Accepted version of: Carel J. Combrink, Chris J. Venter, Seshan Govender, and Mohammed A. Alshareef, "Reconfigurable, XML-driven, OO Framework for Real-time Control and Monitoring of Embedded Radar Signal Processor," *2011 Saudi International Electronics, Communications and Photonics Conference,* 24-26 April 2011, pp. 1-6. Published version available online:
http://ieeexplore.ieee.org/arnumber=5876938

Reconfigurable, XML-driven, OO Framework for Real-time Control and Monitoring of Embedded Radar Signal Processor

Carel J. Combrink, Chris J. Venter, Seshan Govender
Defence, Peace, Safety and Security
Council for Scientific and Industrial Research
Pretoria, South Africa

Mohammed A. Alshareef
Electronics, Communications and Photonics
King Abdulaziz City for Science and Technology
Riyadh, Saudi Arabia

Abstract—**A reconfigurable Control and Monitoring Framework (CMF) was developed, integrated and tested with a Radar Signal Processor (RSP). The methods and processes used to develop and utilise the CMF are highlighted in the context of a complete Control and Monitoring System (CMS) for an RSP. The CMF provides the basis for developing interactive, high-level control and monitoring applications using a multi-layer messaging infrastructure. The Unified Process was used for the development of the CMF. The CMF provides extensibility and modularity through the use of Object-Oriented (OO) code, but also as a result of extensive use of Extensible Markup Language (XML) configuration and description files. Design and scope changes are also minimized through the use of a dynamic scheme for addressing, controlling and monitoring hardware. The CMF has not only been utilised successfully for high-level control and monitoring of an RSP system, but is also being utilised further on projects and for applications being developed by the participating organizations. Working within a verified framework improves stability and reduces the development time of new implementations.**

Keywords- Object-oriented, XML, Reconfigurable, Framework, Radar

I. INTRODUCTION

Radar is an electromagnetic system used for the detection and location of objects. It operates by transmitting a particular type of waveform and detecting the nature of the echo signal [1]. The signal processing that is required to construct radio waves for transmission and to recognize and decode the reflected returns is referred to collectively as radar signal processing. Modern Radar Signal Processors (RSPs) make use of high speed digital circuits instead of analog circuits. Digital Signal Processor (DSP) and Field-Programmable Gate Array (FPGA) technologies are typically employed to perform high speed digital processing in Radar Signal Processing applications.

The authors would like to express their gratitude to the National Program for Electronics, Communications and Photonics at the King Abdulaziz City for Science and Technology for supporting this work.

A. Radar Signal Processor (RSP) Architecture

A generic, reconfigurable RSP architecture which abstracts the hardware details for the processing cards is presented in Figure 1. It is based on the VITA 41 standard which combines the Versa Module Eurocard (VME) legacy parallel standard with enhancements to support switched serial fabrics using the VMEbus Switched Serial (VXS) standard [2]. It utilises VME for control and monitoring data while utilising the VXS P0 connector for high speed Radar data. This Generic RSP architecture will be used as the context for the development of the CMF in the rest of the paper.

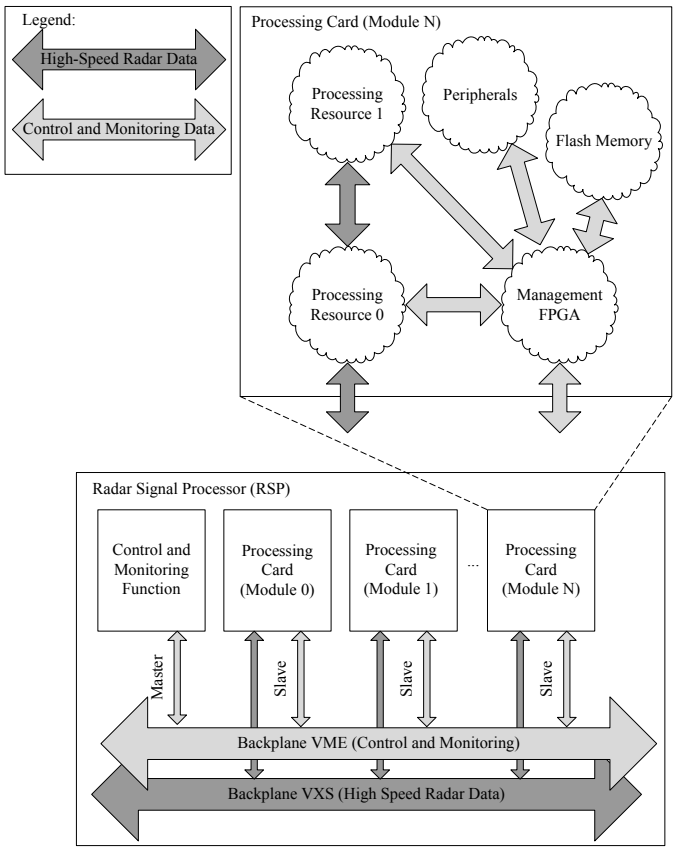

Figure 1. Generic Radar Signal Processing Architecture.

The CMF was developed as part of a complete CMS for an RSP architecture similar to the generic architecture presented here. The CMS development was performed in parallel with the development of a reconfigurable RSP architecture for a Research and Development (R&D) environment.

Two different proprietary processing cards were developed for the RSP where the first performs sampling and generation of signals and the latter performs signal processing. The architecture allows for multiple processing cards of each type to be used. The processing cards have management FPGAs that coordinate all board-level activities. A VME slave interface is implemented by the Management FPGA on the VME bus which runs over the P1 and P2 connectors on the backplane. The Management FPGA serves as an interface to all the resources on the board.

The hardware architecture for the two types of processing cards are remarkably different in terms of the number of processing resources, interconnection topology and other peripherals. The interface that is presented by the cards via the Management FPGA is the same, which allows us to abstract the processing cards for the purposes of control and monitoring in general. As a result, in-depth hardware details of the processing cards are not required to describe the features and architecture of the CMF.

B. Low-level Control and Monitoring

The Management FPGAs perform configuration of other devices, hardware monitoring, Built-in-Test (BIT), flash memory access and address translation. The VME slave interface acts as the primary interface to each processing card from the Control and Monitoring function which is the master on the bus as shown in Figure 1. All hardware registers as well as configuration and control data is exchanged with the RSP via this route.

Control and monitoring of a Radar Signal Processor with multiple cards which each contain multiple processing resources that have their own address spaces can become very complex to manage very quickly. The management of control and monitoring in an environment where hardware and firmware configuration changes are frequent, is even more challenging. An experimental system that is used in an R&D environment typically undergoes frequent changes, in order to accommodate a variety of experimental configurations. A high-level Control and Monitoring System (CMS) which uses a cooperative, dynamic scheme for real-time, system-level control and monitoring is needed for such an environment.

C. High-level Control and Monitoring

A software implementation for a high-level control and monitoring solution would provide a very flexible, configurable, reusable and user-friendly platform. A software framework which allows for Rapid Application Development (RAD) of control and monitoring applications can be made extensible and reusable through the use of object-orientation. The addition of dedicated GUI development functionality would make such an OO application framework all the more useful for developing high-level control and monitoring GUIs.

The primary benefits of OO application frameworks stem from the modularity, reusability, extensibility, and inversion of control they provide to developers. Frameworks enhance modularity by encapsulating volatile implementation details behind stable interfaces. Framework modularity helps improve software quality by localizing the impact of design and implementation changes, which reduces the effort required to understand and maintain existing software [3].

II. DEVELOPMENT PROCESS

The development process that was followed for the CMF is described below along with experiences during development.

A. Unified Process

The Unified Process was used for the purposes of the CMF development. The Unified Process is a software development process that provides a generic process framework that can be customized for a large variety of software systems [4] as shown in Figure 2.

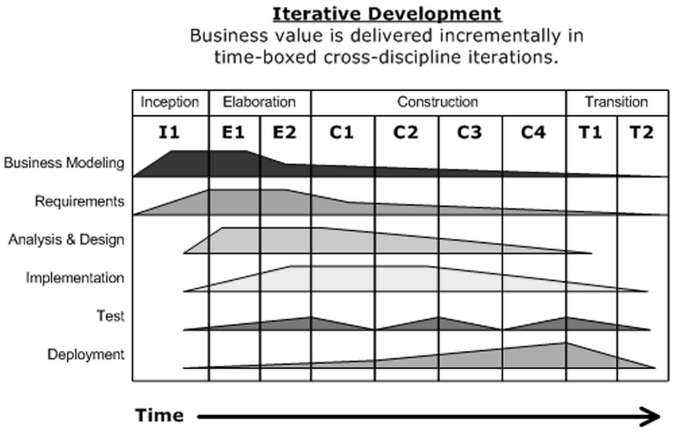

Figure 2. Workflows and Phases in the Unified Process [5].

The Unified Process is an iterative and incremental development process as shown in Figure 2 above. The Elaboration, Construction and Transition Phases are divided into a series of time box iterations. Each iteration results in an increment, which could be a release of the system with added or improved functionality compared with the previous release.

Use Cases are used to capture the functional requirements and to define the contents of the iterations. Each iteration takes a set of use cases from requirements all the way through to implementation, test and deployment.

The architecture sits at the heart of a developmental software system. An executable architecture baseline is a deliverable from the Elaboration phase. This partial implementation of the system serves to validate the architecture and is used as a foundation for the remaining development. The Unified Process uses the Unified Modeling Language (UML) when preparing models of the software system [6]. UML is an object-oriented analysis and design modeling language. The fundamental reason for using UML is communication. It allows the developers to communicate concepts to stakeholders in a clear manner and allows the developers to highlight important details.

An executable skeleton architecture was developed in the early phases of the project using software simulators running on PC systems. This approach allowed for greater user interaction at an early stage of the software development process and allowed areas of concern to be raised and resolved early on. Concepts and scenarios could also be evaluated early on; however this led to client requirements changing frequently as the development progressed. The Unified Process helped maintain control of the overall scope and change in requirements as they emerged. Progress monitoring and scheduling also benefitted due to the increased visibility to stakeholders.

UML models were used extensively during the initial phases of the CMS development. Use Case diagrams, Class diagrams and Sequence diagrams were mainly used to gather requirements and drive development.

B. Design Patterns

Design patterns describe a recurring problem in an environment and then describe the core of the solution to that problem in a way that allows it to be applied in the specific context of the problem. In object-oriented software development a design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design [7]. A number of well-known object-oriented design patterns were implemented in the CMF.

a) Singleton

The Singleton design pattern provides a mechanism for having a single, common point of access to shared objects in a framework. It also restricts the number of instances of a class in a certain context to one.

The Singleton design pattern is used extensively to provide a consistent and safe method for gaining access to shared objects in the CMF such as a message manager and shared memory maps that are used in the system. It ensures that these shared objects always exist when accessed which prevents any issues that could be caused by variations in the startup order of the various units in the system.

b) Interpreter

The Interpreter Design Pattern provides the structure for creation of abstract syntax trees that represent a specific grammar. Classes are defined for terminal and non-terminal expressions with a common abstract base-class. This approach allows expressions to be stringed together and evaluated or transformed abstractly without having to know which operators and operands are contained in them.

The Interpreter design pattern is utilised for XML Scripting functionality that is implemented in the CMF and which is discussed later in the paper. It allows operators and operands to be added easily without changing the code that parses, runs and evaluates the scripts directly.

c) Template Method

The Template Method design pattern allows a base-class to define a set of steps that need to happen in a particular order in a framework. It also allows the steps to be redefined or customized by classes that extend the framework, but the basic recipe remains the same.

The CMF uses Template Method in order to define base components that already provide a structure that needs to be adhered to in order to comply with the framework.

C. Software/Firmware Co-development

The CMF software had to be developed in parallel with the firmware and hardware for the CMS and RSP architecture. This resulted in the firmware developers being critical stakeholders of the software development process. Owing to the complexity of the CMS project, firmware, embedded software and high-level software had to closely interact in order to provide all required functionality.

The Unified Process was highly beneficial in providing the capability to handle changing requirements and allowed the software releases to be available before the firmware implementations were ready for testing. This aided the project schedule as it resulted in moving the software development off the critical path. This also resulted in the software requirements being changed in order to assist with the tasks that remained on the critical path of the project. This resulted in some issues arising in terms of availability of equipment and hardware but did bring the software and firmware development into closer proximity.

III. CMS ARCHITECTURE OVERVIEW

The Control and Monitoring System (CMS) architecture is shown in Figure 3.

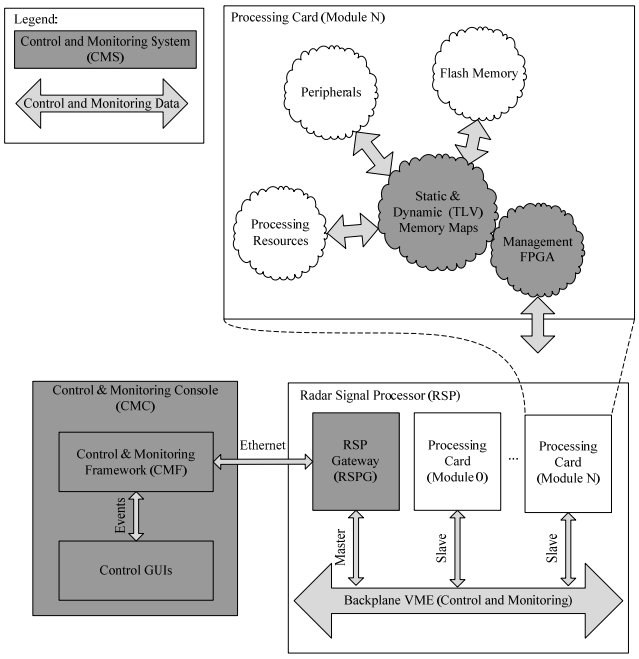

Figure 3. CMS Context Diagram.

The CMS consists of hardware, firmware and software components as discussed below.

A. Hardware

The hardware architecture for the CMS makes use of industry standards and COTS components wherever possible in order to reduce cost and increase interoperability.

The Control and Monitoring Console (CMC) consists of a COTS PC system with a ruggedized chassis for durability and with Ethernet interfaces. A keyboard and mouse are used for input and multiple monitors are used for output with the use of dual-output graphics cards.

The RSP Gateway (RSPG) uses a single-board VME computer which also supports Ethernet. This board serves as an Ethernet-to-VME bridge which translates Ethernet messages into VME bus transactions. This important function of the RSPG makes it possible to use almost any commercial PC hardware for the CMC since it alleviates the need for direct VME compatibility. The CMC is liberated to use the ubiquitous TCP/IP protocol over Ethernet which is robust and simple to use via well-documented socket APIs.

The Management FPGA uses a Xilinx [8] chipset since Xilinx is one of the leading FPGA manufacturers currently.

B. Firmware

The CMS architecture requires firmware infrastructure on the processing cards in order to manage memory maps.

1) Dynamic Memory Map

A Dynamic Memory Map (DMM) scheme is used to construct an address translation layer which is exposed by the firmware on the processing cards as shown in Figure 3. This scheme implements the Type-Length-Value (TLV) concept to create a hierarchical register structure which consists of functional groups as shown in Figure 4. Address space is automatically allocated for the groups, to form a Memory Map (MM) structure for the registers which can be accessed over the VME bus. This address space can grow dynamically when registers are added or removed from the firmware.

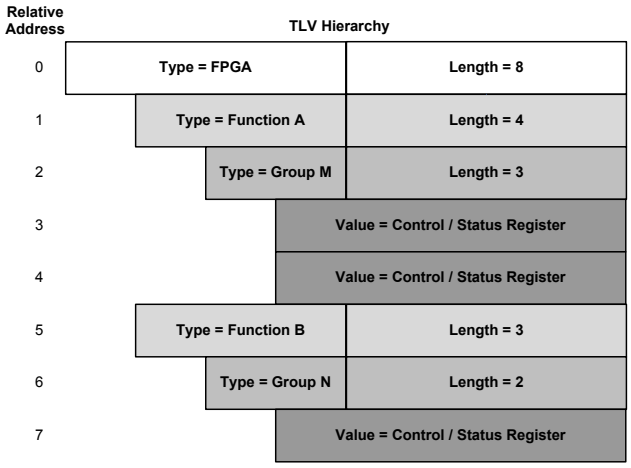

Figure 4. Firmware TLV Structure.

C. Software

The Ethernet-to-VME bridge functionality is provided by the RSPG which runs real-time software written for the vxWorks real-time operating system. The RSPG is designed to be as transparent as possible to avoid internalizing too many system details in this embedded subsystem. An explicit design decision was made to externalize as many of the system details into XML files instead of building them into the software itself. A client/server architecture is used for this interface into the RSP where the RSPG acts as a server for Ethernet requests that are sent from the high-level control and monitoring software running on the Control and Monitoring Console (CMC) as shown in Figure 3. The high-level control and monitoring software for the CMS is performed by the CMF software.

IV. CMF Architecture

The core framework consists of a set of C++ classes that provide an abstraction for asynchronous control and monitoring of proprietary processing cards and other subsystems. It also provides a set of event-driven extensions for developing interactive and configurable control and monitoring GUIs.

A. Tools and Libraries

The core framework is integrated with a number of 3rd party and other components to provide the functionality that can ultimately be used to develop reconfigurable, application-specific control software and GUIs. A description of the main components that were used follow.

1) Libxml2

Libxml2 is an XML C parser toolkit with language bindings for a variety of other languages. It implements a variety of existing markup language standards and is known to be very portable [9]. Libxml2 with C++ language bindings was used as an XML parser for the CMF.

2) Borland C++ Builder

This framework is a Rapid Application Development (RAD) environment for C/C++ development by Borland under Microsoft Windows [1]. It provides productivity tools for developing GUIs in the form of a Visual Component Library (VCL) which ships with the IDE. This framework was integrated with the CMF and provides the outer layer of GUI functionality in the form of an event-loop with visual controls that the user interacts with.

3) CodeGen

CodeGen is a code generation utility which generates TCP/IP communication code for a variety of target platforms (Windows, Linux, vxWorks) and languages (C, C++) according to a network architecture that is defined in XML. This utility was developed in-house and is used to generate code that provides an abstraction to raw TCP/IP UNIX sockets that is tailored according to high-level communication channels that are defined between subsystems. Callback functions are provided as hooks for handlers in user code when messages arrive at a subsystem. Sender function wrappers provide a single entry point for sending messages to other subsystems.

B. Memory Map Discovery and Construction

The CMF goes through a process where it discovers the processing cards present on the VME bus by querying the RSPG for the identifier (ID) of each card. After all cards have been discovered an internal memory structure is built up for each card. For each card an internal static MM is built up by loading a XML file defining the registers in the static MM. When a card has a DMM the CMF goes through a process to discover the groups of registers present in the DMM. XML files which define the register groups are pieced together for the DMM using the IDs of the groups. This creates a MM with the same structure as the registers in the DMM. In addition to identifying the hardware and firmware configuration according to IDs, the versions of the various processing cards and firmware functions are also taken into consideration.

C. Version Management

Updates in firmware functions and modification of hardware modules result in version changes that could greatly affect the operation of the system. The CMF performs version management which provides support for new versions to be added easily while maintaining support for older versions. Each processing card and firmware function stores its own version and makes this available to the CMF. This version gets read by the CMF to load the correct XML file. This allows multiple versions of the RSP hardware and firmware infrastructure to be supported concurrently once the necessary XML files have been defined.

The reconfigurability provided by using XML files provide good support for version changes in processing cards and firmware functions. During the RSP architecture development multiple hardware versions of the signal processing card co-existed in the same system at certain points in time. The CMF was developed and tested in parallel with the development and testing of firmware functions as well. The CMF was successfully used to control and monitor the RSP throughout the development even with rapidly changing configurations.

D. Layered Architecture

The whole framework is built on a set of layers as seen in Figure 5.

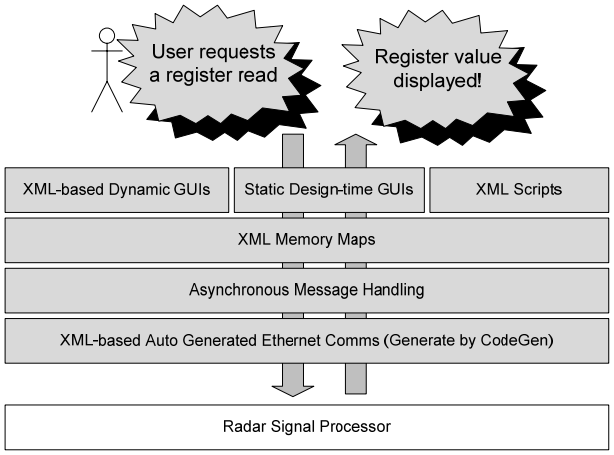

Figure 5. Layered Architecture of the CMF showing a basic user interaction.

1) XML-based Auto Generated Ethernet Comms

The C/C++ code that is generated by CodeGen creates channels between systems for sending and receiving messages. These channels get handled by different threads and communicate over different ports on the devices. Ultimately binary messages are sent over the channels between the systems using TCP or UDP packets. This forms the core of the communications layer between all of the systems.

2) Asynchronous Message Handling

On top of the communications layer there is a message manager that handles the asynchronous sending and receiving of messages. It keeps track of messages that are sent and received externally by the CMC. The message manager dispatches notification events to layers higher up in the hierarchy when responses to messages are received and also when messages time out. Message timeouts occur when a response to a request message is not received timeously according to a user-specified timeout value. All outgoing and incoming messages pass through the message manager so that it can serve as a single point of entry and exit for messages that can be controlled and monitored.

3) XML Memory Maps

Registers are given names using the XML files that get loaded for the MM. XML files are also used to assign additional properties to registers. They can be specified as either read-only or writeable for example. Registers can also be broken up into fields using the XML files. A field is used to apply a mask to the register to read or write only a subset of a register. A register can be broken up into any number of fields. To write or read a register the address of that register must be known. The MM looks up registers and fields based on the names specified by the XML files. The MM also decouples the register from the way in which the register gets accessed. Reading or writing a register on the MM results in the MM getting the address for the register and sending a request using the message manager over an interface associated with that register. When a request is made to write to a read-only register the MM rejects the request and the request fails.

4) GUIs

GUIs are used to modify and inspect the registers in the firmware. GUI controls are built on the memory map to give the user an interface into the system. The GUIs are divided into a few groups. Firstly there are purpose-built GUIs. These GUIs are created in design time and do not change during the execution of the application. These GUIs have a fixed look and feel. They are primarily used for interfaces that will not change, or that are hard to change. The second group is GUIs that adapt to changes in the system. These are created during design time but might change during run time depending on what is detected during the discovery process. Through this mechanism the CMF provides automatic reconfiguration of the GUI according to the active configuration. The last group is pure dynamic GUIs. These GUIs are only canvases for controls. The controls that need to be displayed on the GUI are defined in XML files. The CMF has a predefined set of controls that the user can use to reconfigure the GUI by modifying the GUI XML file. These controls for registers inherit the properties of the registers from the MM.

5) XML Scripts

The GUIs are not the only interface that can be used to modify and monitor registers in the system. An XML-based query language is also used. A whole sequence of commands, from reading registers, writing registers, loops and calculations can be specified in the XML file. After validation, the commands in an XML file will be executed in sequence as a high-level RSP program or script. This mechanism is used to write configuration scripts for the system and also to execute high-level tasks that are too complex to perform directly in firmware or hardware.

V. VERIFICATION

The CMF framework was utilised as part of the CMS to test and debug hardware and firmware in addition to the primary control and monitoring function during the RSP development. The CMF functionality was verified by serving as the primary user interface to the RSP for the various users and developers of the system. New versions of the CMF software was released to users and developers incrementally as new features were required and implemented.

A subsystem-level Acceptance Test Procedure (ATP) was conducted for the CMS in order to verify the functionality. In addition the CMF was again utilised as the primary user interface during a system-level ATP for the entire RSP architecture development. The ATPs were successfully concluded. The success of the ATPs can be attributed to the iterative and incremental integration and testing that was performed during development.

VI. BENEFITS AND CAVEATS

The benefits and caveats of using the CMF that have been observed are discussed next.

A. Extensibility and Reconfigurability

The CMF framework allows multiple developers to work on developing GUIs for a control software application in parallel since the framework provides a well-defined interface to do so. Firmware developers can modify, add or remove firmware functions without assistance from software developers or the need for software changes by merely modifying the required XML files.

B. Reusability

The CMF has been adapted for other projects within the organization. New contracts have also been placed which will build on the CMS using the CMF. The reconfigurability of the system also allows it to be re-used in other systems with relative ease due to the use of object-oriented methodology and configuration through external XML configuration files and scripts.

C. Disadvantages and Issues

The CMF integrates with the main event loop in the C++ Builder GUI framework in order to be able to respond to and trigger events. Issues were experienced due to the highly multithreaded architecture that was required in order to provide truly asynchronous message handling capabilities. Integration of frameworks with the event loop is known to be challenging when concurrency is involved [3].

Using application frameworks adds complexity during debugging since there is often uncertainty about whether problems reside in application code or within the framework code [3]. This problem was experienced during the development of the CMF.

Additional startup time is needed for developers to understand the fundamentals of a framework in order to utilise it properly. It is expected that total development time and quality will benefit from using the framework in the long run.

VII. CONCLUSION

An OO framework which is capable of monitoring and controlling an RSP was developed in C++. The framework was tested and delivered as part of a CMS for an RSP. The RSP is monitored by reading registers and controlled by writing registers in the firmware. The control and monitoring functionality is presented to the user using different approaches for creating GUIs for the framework. The CMF succeeds in interacting with a reconfigurable RSP by also being very reconfigurable. XML files are used in a variety of areas in the framework. It is used to give registers names and other properties. XML files are also used to specify controls on GUIs as well as provide a scripting interface for interacting with the RSP. The framework uses a layered approach in order to provide a high-level abstraction of the RSP interface.

Future work includes extending the framework to be more component based. More GUI controls can still be added to provide the user with a richer interface to the RSP. The CMF is not intended to be a generic solution for performing all types of monitoring and control, although it was developed with a large degree of reusability in mind.

REFERENCES

[1] M.I. Skolnik, Introduction to Radar Systems, International Student Edition. New York, NY, USA: McGraw-Hill, 1962, pp. 1-18.

[2] “VITA - VXS: VME SWITCHED SERIAL,” http://www.vita.com/vxs.html. Last accessed on 10 February 2011.

[3] M. E. Fayad and D. C. Schmidt, “Object-Oriented Application Frameworks,” Communications of the ACM., vol. 40, no. 10 pp. 32–38, October 1997.

[4] I. Jacobson, G. Booch, and J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999.

[5] “IBM Rational Unified Process” http://en.wikipedia.org/wiki/RUP. Last accessed on 22 February 2011.

[6] M. Fowler, and K. Scott, UML Distilled, 2nd ed. Addison-Wesley, 2000.

[7] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, 1st ed. Addison-Wesley, 1994.

[8] “Xilinx” http://www.xilinx.com/. Last accessed on 10 February 2011.

[9] “Libxml2 - The XML C parser and toolkit of Gnome,” http://xmlsoft.org/. Last accessed on 10 February 2011.

[10] “C++Builder Previous Versions” http://www.embarcadero.com/products/cbuilder/previous-versions. Last accessed on 10 February 2011.