DOI 10.5162/ettc2018/12.2

A case study in creating flexible FTI configuration software

Alan Cooke
Curtiss-Wright, Dublin, Ireland,
acooke@curtisswright.com,

Abstract

This paper presents a case study in designing flexible, future-proofed FTI configuration software. It starts by outlining the challenges posed to software in supporting the combined FTI and related product lines of two historically disjoint customer bases that differ in their software user experience but still have the same high expectation levels when it comes to customer support. The paper then describes how Aerospace Instrumentation (AI) used this opportunity to reconfigure two software products to not only support historically separate product lines, but also to create greater choice for their customers in addition to providing greater flexibility and options for the future.

Key words: FTI, Software

Introduction

In an ideal world, Flight Test organizations could use hardware from multiple vendors to pick and choose the best solution. However, there are many issues that prevent this – one of which is the need to use different software suites to configure and manage the hardware. This paper outlines how software from two formally separate businesses are being adapted to allow customers to take full advantage of the vast product portfolio from their combined product lines. It also describes how this change has further enhanced the flexibility and options now available to its customers.

Integration Problems

Flight test programs require Data Acquisition Systems (DAS) to collect and process valuable data. These DAS consist of many elements including sensors, data acquisition units, recorders, switches and transmitters. Setting up these components is achieved through software. There are several vendors that supply some or all of these components and they all have their own software to support them. In an ideal world, engineers would be able to cherry pick equipment from multiple vendors, and re-use whatever equipment they have from previous programs, to build the system they need. However, trying to do this will typically result in integration problems because of the need to use multiple software suites. This means they are very likely to encounter incompatibility issues, for example, with data formats and time synchronization. Even if they can find ways to work around these issues, there is increased work associated with using multiple software suites. It also increases the risk of time delays as the system will be more complex with more things to go wrong.

While initiatives such as iNET are setting standards that should facilitate cross vendor hardware support in several different software packages, it will likely be some time before such standards are widely in use. Various other standards, such as XidML and MDL, have the potential to be part of the solution but without all vendors buying into working towards a universal software platform, engineers will still need to use separate software packages.

In the meantime, engineers in the field must contend with workarounds that reduce efficiency and add risk to a program by adding another layer of complication to a system that may result in delays. Vendors themselves can help here by integrating some equipment with their own prior to delivery. This is an effective solution, but it may add delays in delivery of systems, add cost and is not as flexible. This approach works best when there is limited equipment being integrated as more equipment from more vendors adds overhead to the integration effort.

Posible Solutions

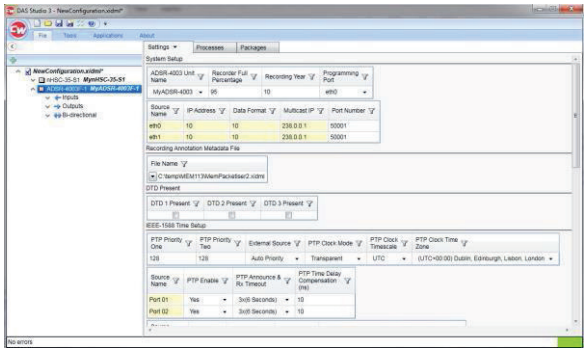
A common scenario in many flight test faculties and at aircraft OEMs is to try and persist with one vendor as long as is practical. This circumvents the problems associated with integrating multi-vendor systems and it also allows the re-use of database and visualization systems as well as negating the need for new product training and processes. The disadvantage is that organizations can find they are getting 'locked in' to certain product lines and may be selecting equipment not because it is the best fit for the job but because it is the most convenient overall.

Case Study

Background

Curtiss-Wright recently acquired Teletronics Technology Corporation (TTC) – a leading supplier of DAS. Curtiss-Wright was already a leading DAS supplier so it now has a combined product range that no other FTI vendor can hope to match. This provides an unprecedented opportunity for customers, but poses a significant challenge to Curtiss-Wright's TTCWare and DAS Studio configuration software.

Individually, both software packages are extremely flexible, each optimized to maximize the productivity of its users. They can both be used to configure Data Acquisition Units (DAU), Recorders, Switches, High-speed Cameras and much more. However, both applications take a different approach to configuring a system. Furthermore, each application has a large, well-established and loyal user base, making it undesirable to simply choose one of these applications as the sole configuration software for the Aerospace Instrumentation product range.



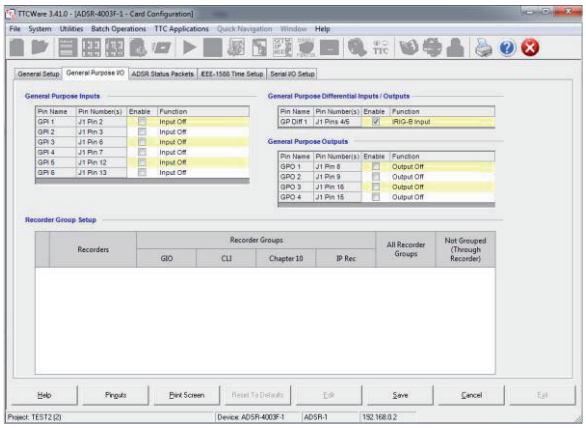


Figure 1 - TTCWare and DAS Studio are both highly capable, but they take different approaches

Without steps to effectively integrate their functionality, customers would have to use two separate pieces of software, maintain two different processes to configure hardware and be forced to support different configuration file formats. This is far from ideal for customers and the issues are similar to those faced by many Flight Test Organizations utilizing equipment from two or more DAS vendors.

Curtiss-Wright therefore initiated an integration effort that was aimed at avoiding the issues encountered when integrating equipment from two separate product lines. The initiative also aimed to allow users to benefit from more choice, greater equipment reuse and ultimately to provide greater overall flexibility.

Flexibile Software

As part of the reconfiguration of Aerospace Instrumentation's configuration software a number of constraints where imposed.

Design Constraints

A number of constraints were imposed on the reconfigured software.

1. The software must facilitate the existing processes and expectations of previously distinct customer bases.
2. Users should be given the choice of either using DAS Studio or TTCWare
3. The software should provide both GUI and Command line interfaces
4. Both configuration software needed to support hardware from both business units
5. The software must maintain the highest level of quality and reliability possible

In addition to these requirements the following objectives needed to be met

1. Add support for new Data Acquisition Cards as quickly as possible
2. Be able to extend the functionality of the software over time. As more and more capabilities are added to combined product line the software needs to be easily able to support these features without major rewrites
3. The ability to grow to meet (the perhaps unanticipated) future needs of customer. This could potentially include new Bus and Transmission protocols, new synchronization mechanism, sensor types, and so on.
4. All or some of the core functionality of both TTCWare and DAS Studio may need to be ported to other platforms (such as mobile devices or different Operating systems) or distributed across multiple platforms
5. Some or all of the core functionality may need to be incorporated into third party software
6. Support emerging and future standards such as iNET

There was also an implicit assumption that the software will need to be constantly reconfigured over time in unanticipated ways. In summary, the reconfigured software needed to be designed for *maximum flexibility*.

Building Flexible Software

The design constraints dictated a flexible design but what exactly is meant by flexible software and how can flexible software be built? From a software architecture point of view, software requirements fall in to two categories, *functional* and *non-functional* requirements. Flexibility is a *non-functional* requirement, and maps to what is known as a *Quality Attribute* [1]. Quality Attributes are used to guide the design of software, they generally do not specify specific technologies or functional features and essentially describe certain properties that a software design must possess.

The Quality Attribute of Flexibility, given the constraints described above can be further refined into other Quality Attributes. There exists a set of well know techniques that are designed to meet any given set of Quality Attributes.

Table 1 lists these.

Table 1 – The Quality Attributes associated with flexible software and techniques used to meet them

Attribute	Description
Modularity	Using reusable components to build software. It helps to maintain the flexibility by enforcing the separation of concerns between different functionally and semantically similar blocks of code. This better enables the reuse of different functionality across multiple pieces of software
Extensibility	This is the ability to easily extend the functionality of the software and to greatly reduce the fragility of the software. This is greatly helped by building software using semantically coherent Modules/Components
Portability	Software portability to multiple platforms i.e. different hardware platforms and OS
Reusability	The ability to reuse some or all of the functionality across multiple pieces of software
Testability	This is the ability to test automatically test the functionality of the software. This is a necessary to maintain the existing quality of software by automatically detecting regression issues in the software
Attribute	**Example Techniques**
Modularity	Code to Interfaces: Using this technique, code implement a specific public interface or "contract". Other code relies on or codes to these published interfaces and is ignorant of how the code is actually implemented behind the published interfaces Semantic Coherence: This is the practice of co-locating functionally related software in the same library or component.
Extensibility	Dependency Injection: The ability to "inject" functionality into other pieces of code, generally done by passing a module that implements a specific "Interface" into dependent code. Specifying in an external configuration file increase flexibility. Loose coupling: For maximum flexibility and to reduce the fragility of the software the various components/modules that compose a system should not be aware of each other. Specifically, the software should be design so that modules/communicate with each other using either a message based system such as a message broker, be event driven or perhaps some combination of both. Product Line Architectures: This technique can be used if variants of the same product are required. This is can be achieved using configuration files and is greatly eased by using some or all of the techniques discussed above [2] Plugins: This allows functionality that implements or conforms to specific interfaces to be automatically discovered and integrated at specific locations in the software
Portability	Layered Architecture: Where software is composed into two or more layers that increasingly abstract the software functionality from the physical platform or OS. A layered architecture combined with the use of loosely coupled, modular components in each layer also aid portability.
Reusability	APIs and SDKs: Creating a set of APIs and SDKs, accompanied by clear documentation can greatly increase the reusability of software. Using semantically coherent software modules helps to partition software into reusable units. Coding to Interfaces helps to isolate the specific implementation details from the software using the modules.
Testability	Coding to Interfaces help in the testing and verification of software by allowing test code to use "Mocking" techniques[1] to test code and to automate unit testing[2] Using layered architectures greatly increases the testability of software by allowing the various layers of the software to be tested separately. In particular, it allows the software that does not interact with hardware to be tested more easily.

[1] See https://en.wikipedia.org/wiki/Mock_object and https://github.com/Moq/moq4/wiki/Quickstart
[2] See https://en.wikipedia.org/wiki/Unit_testing and http://nunit.org/

Results

The following sections outline the results of the reconfiguration effort within Curtiss-Wright, Aerospace Instrumentation.

One software, Two User Interfaces

Both DAS Studio and TTCWare have been reconfigured to use a common functional core (

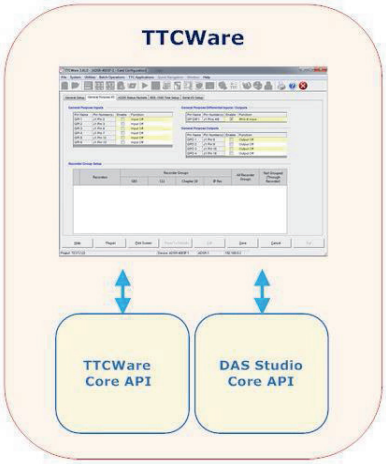
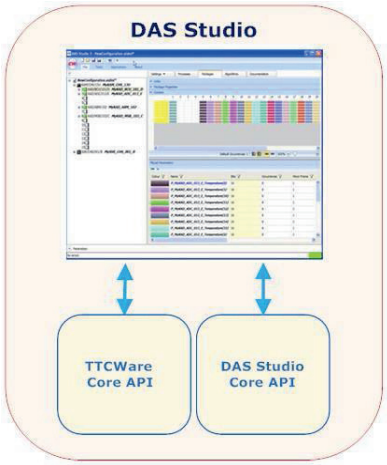


Figure 2).



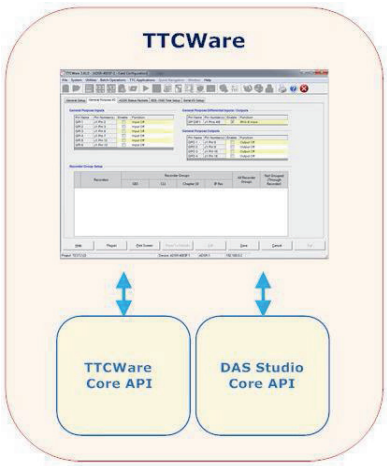


Figure 2 - Both TTCWare and DAS Studio use a common functional core

Specifically, the core business logic for both DAS Studio and TTCWare have been wrapped in separate APIs, and these in turn have been separated from the user interface layers. This permits support for hardware from either company to be added to either application with minimal effort. The approach also allows existing and future customers to use either TTCWare or DAS Studio as their preferred Graphical User Interface (GUI).

Discover & Program

The reconfigured software makes it easy to program mixed systems, for example, DAS Studio can communicate with, discover and configure equipment using native hardware protocols that would have formally only been achievable with TTCWare[3] (Figure 3).

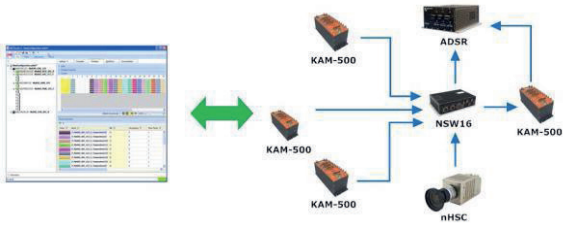


Figure 3 - Using DAS Studio to program a variety of equipment from formally different product lines

Similarly, TTCWare can discover and program equipment using native protocols that would have previously required DAS Studio (Figure 4).

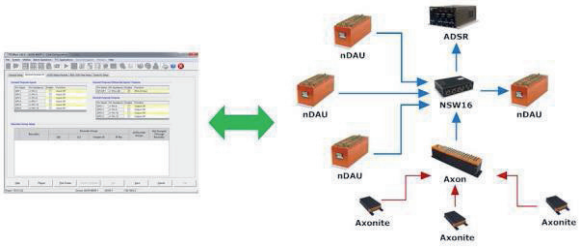


Figure 4 - TTCWare being used to program an Axon along with a variety of TTC equipment

This organizations will be able choose a wider mix of hardware than before and the choice of either a DAS Studio of TTCWare as their configuration software

Enhanced Flexibility

The common functional core also facilitates a greater choice for customers in how they choose to configure their systems.

[3] Such as TDDP protocol used to discover TTC equipment

DOI 10.5162/ettc2018/12.2

Support for Industry metadata standards

Aerospace Instrumentation engineers are very active on industry standard groups such as the RCC and iNET, and in particular, have made key contributions to the development of MDL (Measurement Definition Language). They are also founding members of the XidML community.

This expertise means that both TTCWare and DAS Studio offer unrivalled flexibility in how customers choose to describe their data acquisition systems. Customers can now use TTC XML, XidML and eventually MDL to define the structure of their data acquisition networks and how they are configured.

Command line programming

Many customers store their configuration data in company databases and other proprietary formats. They then process the data in these systems and convert it to file formats required by specific vendor FTI equipment.

To facilitate this approach, Aerospace Instrumentation provides a set of well-documented command line tools for both verifying configuration file formats and for using them to program their systems. This application takes a XidML file as input and can be used to verify the configuration and/or program one product line or a mix of the two product lines. The command line interface will also take an MDL file as input once the standard has been finalized and published.

Powerful APIs

In addition to giving customers the choice of which application to use (TTCWare, DAS Studio or Command Line Interface) a set of powerful APIs are also available as shown in Table 2.

XidML & XdefML APIs	Customers can use these APIs to create, manipulate and validate XidML and XdefML files. These APIs are used extensively in DAS Studio and TTCWare to generate configuration screens.
TTCWare Core API	This API gives users direct access to the core TTCWare functionality. It can be used by customers to discover and program TTCWare hardware, in addition the ability to define configurations and save/read them to/from TTC XML files.
DAS Studio Core API	This API gives customers direct access to the core DAS Studio business logic and functionality. It can be used by to discover and program DAS Studio hardware, in addition the ability to define configurations and save/read them to/from XidML files.
RESTful API	For extra flexibility, combined core API functionality is also accessible via a common RESTful interface.

Table 2- Available APIs

Together, these APIs provide a level of flexibility and choice that are not provided by other FTI vendors.

Built for the Future

The reconfiguration of TTCWare and DAS Studio was also carried out with an eye to the future. In particular, it was anticipated that customers may need the flexibility to deploy the software to different platforms and operating systems

Linux

All of the core APIs can now be run on the Linux operating system.

All command line interfaces are also capable of running on Linux, allowing customers to create and validate configuration files, in addition to programming both product lines.

Browser-based configuration

The RESTful API provides a mechanism for browser-based user interfaces (Figure 5) to configure and program hardware. As a result, the core functionality can now be hosted on individual DAUs and or even if required on company networks offering further flexibility to Aerospace Instrumentation customers.

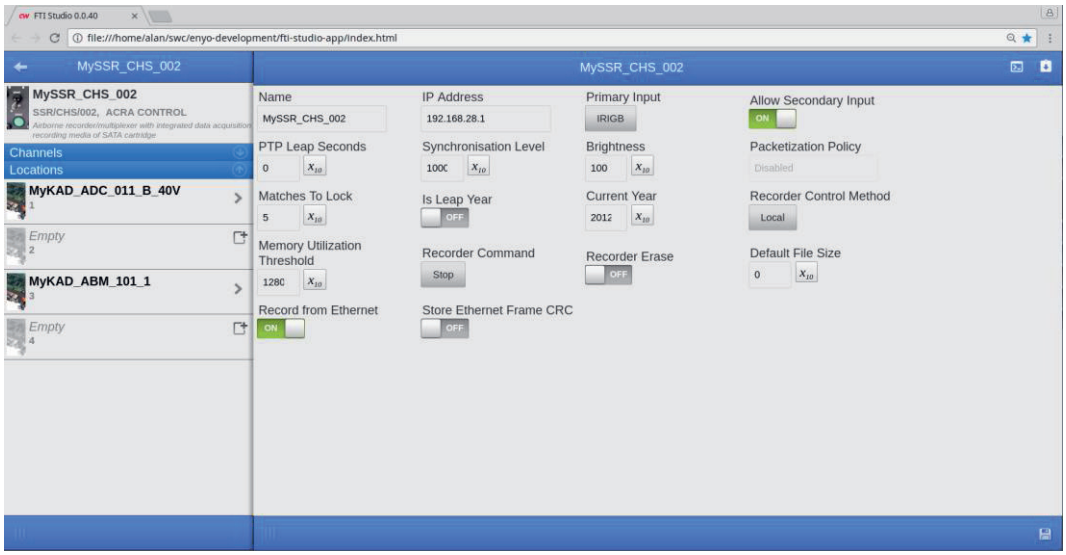


Figure 5 - Browser-based FTI configuration software

Mobile devices

Native Android, iOS and Windows Phone apps can also leverage the RESTful API to configure and program systems. In this scenario, the native mobile app connects to individual DAUs that host and run the RESTful FTI configuration services allowing users to adjust the configuration of the DAU.

Conclusion

Curtiss-Wright's full data acquisition product lines, under the umbrella of Aerospace Instrumentation, provide an unrivalled portfolio of products. This offers customers an unprecedented level of flexibility and choice but also presented a challenge to TTCWare and DAS Studio, the configuration software used for the product lines, as both applications have a wide and varied user base.

To address this challenge both TTCWare and DAS Studio have been reconfigured to allow equipment from either product line to be added to both configuration applications with minimal effort. This was achieved through the creation of a common functional core that is incorporated into both DAS Studio and TTCWare. The end result is that users will have the choice of using either TTCWare or DAS Studio for configuring system comprised of mixed hardware.

The reconfiguration of the software has also allowed Aerospace Instrumentation to offer even greater flexibility for customers by providing a set of powerful APIs and command line tools, support for multiple configuration file formats such as MDL, TTC XML and XidML, in addition to creating a pathway to future software support on multiple platforms and operating systems.

References

[1] L. Bass, P. Clements and R. Kazman, "*Software Architecture in Practice (SEI Series in Software Engineering (Hardcover))*," Addison-Wesley Professional; 3rd edition (September 25, 2012).

[2] P. Clements and L. Northrop, "*Software Product Lines*," Addison-Wesley Professional; 3rd edition (August 30, 2001).