ENHANCING THE DETECTOR CONTROL SYSTEM OF THE CMS EXPERIMENT WITH OBJECT ORIENTED MODELLING

R. Jiménez Estupiñán, A. Andronidis, T. Bawej, O. Chaze, C. Deldicque, M. Dobson, A. Dupont, D. Gigi, F. Glege, J. Hegeman, M. Janulis, L. Masetti, F. Meijers, E. Meschi, S. Morovic, C. Nunez-Barranco-Fernandez, L. Orsini, A. Petrucci, A. Racz, P. Roberts, H. Sakulin, C. Schwick, B. Stieger, S. Zaza (CERN, Geneva, Switzerland), U. Behrens (DESY, Hamburg, Germany), O. Holme (ETH Zurich, Switzerland), J. Andre, R. K. Mommsen, V. O'Dell (Fermilab, Batavia, Illinois, USA), Petr Zejdl (Fermilab, Batavia, Illinois; CERN, Geneva), G. Darlea, G. Gomez-Ceballos, C. Paus, K. Sumorok, J. Veverka (MIT, Cambridge, Massachusetts, USA), S. Erhan (UCLA, Los Angeles, California, USA), J. Branson, S. Cittolin, A. Holzner, M. Pieri (UCSD, La Jolla, California, USA)

Abstract

WinCC Open Architecture (WinCC OA) is used at CERN as the solution for many control system developments. This product models the process variables in structures known as *datapoints* and offers a custom procedural scripting language, called Control Language (CTRL). CTRL is also the language to program functionality of the native user interfaces (UI) and is used by the WinCC OA based CERN control system frameworks. CTRL does not support object oriented (OO) modelling by default. A lower level OO application programming interface (API) is provided, but requires significantly more expertise and development effort than CTRL. The Detector Control System group of the CMS experiment has developed CMSfwClass, a programming toolkit which adds OO behaviour to the *datapoints* and CTRL. CMSfwClass reduces the semantic gap between high level software design and the application domain. It increases maintainability, encapsulation, reusability and abstraction. This paper presents the details of the implementation as well as the benefits and use cases of CMSfwClass.

INTRODUCTION

The Detector Control System (DCS) applications of the CMS experiment are written with WinCC OA, mostly using the native CTRL language with the process variables being modelled in tree-like data structures, called *datapoints*. *Datapoints* are the default persistence layer and they are used by the runtime database to hold the process variables. CTRL language does not include a type definition syntax to declare and manipulate new structures. This can only be done by means of functions from the standard library or through the WinCC OA graphical editing interface. Low encapsulation and coupling between CTRL language and WinCC OA *datapoints* make data manipulation complex, compared to other languages.

The design of software, from the software engineering point of view, must be an independent process, unaware of the technology specifics [1]. During the formalization of a software design, abstract domain problems need to be transferred into algorithms and organized in data models. For that reason, the resources available in the programming platform, such as data structure definition mechanisms or programming language features, have a direct impact on the project implementation. When the semantic distance between the modelling language (e.g. UML) and the programming language is large, then the time and complexity of the translation process increases. Also other aspects of the software may be affected, like the readability of the code and maintainability of the entire software.

The CMS DCS team has attempted to close this gap by creating a development toolkit to add OO behaviour as well as code and data encapsulation down to the datapoint level. CMSfwClass toolkit revises the original WinCC OA and JCOP framework [2] device modelling concepts, enabling more abstract and powerful software architectural designs.

CMSFWCLASS TOOLKIT

The toolkit is composed of two different layers. The first layer is the back-end to add object orientation into WinCC OA. The second toolkit layer is a graphical user interface to provide easy and comprehensive access to the object oriented abstraction layer.

Figure 1: CMSfwClass toolkit modules.

Though the OO back-end has been designed to work in any WinCC OA project, the *GUI* has a certain level of integration with the CERN JCOP framework. Classes can be registered and accessed as JCOP device definitions. The *GUI* also uses CERN made libraries for syntax checking capabilities.

CMSfwClass OO Back-end Features

The OO back-end of CMSfwClass provides most of the commonly used features of modern OO programming and some custom object management features:

- Single inheritance and method overriding.
- Interface definition.

ISBN 978-3-95450-148-9

- Subtyping and interface polymorphism.
- In-memory objects (without *datapoints*).
- Object serialization in files.

The toolkit provides a default base class (CMSfwObject). This class serves as a baseline for other classes and provides several useful methods to handle objects at any level in the class hierarchy. CMSfwClass implements some custom features to express other design aspects of the model, such as specialization of objects at runtime, or composition and aggregation relations between objects. These features facilitate object management; for instance cascaded object deletion.

CMSfwClass toolkit GUI features

The CMSfwClass GUI was conceived as a computer-aided software engineering tool. The user interface guides developers during the process of creating classes and objects. The user drives the development process through different panels where attributes and methods can be introduced, while consistency and programming best-practices are assured by CMSfwClass.

The most relevant features of the GUI are the following:

- Code generation.
- Guided development process.
- Syntax checker.
- Object management.

The user interface provides code generation capabilities and live messages to remind the user how to proceed during the construction of a class. When a new class is created based on a super class, the constructor already includes code for inheriting the behaviour of the super classes. Then developers can override the default behaviour. When a new attribute is added to a class, *datapoint* structures are automatically altered to include the new element. The toolkit GUI offers the developer the possibility of creating accessor methods (getters and setters) for all class attributes. To insert method implementation details, the WinCC OA CTRL language editor is opened automatically for the scope of the method.

Apart from enforcing many of the development guidelines in the auto-generated code, CMSfwClass toolkit drives the users through different panels and pop-up messages to avoid several classic programming mistakes. For example, when adding a new attribute of class type (by composition), the toolkit warns the user about the need of updating the constructor and destructor methods.

Since CTRL language is an interpreted language, there is no compiler at our disposal. Basic syntax checking is provided in a library created at CERN. The library gives information about common defects of the code such as missing variable declarations, missing return statements and other problems that would otherwise only be discovered at runtime. It also contains some other useful functionality to extract function signature information and function location in the analysed files.

Another important aspect of the GUI is the object browser and operation interface which give a comprehensive visualization of the data structures. From the class browser, users can navigate across the class hierarchy, edit class definitions, access object collections and execute the methods of any object.

DEVICE MODELLING USING CMSFWCLASS

WinCC OA *datapoints* allow engineers to model pieces of hardware and logical entities in a hierarchy of basic data structures. The JCOP framework additionally provides utilities to register, configure and handle *datapoints* as devices [3]. CMSfwClass toolkit goes one step further and adds full encapsulation by putting together the device data structure and its behaviour in a single file. A class definition file also contains a description of how the model interacts with other classes and libraries. Classes are written in CTRL language and include the following sections:

- Header: Libraries and constant values.
- List of attributes: primitive types or class types.
- Interactions: parent class, implemented interfaces.
- Class methods.

With CMSfwClass we can model hardware entities and connections between them using classes. To do this, we need to classify components by their common features, to make a proper division of concerns

A possible device classification is in Figure 2, where the model starts with a generic channel class definition inheriting from the baseline class CMSfwObject. An object of this class implements the basic behaviour of a channel. For example, actions to switch the device ON/OFF or a method to access the status of the channel. The original behaviour of a channel baseline class can be overridden to compile with different specifications for high and low voltage channels.

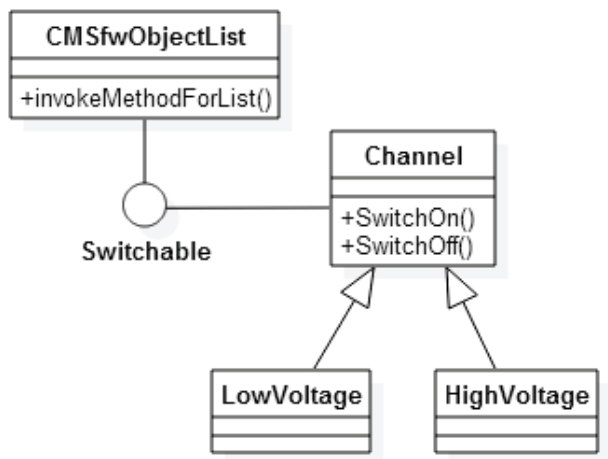

Figure 2: hardware model example.

This particular organization of entities implemented in objects can be useful when handling large, heterogeneous item collections. We can treat them in the same way since they inherit from the same base class “channel” and they implement the interface “Switchable”. We can easily go across the collection and perform the “SwitchON”

ISBN 978-3-95450-148-9

operation, abstracting from the detailed implementation required for every channel.

ARCHITECTURAL DESIGN USING CMSFWCLASS

In addition to modelling devices, developers can use CMSfwClass to model abstract domain problems. The CMS DCS team decided to refactor a small application called CMSfwScheduler, to get a more maintainable and clean implementation of the program. A comparison of the two implementations, with and without CMSfwClass, is shown in Table 1.

Table 1: Code Metrics

	Procedural	CMSfwClass
Code files	4	2
Core code lines	545	369
Auto-generated	0	329
Total	545	698

In this example, we see there is 47 % of code that has been automatically generated by CMSfwClass to perform OO consistent operations. As a result, by implementing the same program with CMSfwClass, the developer wrote 32% fewer lines of code. This comparison does not count the time spent on implementing data structures, defining naming conventions to handle data or creating user interfaces to access the data. These features are available by default in CMSfwClass and also speed-up the development process.

When control system software reaches a certain complexity, it is much easier to translate to code using OO than a procedural approach. The implicit mechanisms of OO modelling are meaningful during the software design but also in the code. CMSfwClass empowers the code by doing complex operations in fewer lines. Design patterns can also be applied in this context, providing tested, proven development paradigms.

TOOLKIT IMPLEMENTATION DETAILS

One of the goals of CMSfwClass is to help create a proper separation of concerns when modelling software. For that reason, the engine itself complies with modern software engineering principles such as modularity, encapsulation and information hiding. The toolkit is composed of different modules.

Model Data Hierarchy

The first time a class file is used in the toolkit it has to be registered. The registration of the class file creates the necessary internal structures to operate with the class, establishing the relation with other available classes and binding the library to a particular *datapoint* type. For every class in the hierarchy there will be a *datapoint* type using the class name and grouping the specific attributes for that class. For every object there will be one *datapoint* per implemented class. The tool transparently handles a parent-child representation of the classes to determine where to find a particular attribute in the hierarchy. Thus CMSfwClass maintains a clear separation of concerns between attributes and objects of different classes.

OO Syntax in CTRL Language

The implementation of OO features is subject to certain limitations of the CTRL language. Methods and object names have to fulfil the following naming convention and rules:

- A method always uses the first parameter to transfer the object name inside the function scope.
- References to objects and class names are stored in variables of type string.
- A method is uniquely identified by its signature, using the class and method name with underscore in between: *<class name>_<method name>*
- Overriding a method implies changing the signature of the function, using the class name where the method is implemented.
  - *<class-A>_<method name>*
  - *<class-B>_<method name>*
- Object names are unique.
- An object can implement many classes, and its attributes are distributed in many *datapoints* (one per class) using the following convention:
  - *CMSfwClass/<class-A>/<object>*
  - *CMSfwClass/<class-B>/<object>*
- Objects can be referenced by any of its *datapoint* names.

Subtyping and Interface Polymorphism

Object information and behaviour can be hidden using subtyping and interface polymorphism. CMSfwClass considers that the relation between a class and its super class is an inclusive specialization of the superclass (subtyping). Therefore, methods written in the super class can operate on objects of the subclass. This feature enables treatment of objects of a specialized class as if they were instances of any of its super classes. Indeed, since CMSfwClass distributes the class attributes through different *datapoints* (one structure per implemented class), we can use any of its *datapoint* names to reference the same object. In CMSfwClass this feature is called *object shapes*.

Interfaces are also used to abstract and hide object details. Interfaces can be used to define what objects must offer in different parts of the software and to limit contexts only to objects implementing a certain interface.

Single Inheritance and Method Overriding

CMSfwClass implements single inheritance. This means that a class can only extend the functionality of a single super class. This mechanism provides children classes not only with all the attributes declared in the extended class and above, but also with all the behaviours described in the class hierarchy.

There are two different ways of invoking a method. Developers can explicitly invoke a method using its

ISBN 978-3-95450-148-9

original function name, or they can delegate to CMSfwClass to perform a dynamic method invocation (using wrapper functions). Dynamic method invocation verifies the method's accessibility and looks for the most specialized version of the method. The class an object is instantiated determines the method implementation to be executed. However, the shape of the object determines the accessibility of its attributes and methods. Since there might be more than one method implementation, CMSfwClass performs a search across the hierarchy from the instantiated object class to the less specialized class.

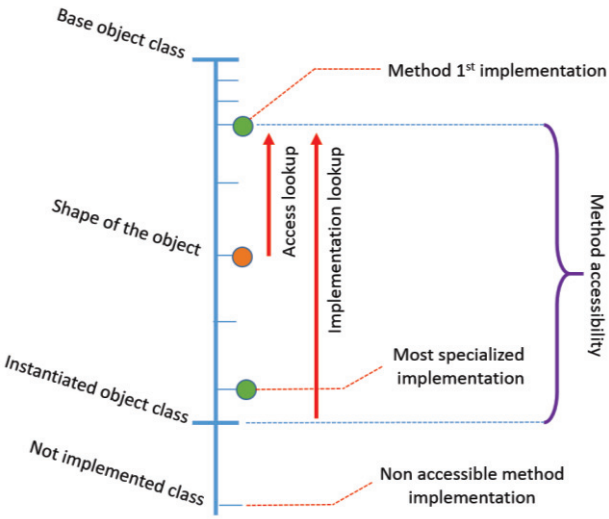

Figure 3: Method lookup

Interface Definition

CMSfwClass provides a mechanism to define sets of method signatures by class interfaces. When a class implements an interface, CMSfwClass toolkit enforces the creation of the methods defined in the interface. This mechanism adds a level of homogenization, flexibility and modularity from the point of view of software engineering. A class implementing an interface provides a first implementation of the methods. The following sub-classes extending from the parent class comply with the interface definition and therefore also implement the interface.

In-memory Objects

While in other languages, data structures are allocated in memory by default, *datapoints* are not. CMSfwClass introduces the possibility of instantiating objects in memory, eliminating unnecessary I/O operations to the persistence layer. This feature has a positive impact in terms of performance for volatile data, but adds extra complexity used for persistent data. An object can be instantiated directly in memory, or can be copied from *datapoints* to memory and vice versa. All class elements are created in a global memory variable. CMSfwClass provides a set of functions to manipulate and add event-driven messaging functionality to in-memory objects.

Object Serialization in Files

The file format to export and import *datapoint* structures in WinCC OA can be complex and hard to manipulate from a text editor. CMSfwClass includes a tool to export object collections into XML formatted files. This can be particularly useful for backing up and recovery tasks.

CMSFWCLASS STANDARD LIBRARY

As mentioned before, CMSfwClass includes a set of basic classes to help developers in the construction of the software. They form the standard CMSfwClass library, and provide with an OO version of common programming features such as:

- Polymorphic object lists and sets.
- Runnable interface for object threading.
- Event-driven object messaging.
- Design pattern solutions for object visualization.

CONCLUSION

The software described in this paper has been built, tested and used in the DCS context for the CMS experiment. The CMS DCS central team identified multiple use cases for CMSfwClass before and after releasing the first version of the toolkit. It has been proven to be an efficient environment for creating complex and high abstraction software architectures with a short development time.

Apart from the implicit benefits of using an OO oriented programming approach, CMSfwClass toolkit helps in the construction of more robust software. The real-time semantic checks and the enforced best practices in the code generation are propagated throughout all software layers.

The toolkit has helped CMS DCS team to balance the amount of time spent in different tasks of the software construction process, spending more time in designing quality architectures rather than writing code.

REFERENCES

[1] Rebecca Wirfs-Brock; Alan McKean, *Object Design: Roles, Responsibilities, and Collaborations.* (Addison-Wesley, 2003).

[2] O. Holme et al. "The JCOP Framework," ICALEPCS'05, Geneva, Switzerland, October 2005, WE2.1-6O.

[3] L. Del Caño et al. "Extending the capabilities of SCADA – Device modelling for the LHC experiments", ICALEPCS'03, Gyeongju, Korea, October 2003, TU212.

ISBN 978-3-95450-148-9