump list),
- « functions » which englobes the definitions of all the tools functions,
- N tags « function » where N is the number of functions. Inside of this one, we have a tag « arguments » which lists the arguments of every function, and a tag « returns » which describes the parameters sent back by the function,
- « definitions » describing the returns and transitions,
- X tags « data » where X is the number of returns. Inside, we find a tag « attributes » giving the details about the parameters of the return structure and its transitions.


```

<GENERIC_TOOL>
  <HMI>
    ...
  </HMI>
  <functions>
    <function>
      <arguments>
        ...
      </arguments>
      <returns>
        ...
      </returns>
    </function>
  </functions>
  <definitions>
    <data>
      <attributes>
        <attribute/>
        ...
        <attribute/>
      </attributes>
    </data>
  </definitions>
</GENERIC_TOOL>

```

Figure 3: XML configuration file skeleton

Of course it is necessary to add description tags and values inside every previous tag. The following tables give more details (*behavior in GTA*).

<GENERIC_TOOL>

Parameters	Values
name	Name of the tool used for the command
toolDisplayName	Name of the tool used for the HMI

<HMI>

Parameters	Values
hasActionOnFail	"1" or "0". If "1" displays a zone allowing to choose what happens after the action fails (« continue » or « stop »)
hasTimeout	"1" or "0". If "1" displays a zone allowing to set a timeout
hasComment	"1" or "0". If "1" displays a zone allowing to add a comment
hasDumpList	"1" or "0". If "1" adds the parameter to the « Dump list ». This list allows to print the values of all the parameters contained, at the end of the test

<function>

Parameters	Values
name	Name of the function
functionDisplayName	Name of the function to display in HMI
toolId	ID used by scheduler.
argList	List of the names of the arguments wished to be displayed in HMI. If nothing is detailed in « argList », they are all displayed

<argument>

Parameters	Values
name	Name of the argument
HMILabel	Name of the argument to display in HMI
type	Type of data taken by the argument, namely : <ul style="list-style-type: none">• string• int• float• boolean
searchType	Name of the database in which a search has to be done. In the HMI, a new window opens when we want to give a value to the argument
defaultValue	Default value we give
values	List of the values that the argument can take. In the HMI, these values are displayed in a drop down list
constValue	Constant value to be assigned
cond	Name of the parameter on which the condition depends. The values linked to this condition have to be inserted in the « values » option
mandatory	« yes » or « no ». If « no », the user doesn't have to give a value to the argument. By default it is necessary to fill in all the arguments

<return>

Parameters	Values
name	Name of the variable in which we wish to stock the return of the function
type	Type of return. Often of type « struct »

<attribute>

Parameters	Values
name	Name of the attribute where we stock the return value of the function
type	Type of the attribute, namely : string int float boolean
trueCond	Value to which we want to compare the attribute to during the transition. For example, if it is « OK », we will go to the next step if the attribute's value is also « OK ».
returnCode	« true » or « false ». If « true », the attribute's value will be used for testing the transition

There are many examples which can be used as templates in the config file path.



2.3. Populate from XML configuration files

This part details the “Populate from XML config.vi”. It allows extracting the relevant information for all the tools, to a big array. At this stage, everything is imported, but it is not all used later on.

The next figure shows the hierarchy of the VIs and SubVIs, around the “*populate from XML config*” VI.

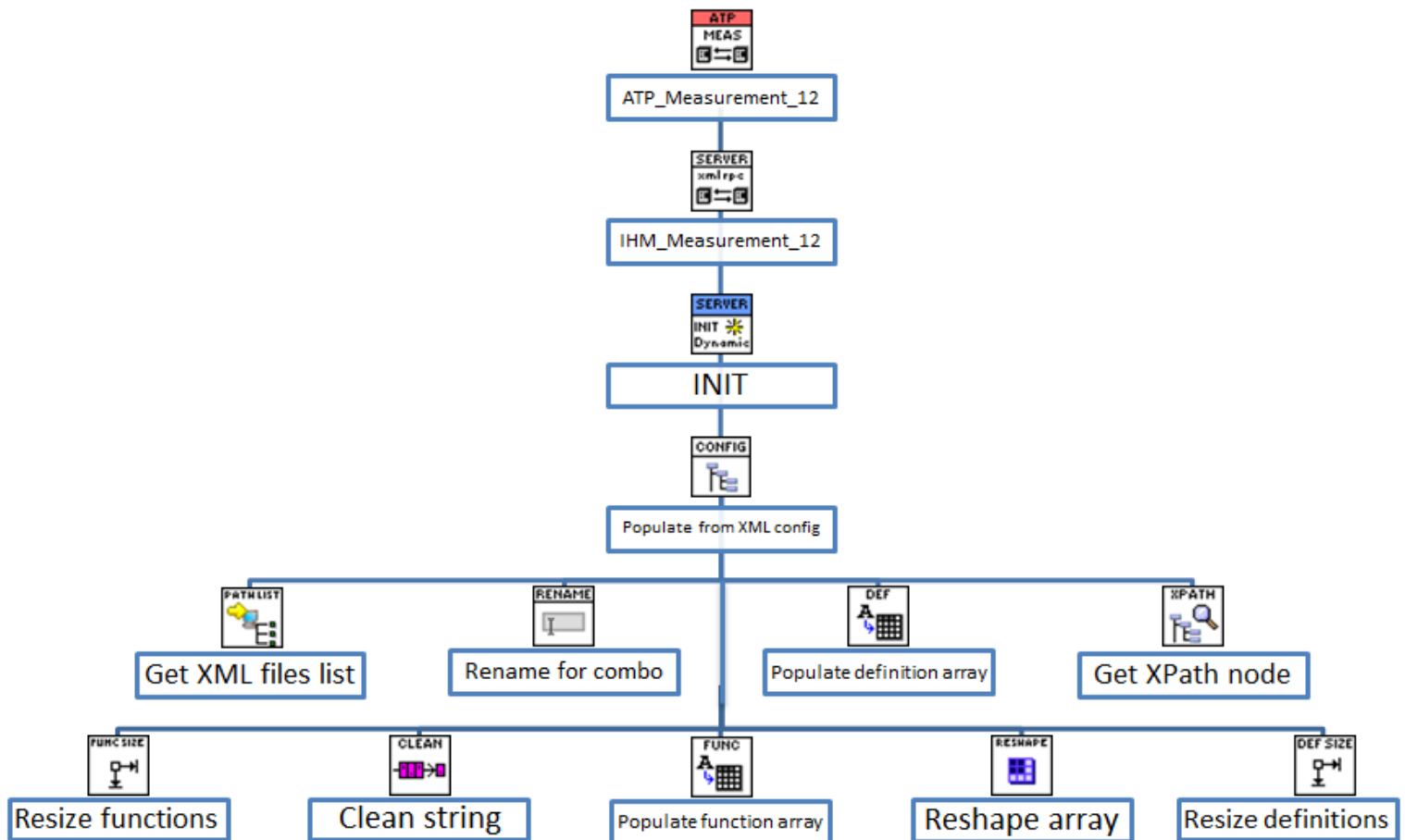


Figure 4: Structure of VIs for populating from XML configuration

2.3.1. Main VI

CONFIG	Input	Output
	Configuration files path	Final array

The output array is technically a 1D array of clusters of strings and 2D arrays of strings, as seen in figure 5.

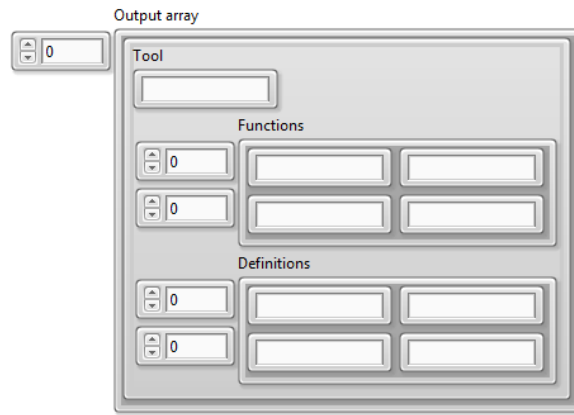


Figure 5: Output array structure

Each row of the output array will be specific to a tool, whose name will be displayed in the “tool” string. Then the “Functions” and “Definitions” array respectively contain **all the properties of the tool**, such as functions, arguments and values. It is a representation, in an array, of the XML files described previously.

The **main steps** of this VI are:

- We initialize the arrays and values, and populate the array of configuration files path.
- For every different tool, we initialize the runtime parameters, before populating the definitions and functions arrays
- We use Xpaths to find the specific data, which are hard-coded in the populate VIs
- A reshape and resize step allows to modify the arrays so that we do not have any blank rows or columns
- We rename arguments and values for later use in the HMI
- We finally bundle all this data in the final output array

Now we can get into the details of every subVI used in this main one.

2.3.2. Get XML files list (SubVI).vi

PATH LIST	Input	Output
	Configuration files path	Array of paths for each file

Lists the XML files present in the specified configuration folder, into an array of paths.

2.3.3. Get XPath node (SubVI).vi

XPATH	Input	Output
	Configuration file path	Array of nodes described by the XPath
	Xpath string	

Gets all the nodes which are described in the input Xpath, and returns them in an array. **XPath** (XML Path Language) is a query language for selecting nodes from an XML document.

Examples:


- **//GENERIC_TOOL/functions/function/@functionDisplayName** refers to the value of the "function display name" in that specific path of tags.
- **//GENERIC_TOOL/definitions/data[@name="ResultStruct"]/@type** refers to the type of the data, whose name is "ResultStruct".

2.3.4. Clean string (SubVI).vi

	Input	Output
	A string in between quotation mark "..."	The string which was inside


Keeps the value of the string which is in between quotation marks.

2.3.5. Populate definition array (SubVI).vi

	Input	Output
	Array of data names found by Xpath	
	Configuration file path	
	Empty definitions array: Def	
	Number of attributes in definitions	
	Number of columns in definitions array	

For each input function of the For loop, get the values of the attributes given in the list of Xpaths, and populate the definitions array. At the same time, count the number of attributes found, and the number of columns written, to be used when reshaping the array later on.

2.3.6. Populate function array (SubVI).vi

	Input	Output
	Array of function names found by Xpath	
	Empty functions array: Func	
	Number of columns in functions array	
	Number of arguments in functions array	
	Number of rows in functions array	


For each input function of the For loop, get the values of the arguments given in the Xpaths, and populate the functions array. At the same time, count the number of arguments found, and the number of columns and rows written, to be used when reshaping the array later on. And add a row to the array only for every new argument.

2.3.7. Reshape array (SubVI).vi

	Input	Output
	Array	Array reshaped


Delete empty rows and columns of the array to make it as small as possible.

2.3.8. Resize functions (SubVI).vi

	Input	Output
	Functions array: Func	
	Functions array reference in output Array	

Resize the display of the functions array in the output array, to the smallest array including all the data.

2.3.9. Resize definitions (SubVI).vi

	Input	Output
	Definitions array: Def	
	Definitions array reference in output Array	

Resize the display of the definitions array in the output array, to the smallest array including all the data.

2.3.10. Rename for combo (SubVI).vi

	Input	Output
	Output array	Modified output array

Rename the argument column, to have "function.argument", and for every value rename it to: "function.argument.value". This allows us to update the HMI values easily.

3. XML-RPC class

3.1. XML-RPC.ctl

The **control** is the main element of the class: it contains the data exchanged; it is composed of:


- The "addresses" array which contains the IP and port of every different tool. It is obtained from the .ini file described earlier.
- The array populated with the XML config files, containing all the different properties and functions of each tool.

3.2. INIT.vi

	Input	Output
	XML-RPC class in	XML-RPC class out


This VI is used to initialize the previous control. One part of it uses the “populate from XML config” VI to populate the array of data in the main program, and the other part reads and gets the data from the .INI file, by reading sections and keys of the file.

3.3. XML-RPC_main.vi

	Input	Output
	XML-RPC class in	XML-RPC class out
	XML string	Response body
	Tool used	

This program is the key component to allow the XML-RPC communication. First, according to the tool used, it creates the proper address with an IP and a port. Then it sends the input XML string **via HTTP**, and gets a response back, which is output. The “HTTP Post” VI is native to LabVIEW; thanks to a given URL it is possible to send a message through the server. The message we send has an **HTML format**, to be able to be understood by the tools.

3.4. Read config cluster.vi

	Input	Output
	XML-RPC class in	XML-RPC class out
		Configuration cluster

This is a simple program that unbundles the configuration cluster in the XML-RPC class to the output. It is done here, simply because it cannot be done in the main VI.

4. Tool setup

4.1. Global drivers

Path in project: Drivers/_Commun/GLOBAL_DRIVERS_Objects.vi

This VI is defined as a global variable, which allows to declare the resources available for the software. Each resource corresponds to an object of the class for the type of equipment.

The name of the resource has to correspond to the name of the configuration folder; in our case “XML_RPC_1”.

We simply need to **add the object class** for the driver we want to use.



4.2. Manual mode drivers

Path in project: Drivers/_Commun/DRIVERS_ManualMode.vi

This VI allows us to customize the tools available in the Manual mode bar. For the XML-RPC tool, the button was designed and placed in Custom_control/Control_PageView, as seen in the next figure.

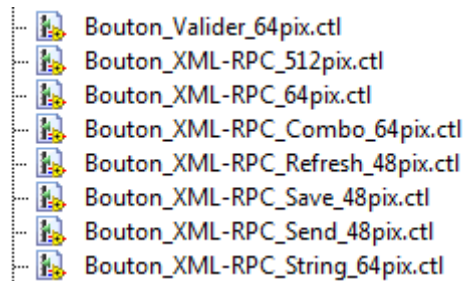


Figure 6: New button design

The next figure shows us the flat design button of the XML-RPC tool, when normal (left) and pushed down (right).



Figure 7: Tool button design

Notice that **label names** are very important for each button: they have to redirect to the ATP/HMI in question. For a measure: " MEAS_ ", and for a command "CMD_ "; followed by the unique number of the driver. In our case, the XML-RPC tool is a measure, indexed with the number 18, is the label is "**Meas_18**".

5. HMI

5.1. TypeDef_XML-RPC_clt_config.ctl

This control is a key element to transfer data **from the HMI to the ATP**, and save the parameters in a variable. It is, visible in figure 8, a **cluster** composed of:

- A "Tool" comboBox
- A "Function" comboBox
- An "Argument(s)" array with a string with the name of a parameter, next to a comboBox with its possible values.

Tool
 ▼

Function
 ▼

Argument(s)

<input type="text"/>	<input type="text"/> ▼
<input type="text"/>	<input type="text"/> ▼
<input type="text"/>	<input type="text"/> ▼
<input type="text"/>	<input type="text"/> ▼

Figure 8: TypeDef cluster

5.2 ATP_Measurement_12.vi

ATP MEAS	Input	Output
	step_STATUS in	step_STATUS out
	Editor mode boolean	value

The ATP is just a step from the main control to the HMI. It gets the data from the HMI, processes it and **makes it available** for other VIs. It also allows us to write a comment in the report thanks to the ATP_comment VI, which writes in a string the information sent back from the tools.

ATP_comment.vi :



5.3 IHM_Measurement_12.vi

SERVER xmlrpc	Input	Output
	Configuration	Configuration out
	IHM control?	TAG
	Editor mode?	Value
		Output results

5.3.1 HMI interface

This is the main VI for the XML-RPC tool, which is also where the HMI is defined as seen on the next figure.

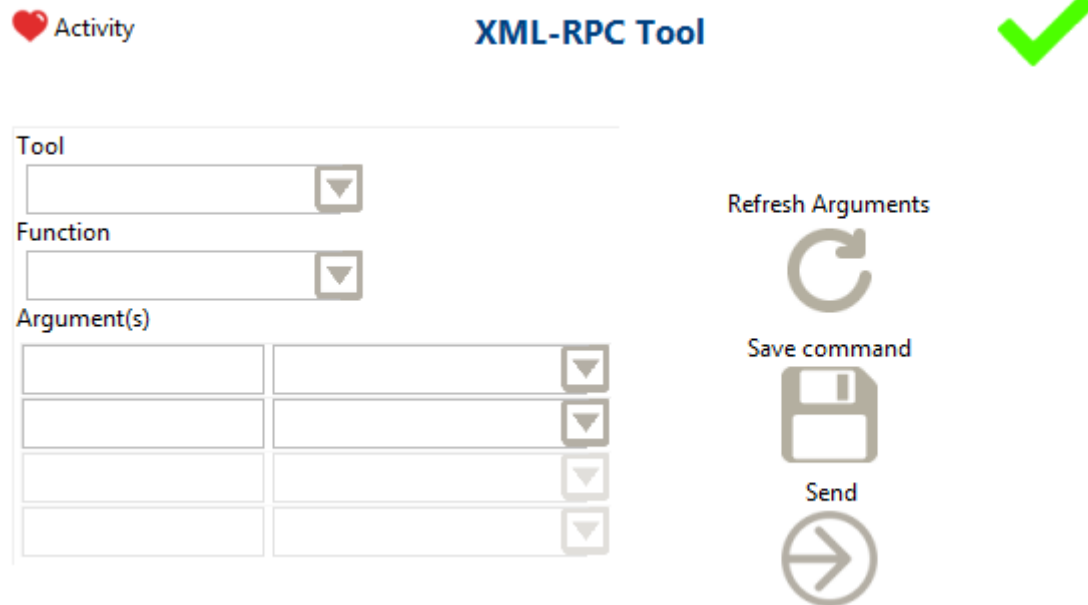


Figure 9: HMI of the XML-RPC tool

On the left side we can spot the typeDef control, which is the “IHM parameters” in the program.

On the right side we have three buttons:

- “Refresh arguments” allows to **update** the list of functions or Arguments when a new tool or function is chosen (if not done automatically).
- “Save command” allows to **save** the chosen configuration before sending it to the tool or writing it on the test procedure.
- “Send”, which is only available in Manual Mode, allows to **send** the desired parameters to the chosen tool.
-

5.3.2 HMI modes

The program is built to have a different behavior for **three different modes**, which are controlled thanks to **two Booleans**: “IHM control?” and “Editor Mode?” which are both False by default. The following table defines which mode we are in, according to these Booleans:

	Editor Mode	IHM control
Manual mode	F	T
Editor	T	T
Sequencer	F	F

- The Manual mode allows us to choose and send directly the command to the tool by clicking on send.
- The Editor mode is useful to create a step for a test in a procedure.
- The Sequencer mode executes the procedures without showing the HMI.

5.3.3 Block diagram template

As for the block diagram behind the HMI, the template/very simplified version looks like this:

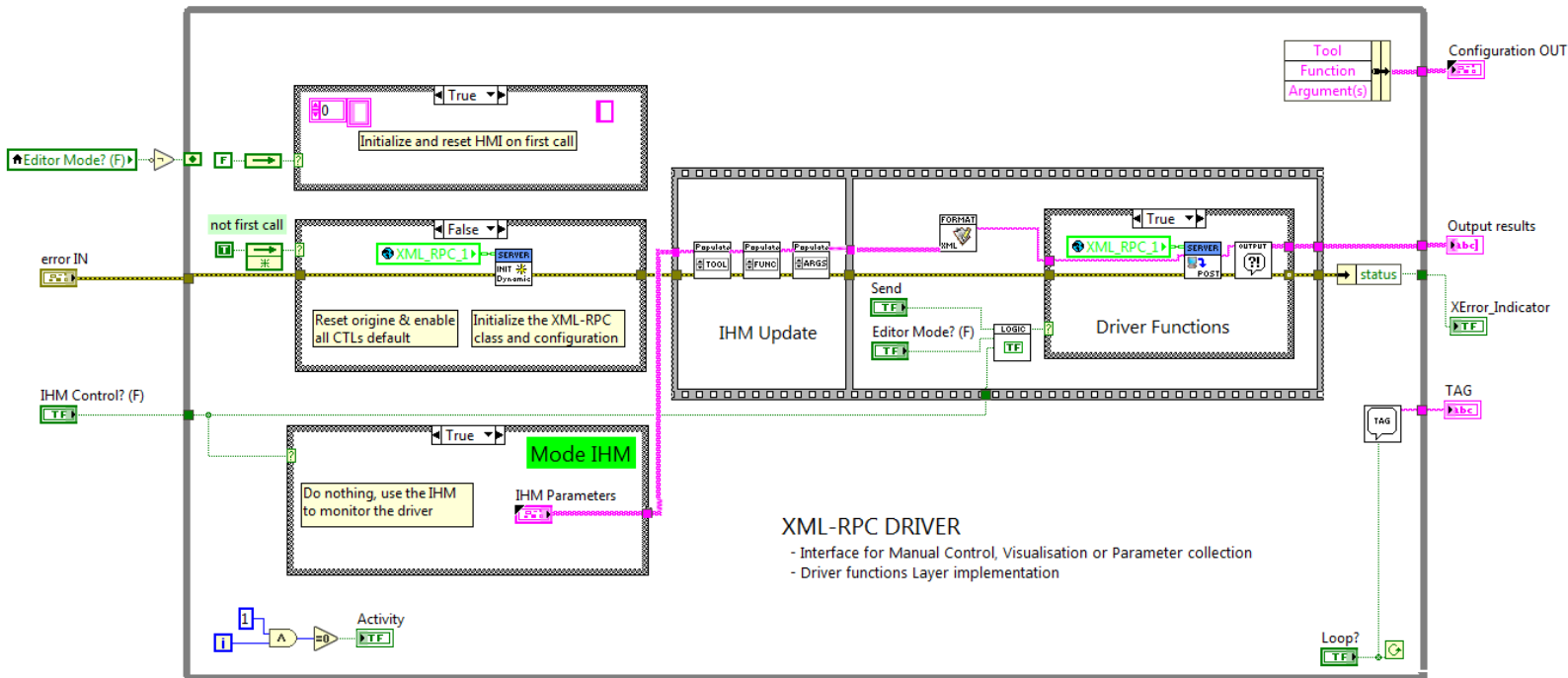


Figure 10: Block diagram template of the XML-RPC tool

The bottom left case structure controls the information to send (**Configuration control** or **IHM parameters**), depending if we are in sub-VI or HMI mode. The HMI mode is used for Manual mode and when creating a test, whereas the sub-VI mode is used when a sequence is executed.

The middle left case structure allows us to **initialize all the parameters** and controls useful for the execution of the VI: it is called the INIT step. It is called only once during the first call, then we just reinject the same data to the program.

The top left case structure allows us to **reset all the HMI indicators to blank**, so they can be refreshed with new input values. It is done every time the VI is called, except in Editor Mode, because when we reopen the VI, we want the previously chosen values to be displayed.

The flat sequence is divided in two parts:

- The HMI update one, which allows to **update** the data in the strings and combo boxes of the "IHM parameters" indicator of the HMI.
- The "driver functions" one, which allows to **send** the data to the tools via the XML-RPC server. It is only activated in sequencer mode or when we press "Send" in manual mode.


5.3.4 INIT step

5.3.4.1 TOOLS_Reset_All_Panes_Origin.vi

 TOOLS Reset Panor Origin	Input	Output
	Origin	

This simple VI allows to reset the origin of all panes on the screen for display. Present for each driver.


5.3.4.2 CTL_Enable_ALL_VI_CTLs.vi

 CTL Enable ALL CTL	Input	Output
	VI Refnum	VI Refnum out
	Set Default? (F)	

Here, we get the list of all controls on the front pane, to initialize them. Also present for each driver.


5.3.5 HMI update

5.3.5.1 Populate_tools (SubVI).vi

 Populate TOOL	Input	Output
	Tool ComboBox reference	
	Config Array	


This simple VI gets the name of **all the tools** referenced in the configuration array, and populates the combo box of the HMI for the user to choose the desired one to work with.

5.3.5.2 Populate_functions (SubVI).vi

 Populate FUNC	Input	Output
	Function ComboBox in	
	IHM Parameters	
	Input array	

When a tool is selected, this VI updates the function combo box to the **list of functions** available for the chosen tool. It gets its data from the input array and is triggered when the tool combo box changes value.

5.3.5.3 Populate_Argument(s) (SubVI).vi


	Input	Output
	Argument(s) reference	
	IHM parameters reference	
	Function	
	Input array	

This VI allows to populate the strings in the Argument(s) array: for the chosen function, it finds **all the possible arguments**, and writes their name in the HMI array (as well as resizing it).

It finds the relevant argument name thanks to the populate_IDs (SubVI), which looks in the Input array for the data. The **IDs** in question are "function.argument".


Also, a similar VI populates the values of the related combo boxes with the **default value** of the argument (if it has one), thanks to the populate_defaults (SubVI).

5.3.5.4 Coordinates to Index

	Input	Output
	Coordinates	Vertical in Range?
	Array ref	

Here, we get the row selected in the array, **based on the mouse location**. So the program knows which row of the array with the argument names we are in, to be able to update the related combo box dynamically.

5.3.5.5 Populate_comboBoxes (SubVI).vi

	Input	Output
	Function	Argument specific Values
	Array of all values	
	IHM Parameters	

Depending on where the mouse is when we click, only the **relevant values** of the argument will appear.

In reality, every combo boxes have all the values listed in them, but we only show the ones related to the desired argument.


5.3.6 Driver functions

5.3.6.1 Get type (SubVI).vi

	Input	Output
	IHM parameters	Types array
	Array in	

This VI looks for the type of the arguments in the IHM parameters. It outputs all the types in an array that is read later on when transforming the data to an XML format.

5.3.6.2 Format to XML (SubVI).vi

	Input	Output
	Method name	XML string
	Values	

This VI transforms the input set of function and values to an **XML string format** readable by the tools.

The next figure gives us an example of the XML string in question:

```
<methodCall>
  <methodName> Function name </methodName>
  <params>
    <param>
      <value><string> First value </string></value>
      <value><string> Second value </string></value>
    </param>
  </params>
</methodCall>
```


Figure 11: Output XML string

5.3.6.3 Multi-output message filter (SubVI).vi

	Input	Output
	Return XML body	Output results

Here, we read through the XML body that we got back from the XML post (XML-RPC_main.vi), and look for all the **returns of the tool**. These returns can be values of a parameter of the tool, and also the status of the tool and received command.

5.3.6.4 Values_for_tag (SubVI).vi

	Input	Output
	Tool	TAG
	Function	
	Values Array	
	Loop?	


Once we have finished choosing all the IHM parameters for a test, this VI creates a TAG to write in the report, which gives:

- the tool's name
- the function's name
- the values' name(s)

In this format: **Tool.Function(Values)**

6 Report generation

The report generation is an important step for the user: it allows him to read the procedure he has written, and to determine whether steps were **OK** or **KO** during execution.

It is done in the  HIST_Read-Write_File VI (situated in C:\WORK\TAES_F5X\MAIN_SW_GENERIC\FilesIO\HIST_Read-Write_File.vi). By default the software generates its own report with a .hist extension, but the next figure shows us the part of the program allowing us to **generate the excel report**.

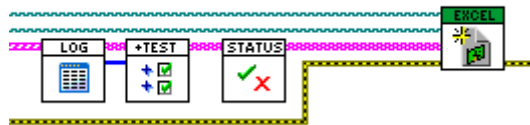


Figure 12: XLS report generation

The report is composed of a header detailing the characteristics of the test; followed by a table where we can see the procedure we have executed, and the result and status of each step as in the following example:

Test	Action	Comment	Status
Sequence Initialization			OK
	Sequence Initialization	Sequence Initialization OK	OK
Test folder\French_Flag			OK
	Demonstrator : change_color(green)	Server Return : green	OK
	Wait (1,00s)	1,00s wait done	OK

The figure 13 details the dependencies of the Excel generation set of VIs (placed in HIST_Read-Write_File).

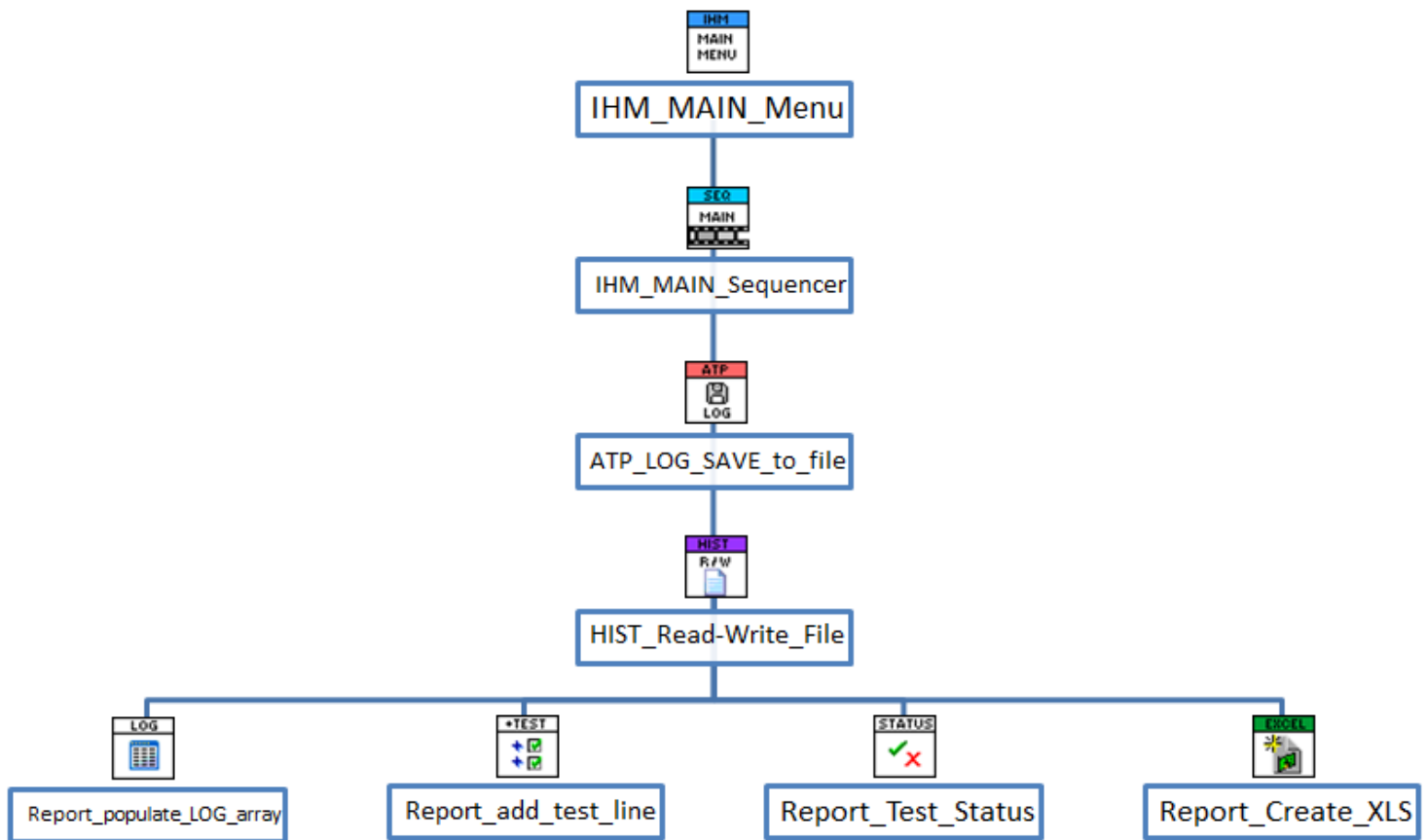


Figure 13: Structure of VIs for XLS report generation


6.1 Report_populate_LOG_array(SubVI).vi

LOG	Input	Output
	Sequence LOG (write mode)	LOG array
		Index change array

From the HIST_Read-Write_File VI, we obtain an array containing several elements in a specific format. Our aim is to extract specific data to a 2D array, to be able to use it later one. Namely, we are interested in:

- The TEST_Index, which is the number of the test
- The TEST_STATUS's current step, which is the name of the current test (**Test**)
- The Step_STATUS's status (**Status**) & Current Step (**Action**) & Comment (**Comment**)

6.2 Report_add_test_line(SubVI).vi

	Input	Output
	LOG array in	LOG array rearranged
	Index change array	


This VI allows modifying the appearance of the LOG array: each test is regrouped, thanks to the index array, and the left column only shows once the name of the test as a title on one empty line.

6.3 Report_Test_Status(SubVI).vi

	Input	Output
	LOG array rearranged	LOG array rearranged and with status

Here, we simply check the **global status of each test** and write it down. The idea is a test is OK only if all the steps are OK, else it is KO.

6.4 Report_Create_XLS(SubVI).vi

	Input	Output
	Report file path in	
	Template file path	
	Final LOG array	

This VI uses functionalities included in LabVIEW to **populate an excel file** with the input LOG array, and save it. We use an excel file as a **template** for the header and the table titles.

7 More

For information about the software itself, see with the C.E.D.M. Team at Assystem Technologies. It is easily made possible to add new tools/drivers to this main software.

Abstract

Assystem Technologies is an international engineering and innovation consulting group, which assists major industrialists in the various stages of their projects. I realized my thesis in Toulouse, France, with the "automatic test" team. They mainly work for the customer Airbus on the support and validation of a sequencer, performing test procedures on aeronautical systems test benches.

During my time in Assystem Technologies, the aim of my work was to propose innovative solutions for the Assystem Technologies sequencer technology, in order to implement a prototype which can be shown as a demonstrator to the company customers, detaching from the current specific Airbus implementation. The current sequencer was developed under LabVIEW by a team in the company, and the challenge of my internship was the integration of an XML-RPC server to pilot new generic tools, such as a robot arm and an image recognition software.

The benefits of the proposed implementation are the simplicity and effectiveness of the XML-RPC server, handling the communication between the software and the tools.

Final output of the internship was a generic software, which is now Assystem Technologies' property, that allows writing procedures, executing them, and generating a user-readable report.

Keywords: demonstrator, test procedure, sequencer, XML-RPC server, LabVIEW, generic

