

# A FLEXIBLE SOLUTION TO DEPLOY AVIONICS DISPLAYS TO MULTIPLE EMBEDDED PLATFORMS

*Yannick Lefebvre, Presagis, Montréal, Québec (Canada)*

## Abstract

Avionics HMI developers depend on both in-house and COTS HMI tools in their day-to-day work. Unfortunately, many of these solutions support only a narrow set of embedded computing platforms or do not meet the performance requirements that a program defines. This means that engineering choices can be limited for HMI software and hardware combinations at the initial design phase and that seemingly straightforward hardware upgrades during an aircraft's lifecycle can become problematic.

Beyond these hardware constraints, HMI designers often have to deal with a disjunction between groups developing the prototypes and those deploying to the target platform whereby applications are frequently completely re-coded for the final environment, resulting in little opportunity for re-use from prototyping to production. This type of sequential re-development of a display makes it very difficult to make any changes to the design late in the development cycle or even to be able to evaluate a display iteratively in its production environment.

To overcome these restrictions and inefficiencies, engineers can invest in vendor-agnostic COTS HMI tools, such as VAPS XT from Presagis. By grouping all function calls that are specific to an operating system or a graphic library, the porting layer acts as a bridge between the HMI design and embedded computing platforms.

There are many benefits to using a tool based around a porting layer, including:

- increase design flexibility
- enabling HMI designers to keep refining their displays until the very end of a project cycle
- facilitate avionics updates after initial program deployment
- deployment on multiple embedded platforms

- injection of new features through product upgrades during the development cycle

Whether customized in-house or by a third-party service, the porting layer eliminates the constraints associated with choosing COTS embedded software and hardware based solely on existing interoperability.

## The drawbacks of in-house tools

### *Timelines and budget*

In the past decade, there has been a steady growth in the acceptance and use of model-based tools to create Human-Machine interfaces (HMIs). However, some projects still use hand-coding or home-grown tools to produce the code driving their products and to present information to the end-user. In this era of increasing productivity expectations and shorter deadlines, these methods are not always able to produce the desired results.

### *Turnover and maintenance upgrades*

Another drawback of hand-coding and internal technologies is that it produces results that are difficult to maintain over a vehicle's lifespan and even sometimes hard to update iteratively during the initial definition of a display.

This is often due to the reality that engineers assigned to create the initial delivery of a display often move on to different projects upon completion, leaving their replacements with poorly documented code and a very steep learning curve. In such a situation, it becomes very difficult to be able to perform the necessary corrections and minor upgrades that are commonly required on many aircrafts.

During the initial development cycle, internal product development often revolves around a few key individuals whose time becomes a rare commodity. This makes it difficult to explore new display concepts as they cannot implement them within their schedules.

## ***Platform-Specific Results***

A limiting aspect of hand-written code and in-house tools is that it is very often tied to a specific operating system, a specific computing architecture or a specific graphic library (OpenGL or other proprietary graphics API). This is typically due to tight deadlines and the lack of directives to create flexible code as part of the overall design. This makes it difficult to change vendors during an aircraft's lifecycle or to adapt working code to a new architecture that is dictated by the obsolescence of the previous hardware.

## ***Support for new standards***

When new architectural design standards such as ARINC 661 are introduced to the industry, it is often very difficult— if not impossible— to adapt home-grown technologies to the new design concepts since the cost would be prohibitive. ARINC 661 is an architecture standard that is currently primarily used in commercial avionics on projects such as the Boeing 787 as well as the Airbus A380 and A400M which defines a standard set of GUI-building components as well as a runtime communication protocol and separation between graphics and logic.

## ***Inter-company collaboration***

While using a home-grown tool in-house might be acceptable, it often becomes an issue when more than one company is involved in a project. The minimal internal documentation, along with the lack of a dedicated support team, makes it very difficult to standardize on this type of technology when working on a partnership project. Beyond these practical aspects, companies typically don't always want to share their intellectual property with other companies who might be their competitors for other projects.

## ***Limited Re-Use***

Finally, re-use across multiple projects is almost impossible as each program tends to have very specific graphical and functional requirements and code is usually not organized in a modular fashion to facilitate re-use. Some companies have been able to create derivatives of their work when two projects share a very similar look-and-feel but

it becomes a lot more difficult when applications are very different or when there is a need to only re-use part of a code base. Re-creating a new design from scratch often becomes more attractive than trying to re-use an existing code base.

## **Drawbacks of Platform-Specific Solutions**

Amongst the various HMI tools available today, some have been designed to work on a specific number of embedded targets while others are providing solutions that can be adapted to an unlimited number of environments. While the comfort of acquiring a solution that works for a specific platform today can seem like a very strong argument in selecting a technology, changes to the hardware for budgetary or obsolescence reasons as well as updates to the operating system need to be kept in mind when selecting an HMI solution.

## **The current state of HMI tools**

Before delving into the specifics surrounding the creation and use of a porting layer, let's take a look at the current capabilities of HMI development tools. While there are many commercial solutions out there today, a complete solution should offer most if not all of these features within their respective development environments.



**Figure 1 Primary Flight Display created with VAPS XT**

## Graphical content creation

By using a graphical editor and an extensible architecture, HMI design tools such as VAPS XT allow users to focus on the visual appearance and functional aspects of their displays while more advanced programmers can augment the tool as necessary by creating new visual components as well as interfaces to external data sources and hardware components. When modifications to a display are required after the initial delivery, all of the design artifacts can be loaded back into the editor in order to easily create a new revision of the display code.

Beyond the ability to define the visual appearance of a display, HMI design tools are also used to specify the behavior and data ranges of all elements that are rendered on-screen, to define the connectivity between these components and their respective data sources (internal or external) and to define more advanced logic between the components on interactive systems that include user interaction and multiple pages.

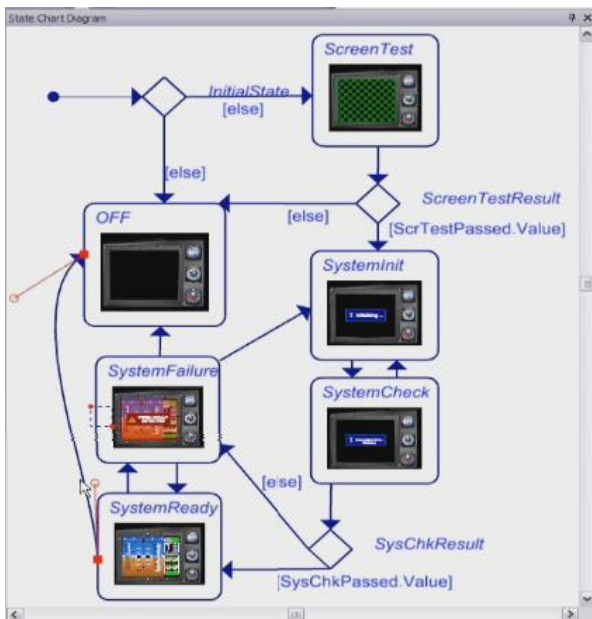


Figure 2 UML-based Statechart Display Logic

## Component Re-Use

By following an object-oriented design to create their display pages and custom components, companies can re-use elements that they have

created for one aircraft across multiple projects through simple drag-and-drop operations.

In VAPS XT, users can modify the look-and-feel of existing object and put them in an object library to facilitate sharing. It is also possible for HMI designers to create brand-new components to implement functionality that is specific to their display design. In such a case, it is possible to use all of the power of the graphical interface to create new components with user-defined properties, events and methods. Alternatively, advanced programmers can create new custom components through direct C++ programming. This flexibility allows users to create some very complex visual or computational components.

After they are added to the design environment, new user-created components have the same appearance as pre-defined elements and can easily be documented and re-used across multiple projects. This enables all users to use them and easily configure them for use in their own projects.

## Support for new standards

For new emerging standards like ARINC 661, a COTS tool such as VAPS XT provides the ability to edit and save content by following the standard as well as the ability to create embeddable CDS (Cockpit Display System) code that can load user-created definition files.

## Test Mode

During the display creation process, users can regularly run the display in a test environment to validate that the functionality they implemented works as expected and to inspect the data that is being sent between objects and external data sources. The test mode instantaneously produces the same results as compiling the application in an executable form.

## Document Generation

Another appeal of using COTS HMI solutions is when they provide the ability to automatically generate documentation from the model-based design. This allows users to extract images and information about all aspects of the display design,

including custom user comments, and output them in a user-configurable format. Once a document template is specified, specifications and technical documentation can easily be re-generated from the current model.

### ***Integration with Process Tools***

Selecting a tool to produce the HMI code is often only part of the software ecosystem that is used to produce a final result. Integration into other process-oriented tools, such as requirement tracking software and configuration management application can facilitate the complete lifecycle of an application.

At the requirement tracking level, the ability to automatically map one or more HMI design elements with one or more requirements in an application such as Telelogic DOORS makes it very quick and easy to keep track of requirement fulfillment and to produce the proper reports during the project execution.

On the configuration management side, having access to version control capabilities directly from the HMI tool interface makes it easier for developers to identify the current status of files and to be able to perform check-in and check-out operations quickly. Since there are many commercial packages available to perform configuration management tasks, it is preferable to use a tool that supports multiple environments such as ClearCase or Serena Changeman out of the box. For even more flexibility, the VAPS XT tool takes the approach of providing the source code to its version control interface to allow users to create new connections to third-party applications.

### ***Code Generation***

The true power of HMI tools resides in the ability to create stand-alone executables from the display designs once the user has finished creating them.

An automatic code generator analyzes the contents of the application, generates C++ code for all user-created content and compiles the results directly from the user interface into a stand-alone executable. The code can be compiled to run on the Windows platform or be re-targeted to any number of embedded platforms.

The resulting program can be executed without needing to have any additional product packages installed and can be used in a variety of environments:

- PC-based simulator
- Internal review or sending to customer for review
- Application testing
- Embedded systems

### ***Code Certification***

With the need to certify all code running on commercial aircrafts to DO-178B or ED-12B, as well as a growing requirement to produce military displays that follow these stringent coding standards without necessarily going through the actual certification process, using a COTS solution to produce display code becomes much more interesting than hand-coding a display or using an in-house solution. COTS solutions offer complete testing and certification artifacts for all of their internal runtime libraries. They also provide qualified code-generation tools that produce output that complies with the standards.

With all of these pieces of information in hand, the main remaining tasks for customers is to certify their custom porting layer and to run a subset of the test cases on the hardware to validate the results of the entire test suite that was executed by the software vendor.

## **Abstracting the hardware through the porting layer**

### ***Platform-agnostic code generation***

The first key to true platform-independence is to produce code that does not make direct reference to Operating System or graphics calls. This is the case with the C++ code that is produced by the VAPS XT CODE nGEN tool. It only contains the definition of the pages and objects that make up the display, as well as the logical relationships between these elements and any advanced state-based logic that was defined in the model-based editor.

Calls to the graphic card and operating system to render the display are made at runtime once the display code has been linked with a set of runtime libraries that are tailored for the target computing platform. In VAPS XT, the default runtime libraries that are delivered with the product are designed to run on the Microsoft Windows operating system and to render graphics using the OpenGL library.

To get the code running on a different combination of hardware and operating system, customers acquire the source code to the execution runtime libraries and customize them as necessary, then recompile them using the development tools provided for the target platform. This is typically referred to as the porting process.

### ***Embedded System Architecture / Porting Layer***

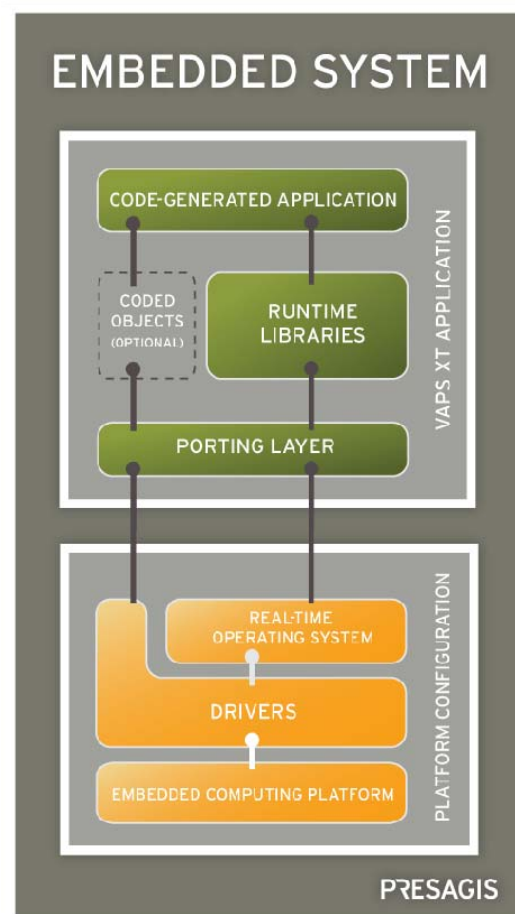
There are many different layers of code that produce the final display result when using a COTS HMI solution to generate displays. Besides the code that is produced by an automatic code generator from the display definition, the other elements of an executable produced by VAPS XT are the runtime libraries and the porting layer. It is also possible to see some user-created code to enable custom components that were programmed in C++ by the HMI developers.

The runtime libraries contain a multitude of internal mechanisms, including the scheduler, event handler, data propagation algorithms and general iteration update loops. This block of code is typically left unchanged when creating a new set of runtime libraries, unless a project requires a specific modification to its functionality.

Moving on to the porting later, this code section contains all of the calls to the OS and hardware drivers, including function calls to initialize the display, allocate memory, and initialize timers and callbacks (when available) as well as all graphic rendering commands.

By having direct access to the source code of the libraries, users can port code-generated applications to a large variety of platforms. While the default graphic rendering is done using OpenGL, the runtime libraries can be adapted to render other standard graphic libraries (such as DirectX or WinGDI on Windows), graphic libraries

optimized for embedded targets (e.g. OpenGL ES or OpenGL SC) or any custom API on embedded computing platforms (e.g. Fujitsu Jade API). As for the operating system, any commercial RTOS can host code-generated avionics displays, including Wind River VxWorks, QNX Neutrino, and Green Hills Integrity. For devices that have very limited resources, it is even possible to get executables running on a minimalist kernel that only offers basic services such as device initialization and timers.



**Figure 3 VAPS XT Embedded System Architecture**

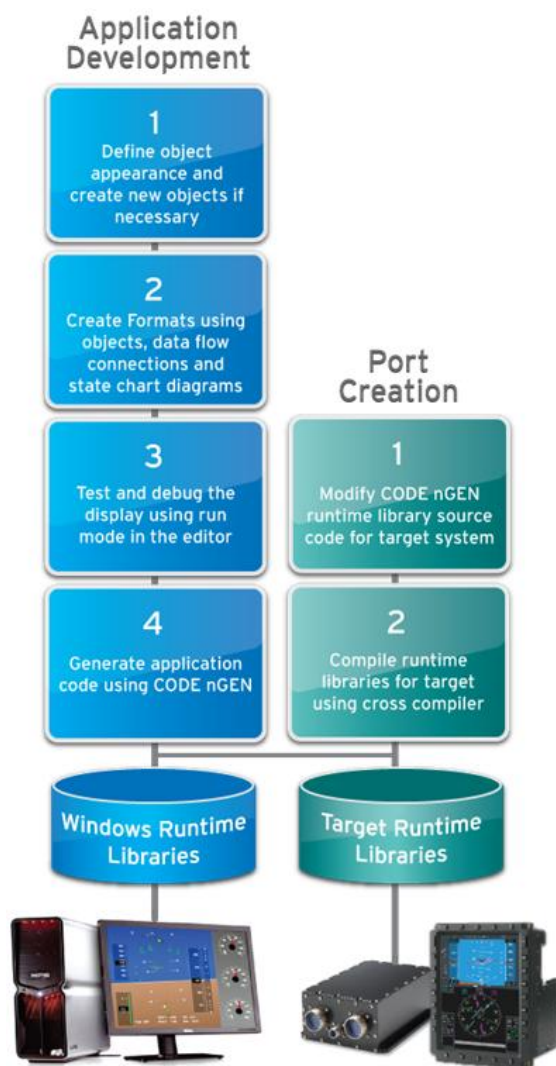
### ***Re-Using Tailored Runtime Libraries***

Once a set of runtime libraries has been ported for a specific target, it can be used to quickly build any application and see it running on embedded system. This allows developers to create multiple alternative display scenarios and get feedback from

pilots or other subject-matter experts in a realistic setting. It also promotes re-use of similar hardware across multiple airframes to be able to re-deploy the same porting layer.

It is also possible to create multiple sets of runtime libraries for different pieces of hardware, thus enabling an application created from a single design to be distributed to multiple targets. It also empowers avionics developers to change the hardware during the display creation process to benefit from improvements in technology or lower costs from an alternative vendor.

### ***Creating applications and porting layers in parallel***



**Figure 4 Application Creation Process**

Beyond giving greater flexibility to avionics developers to target any embedded environment, the availability of Windows-based runtime libraries and an interpreted run mode allows some engineers to work on the display application development while others focus on adapting the runtime libraries to work on the target system in parallel. In such a workflow, the display creation engineers can regularly create new executables and get feedback from their customers or from internal HMI specialists within their organization until a porting layer is ready. In many cases, users can even use the same display hardware that would go in the final aircraft (through the use of special adapters) to connect them to a PC in order to get the right display resolution and the same look-and-feel as the final product.

Once the porting layer is completed, it can be used by changing a simple user interface configuration. Having access to generate executables for the target from the HMI tool facilitates iterative development to be able to see the display running on the final production hardware and be able to quickly make changes and re-deploy the display on the target. The same porting layer can even be used on multiple projects that share the same hardware.

### ***Getting new features with upgrades***

By isolating all of the platform-specific calls in a single area, model-based COTS solutions like VAPS XT have the additional benefit of providing new features to users with every new release. While a current version of the software might satisfy the current needs of a project, there is very often the need to add functionality to a display system after it is originally fielded. COTS HMI solutions developers are constantly on the lookout to identify market trends and add new functionality to their tools.

When facing an upgrade to a new version of an HMI tool that uses a porting layer, the upgrade process completely revolves around re-applying the code customizations made to get the code running on the target to the new set of runtime libraries. Once a new set of runtime libraries is created, the model can quickly be loaded by the new user interface to enable display designers to take advantage of new product features.

## A closer look at the porting layer

As mentioned before, the porting layer is the sub-section of the runtime libraries that contains all calls that are operating system specific or tied to a particular graphic library.

In VAPS XT, all sections of code that are specific to the Windows environment or to the OpenGL graphic library have been carefully identified and documented throughout the entire porting layer. The documentation indicates the purpose of the original code. With this information in hand, finding the right functions to perform the equivalent tasks for the target hardware and

operating system can be done quickly and efficiently.

### *OpenGL code example*

In the following example, we can see code that enables or disables anti-aliasing based on the preference indicated by the display designer for each graphical element inside of the model-based tool. The engineer adapting the runtime libraries for the target system can easily identify that a change is required in this section and make the necessary modifications based on the required functionality.

```
// *** PORTING (Non-OpenGL) ***
// Enable anti-aliasing depending on the setting of the enable_flag.

if (gfx->is_anti_aliased != requested_state)
{
    gfx->is_anti_aliased = requested_state;
    if (requested_state == VXT_GAPI_IS_ANTIALIASED)
    {
        glEnable(GL_LINE_SMOOTH);
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    } else {
        glDisable(GL_LINE_SMOOTH);
        glDisable(GL_BLEND);
    }
}
```

**Figure 5 Sample OpenGL code in porting layer template**

```
// *** PORTING (Non-Windows) ***
// Retrieves the number of milliseconds that have elapsed since the
// system was started

return static_cast<vxtULong>(GetTickCount());
```

**Figure 6 Sample Windows code in Porting Layer**

### *Platform-Specific code example*

Similarly, in Figure 6 (above), we can see a block of code that makes a call that is specific to the Windows operating system. This call, accompanied by a description of the functionality that it was performing on the Windows platform, needs to be replaced by the proper function call to perform an equivalent task on the target system.

### *External Tool Configuration File*

The porting process also includes the creation of a configuration file that informs the code

generator and user interface of the compiler and linker tools that should be used to produce the custom runtime libraries and any application created for that target system. The configuration file also contains: a complete list of system libraries that should be used when linking the code, any pre-processor definitions that need to be present to activate the correct parts of multi-platform libraries and any header files that need to be present to get a successful build.

When compiler and linker tools are available on Windows to build code for the target environment, this configuration file enables users to

generate executable code for their target without ever leaving the graphical user interface.

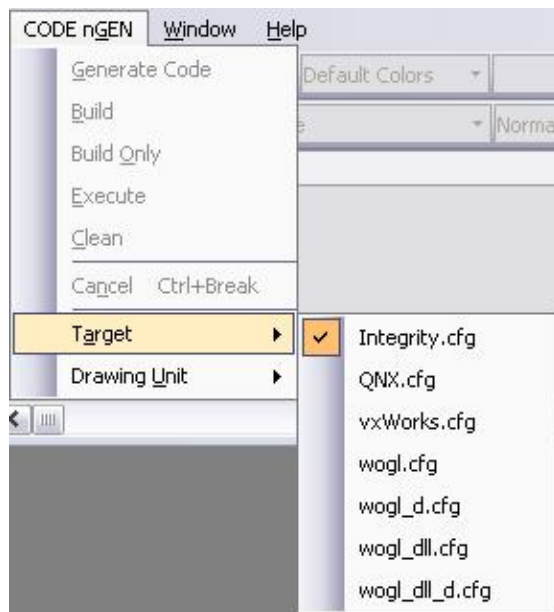
Figure 7 shows a few configuration parameters that describe the linker that should be called to produce an executable for the Windows platform.

```
# The following line specifies the linker.
LD=$(ECHO_ON_OFF)link.exe

# Add your linker options below if necessary.
# Include here any object files that must be linked with all applications.
LDOPTIONS= -nologo -SUBSYSTEM:CONSOLE -MACHINE:X86 /NODEFAULTLIB:nafxcw.lib
/NODEFAULTLIB:uafxcw.lib /delay:nobind /delayload:sqxComms.dll /delayload:sqxCommsHelper.dll
Delayimp.lib
```

**Figure 7 Sample External Tool Configuration File**

Once it is created, the configuration file is automatically added to the graphical user interface and selectable by the user. The build process can be launched in the same manner as one would select to build an executable for Windows.



**Figure 8 Selecting target platform**

### ***Compiling the code in other environments***

If the compilation environment for a given platform is not available for Windows, then the alternative route that needs to be followed is to move the code for the runtime libraries, along with all of the code produced by the code generator, to the development platform indicated by the target manufacturer. This could be done using an FTP tool or any other type of file-transfer application. For

By specifying the name of an executable and its command-line arguments, the user interface will be able to find and call the tool. It is also possible to specify complete paths for the tools to be sure that they are found by the build environment.

this purpose, the code generated by VAPS XT and all source code for runtime libraries are provided with makefiles instead of platform-specific project files.

In conjunction with the use of makefiles, using a multi-platform make tool such as gmake to generate executables on all platforms, including Windows, makes it a lot easier to move the resulting code from one environment to the next.

## **Summary**

As we saw in this document, beyond all of the display authoring benefits and time savings of using a COTS solution, selecting an offering that provides a flexible platform-agnostic porting layer can produce displays that will run on a variety of software and hardware environments.

This approach enables display manufacturers to create rich applications using a visual development environment and gives them the capability to freely change hardware solutions at any point in the development process without having to completely redesign the application.

## **Email Addresses**

The author can be reached at the following e-mail address:

yannick.lefebvre@presagis.com

*27th Digital Avionics Systems Conference  
October 26-30, 2008*