

## APPLICATIONS OF AN OO METHODOLOGY AND CASE TO A DAQ SYSTEM

C.P. Bee<sup>a</sup>, S. Eshghi<sup>b</sup>, R. Jones, S. Kolos<sup>c</sup>, C. Magherini, C. Maidantchik<sup>d</sup>, L. Mapelli<sup>e</sup>,  
G. Mornacchi<sup>f</sup>, M. Niculescu<sup>j</sup>, A. Patel<sup>g</sup>, D. Prigent, R. Spiwoks<sup>h</sup>, I. Soloviev<sup>c</sup>.

*CERN, Geneva, Switzerland*

M. Caprini<sup>i</sup>, P.Y. Duval, F. Etienne, D. Ferrato, A. Le Van Suu, Z. Qian,  
*Centre de Physique des Particules de Marseille, IN2P3, France*

I. Gaponenko, Y. Merzliakov

*Budker Institute of Nuclear Physics, Novosibirsk, Russia*

G. Ambrosini<sup>j</sup>, R. Ferrari, G. Fumagalli, G. Polesello

*Dipartimento di Fisica dell'Universita' e Sezione INFN di Pavia,  
Italy*

The RD13 project has evaluated the use of the Object Oriented Information Engineering (OOIE) method during the development of several software components connected to the DAQ system. The method is supported by a sophisticated commercial CASE tool (Object Management Workbench) and programming environment (Kappa) which covers the full life-cycle of the software including model simulation, code generation and application deployment. This paper gives an overview of the method, CASE tool, DAQ components which have been developed and we relate our experiences with the method and tool, its integration into our development environment and the spiral lifecycle it supports.

### 1 Introduction

RD13 [1] was originally approved by the CERN Detector R&D committee in April 1991 for the development of a scalable data taking system suitable to host LHC-like trigger and data acquisition (DAQ) studies. The evaluation of software engineering technology applied to DAQ systems is one of the areas for study indicated in the RD13 proposal and has been performed throughout the activity of the project. To this end we require methodologies covering the software life-cycle (from design to maintenance) supported by com-

---

a. Now at University of Zurich, Switzerland.

b. Now at University of Utrecht, The Netherlands.

c. On leave from the Petersburg Nuclear Physics Institute, St. Petersburg, Russia

d. Also with Federal University of Rio de Janeiro, Brazil.

e. Spokesperson.

f. Contact Person.

g. On leave from School of Computing and Information Systems, University of Sunderland, UK.

h. Also at the University of Dortmund, Germany.

i. On leave from the Institute of Atomic Physics, Bucharest, Romania.

j. Now at University of Bern, Switzerland.

mercially available products; we believe it is not our role, at least initially, to develop our own tools. In order to extract the maximum information from our evaluation exercises, we have applied this approach to real-life problems and produced applications which have been used by real users (i.e. at test-beam sites).

The current version of the RD13 DAQ is developed according to the techniques commonly known as Structured Analysis/ and Structured Design (SA/SD)[8] [9]. The existing software development environment covers a large proportion of the software that needs to be implemented for the DAQ system. However, there are still some areas (such as the functional code of applications) for which there is no direct support from the set of tools or are not supported over the whole life-cycle of the software. Also, the integration of the software generated from by the various tools is not always obvious. To improve this situation, we decided to evaluate an object-oriented method and CASE tool.

## **2 Overview of Object Oriented Information Engineering (OOIE) Method**

The Object Oriented Information Engineering (OOIE) Method [2] is closely related to the Ptech method originating from Associative Design Technology in the US.

The method makes a clear division between object structure (static) and object behaviour (dynamic). Analysing the object structure produces a set of diagrams showing object types and their associations, inheritance and aggregation. The object model is quite rich with support for multiple and dynamic classification. An analysis of the object's behaviour produces a set of diagrams defining sets of operations that may occur in the system in terms of the operations themselves, event types, trigger rules and control conditions.

### *2.1 Overview Object Management Workbench (OMW) CASE tool*

Object Management Workbench [3] is a commercial CASE tool sold by Intellicorp Inc. The CASE tool supports the OOIE methodology described above over the life cycle from design through to code generation and testing. The tool set includes:

*Object Diagrammer* - an interactive diagramming tool to define object classes, their relationships and attributes (see Figure 1.)

*Event Diagrammer* - an interactive diagramming tool to define the behaviour of the model in terms of events and operations triggered by events. Once defined, the behaviour can be simulated before the code generation stage (see Figure 2.)

*Business Rules Editor* - a syntax directed editor for defining rules that constrain the behaviour of the model by adding conditions to the execution of operations (see Figure 3.)

*Interface Workbench* - a direct manipulation tool for WYSIWYG creation of graphical user interfaces. All screens created with the Interface Workbench are fully portable across UNIX workstations running the X Window System and Microsoft Windows-based PCs.

*Data Linkage Editor* - creates and manages connections between graphical components and objects defined in the application.

*Scenario Manager* - is a testing and simulation tool used to execute and validate the models created in OMW. Scenarios can be stored on disk and reloaded at a later date to test new versions of the models.

*Report Browser* - allows the user to automatically generate documentation and check problems on the developing application.

The above tools are built on top of the Kappa [3] programming environment. The Kappa system is an ANSI C-based visual environment for developing and delivering applications in open systems environments. The system includes the following tools:

*ProTalk* - a high-level language of the Kappa System. It is a hybrid of a 4GL and OO language that gets compiled into ANSI C.

*ProTalk Workbench* - an interactive symbolic debugging tool for developing with ProTalk.

*C Listener* - provides a C interpreter, symbolic debugger and various other programming environment tools.

*Object Browser* - provides inspection, creation, and manipulation of objects and their attributes, and manages the organization of programs and the creation of run-times.

*Monitors* - are special-purpose Kappa objects that are attached to attributes of other objects and execute some behaviour when the attribute is accessed or modified.

*Integrated Data Access* - Kappa provides bidirectional transfer of data between relational databases and Kappa objects by dynamically generating SQL to map records and tables into Kappa objects.

Kappa applications may be fully embedded and support standard ANSI C linking conventions to integrate external code and libraries. Applications developed with OMW/Kappa can be divided into domains or *name spaces* of an application in which all object references are unique. The OMW/Kappa development tools run on Sun (SunOS and Solaris) and HP workstations. The developed applications can also be ported to IBM compatible PCs running Microsoft Windows.

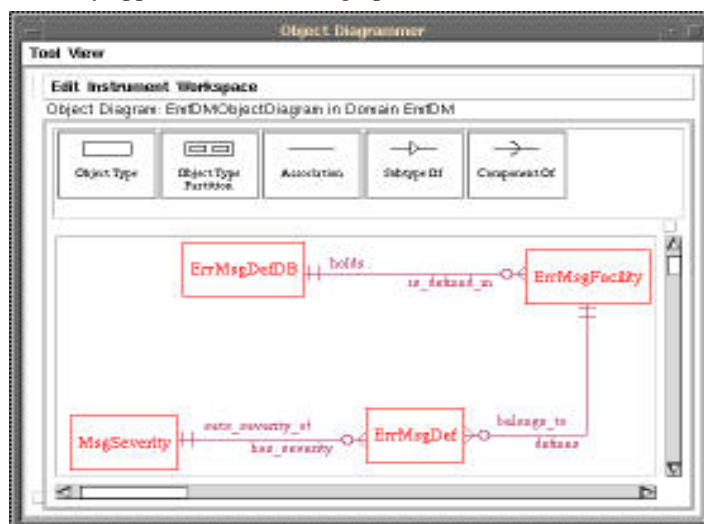
### **3 Applications developed using OMW**

We have used the OOIE method and OMW CASE tool to develop a number of applications which have become integrated components of the RD13 DAQ.

#### *3.1 Error Message Facility*

We have used OMW to develop a replacement for the existing tools for producing and maintaining unique error codes within the DAQ software. Such error codes are used by DAQ components to report their status or problems and rely on the fact that there is only one possible interpretation for each code. The error code definitions files are stored in a database implemented using OMW's object repository. Two application programs work on the database - EmfEditor and EmfTrans. EmfEditor is used by the developers off-line to interactively modify the definitions of error messages used by the RD13 DAQ system. The error codes are loaded from disk, manipulated using the graphical user interface and

then saved back to disk. A C include file containing all the error code definitions can be generated for use by applications from the graphical user interface.

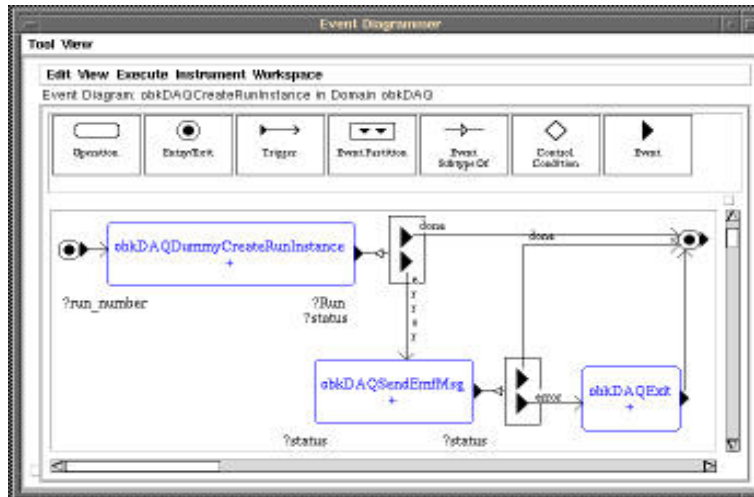


**Figure 1. OMW Object Diagram for DAQ error message database**

EmfTrans is an online server process that performs the translation from error code to textual error message. Application programs exchange network messages with EmfTrans (via ISIS [10]) to retrieve the text associated with error codes. EmfTrans loads the database of error message definitions and connects to the distributed run control system. When it receives a translation request from DAQ processes it queries the given error code in the database and returns the associated textual message. The database queries are implemented as methods in the ProTalk 4GL OO language. ProTalk has the notion of knowledge expressions for accessing information about objects in domains which can be used with backward chaining to find solutions to statements.

### 3.2 Online Volume Bookkeeping

The Online Volume Bookkeeping (OVBK) system is designed to provide an automatic log of the data recorded by the DAQ system. The data describing the run configuration, run date, archive file information and run quality are maintained in a database, which is implemented using OMW's object repository. The OVBK system also provides a graphical user interface to interactively access the database and to generate reports. This application required integration with the run control system, the run parameters database and the error message facility. It is implemented as a process on a workstation while the DAQ is taking data and writes the details of the current run to the database. Much use is made of trigger rules to decide what action should be performed according to the messages received from the DAQ system.



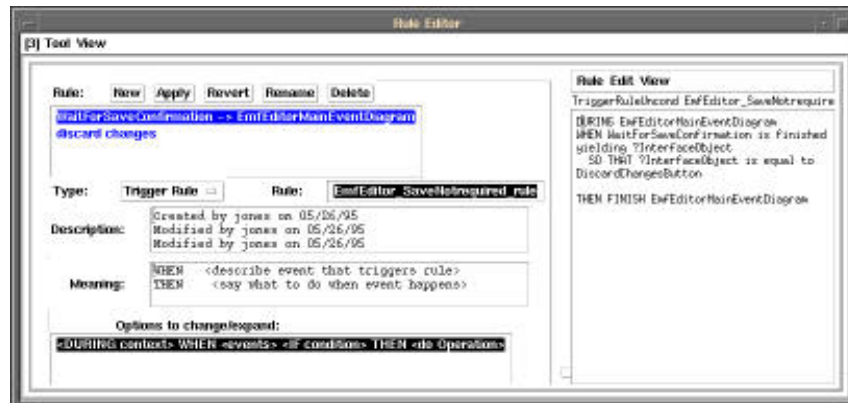
**Figure 2. OMW Event Diagram from the Online Bookkeeping System**

### 3.3 Hardware Database Editor

The hwdbEditor program allows the user to edit the contents of the hardware database used by the RD13 DAQ system. The user can modify the existing configuration, define new hardware modules or delete obsolete ones. The application uses the Object Diagrammer to make an OO equivalent of the existing database (implemented using QUID [6]) E-R model. The contents of the hardware database are loaded at run-time from QUID and instances of the object types defined on the Object Diagram are created. The user can then manipulate these instances via the GUI and the modified contents are written back to disk with QUID.

### 3.4 OMW/Itasca interface

As an evaluation, we have developed a prototype application that provides a link between the OMW CASE tool and the Itasca ODBMS [4]. The motivation for this prototype was to see if an alternative technique for object persistency could be used with OMW and also to provide a graphical schema editor for Itasca. The prototype is implemented as an OMW application that uses the Kappa library to access information stored on object diagrams and Itasca's C client library to access corresponding definitions in the database. The prototype can define classes with attributes and relationships, store OMW object instances in Itasca and reload instances from Itasca to OMW. A limitation of this technique is that the OMW object methods could not be stored in the Itasca server because they are not written in LISP.



**Figure 3. OMW Business Rule Editor**

### 3.5 Application structure

The above applications have been structured in a similar manner composed of several Kappa domains:

*DM* - data model that defines the classes used in the application. This domain consists of object structure diagrams (see Figure 1.)

*DMi* - holds the instances of the classes defined in DM domain. DMi represents the “data-base” of the application and is loaded/saved to disk using Kappa’s object persistence or the instances are created dynamically at run-time. No diagrams or code are held in this domain only object instances.

*IO* - defines the input/output operations of the application. The domain contains event diagrams and hand-written methods in C and Protalk.

*Editor* - defines the graphical user interface used to modify the contents of DMi. Screens are defined using the Interface Workbench and the Data Linkage Editor is used to link them to object instances from the DMi domain.

The source code files for the object methods and the files that define the model for the CASE tool are kept under the RD13 code management scheme and archived with CVS [5]. Applications are built using the makefiles generated for each domain by OMW.

### 3.6 Assessment of OOIE Method and OMW CASE tool

We have found the OOIE method to be clear yet sophisticated with sufficient support for static object definition. The objects structure offers such concepts as specialization, generalization, classification, composition and partition. There are constructs in the event diagrams to represent synchronization and concurrency which would help with distributed applications but it lacks other concepts for real-time work (e.g priority, exceptions etc.).

Through our prototype interface between OMW and Itasca, we found that the OOIE object model as supported by OMW and Kappa is very rich and that many features could not be implemented in Itasca (e.g. user-definable Object Identifiers, monitors and slot formulae).

There are several facilities in the method which would be useful to us but are not supported by the CASE tool, such as timer events and finite-state diagrams as an alternative means of representing object behaviour (as opposed to event diagrams). Some other features of the method are only partially supported by the CASE tool, for example, event diagrams are not re-entrant which means they cannot be used recursively and the code generated is single threaded so concurrent operations are performed sequentially and their order cannot be controlled.

The incremental development cycle supported by OMW is very practical and one of the best features of the tool. One of the most important advantages of using OMW is the clarity of the diagrams produced (i.e. event and object diagrams). Once a developer knows the notation (which is very simple when compared to other notations such as Booch [7]) he can quickly get an impression of how an application is structured even if it was developed by someone else. The diagrams are guaranteed to be up to date and complete since they are used to generate the run-time code. The developer has a global view of his application since all aspects of the model defined by the various diagrams are represented in the same programming environment.

The principle of object domains allows the developer to modularise the application into groups of closely related objects. OMW is a very open tool as shown by the number of third party software packages that we have been able to integrate with our applications.

The built-in simulator allows applications (or single event diagrams) to be tested before generating a run-time. This means the developer can remove the most obvious bugs without leaving the tool. It is also an aid to other developers who need to perform maintenance on the software by helping them understand how it works via the animated event diagrams. In general we have found that those bugs which persist beyond this stage of testing are usually related to integration with third party software (e.g. ISIS or other DAQ components) and effects of speed and space differences between interpreted and compiled code (the simulator interprets the application code while at run-time it is compiled). The report generator provides useful documentation of a model but is also helpful for debugging purposes since list of problems found on the various diagrams can be produced.

The underlying Kappa programming environment is very rich and sophisticated. We have not used all of the features provided but have found the built-in container classes and object manipulation facilities very useful. The ability to modify the object structure dynamically (e.g. add a new super-class at run-time) is powerful but obviously increases the complexity of the underlying system.

The integration of the C and ProTalk languages is quite simple with the ability to mix the two languages inside the same method. The ProTalk language is best used for object manipulation and C for accessing third party software and the underlying operating or file system. Knowledge expressions are a very powerful feature of ProTalk which can be used to construct forward or backward chaining rule sets forming the basis of expert systems. We have used knowledge expressions to perform searches on object instances in a manner that is similar to a database query language (e.g. SQL) but with the advantage that the con-

structs are part of the programming language. We would prefer to be able to use a non-proprietary language (e.g. C++) instead of ProTalk but recognise that some features of ProTalk would need to be provided by other means (e.g. the ability to dynamically change the inheritance of an object instance, access to meta-model information, knowledge expressions etc.)

The Interface Workbench offers similar facilities to tools like X-Designer [11] but has one important advantage - the Data Linkage tool. This tool allows application objects to be linked to graphical objects without programming. The relation between the application and graphical objects is maintained in both directions and automatically updated. To provide a similar behaviour with X-Designer the developer would be required to write many callback routines to move the values to and from the screen. However, we have found that we cannot integrate third party widgets so easily in the Interface Workbench as one can with X-Designer.

The internal object persistence is very useful and has been used as the basis of the error message and online bookkeeping databases. The system also has the advantage of providing limited schema evolution - that is to say that objects can be saved to disk, the schema changed (using the Object Diagrammer) and the database re-read into memory. The facilities of the object persistence cannot be compared to a real database since it has no notion of transaction support, concurrency control, versioning or distributed access but it is more akin to an OO version of QUID. We have shown that it is possible to integrate OMW with third party archive systems such as CVS. We have not yet had the opportunity to test the multi-developer extension to the tool which has been delivered as a beta-release with the latest version.

Because of the sophistication of the Kappa object environment, we can see the need for the run-time library. However, the run-time library restricts the platforms on which we can run our generated applications. This means that we cannot use OMW to develop applications which must run on our front-end processors (e.g. RAID boards running EP/LX or LynxOS).

There are a number of problems and bugs in the CASE tool that have slowed down the development of the applications but this is to be expected when using version 1.0 of any software package. In general we have found that the interpreter is the source of many problems especially if the currently loaded application becomes corrupted. The tool requires a lot of resources in terms of memory and swap space which restricts its use to the more powerful workstation configurations.

#### **4 Conclusions and future**

Our experience has been very positive and we believe that there is a great future in software engineering and CASE technology for HEP, especially for large software production. The benefits in communications between the developer and the user, with provisions for project management and support for the full life-cycle including code reliability and maintainability are tremendous. The quality and quantity of commercial CASE tools is increasing rapidly and are sure to offer the most cost-effective and organized manner for HEP projects to develop their software.

We will continue to use OMW in the near future and other applications for the DAQ are already being constructed. In general one can say that OMW and OOIE provide nearly all the features we think we would need to develop software for LHC data acquisition systems but we would prefer a tool which:

- easier to master - (OMW's sophistication can be bewildering for a beginner)
- requires less resources to run
- provides support for finite-state-machines as a means of defining object behaviour
- allows the developer to write methods in a non-proprietary OO language
- generates code that does not require a run-time library from the software tool vendor

It is unlikely that all these requirements can be satisfied by a single CASE tool. OMW is by far the most comprehensive tool we have seen to date and we would like to use it as a basis for defining the features required from some future set of tools, probably from different vendors, which could form the software development environment for LHC DAQ systems.

### **Acknowledgments**

Thanks go to Gottfried Kellner for contributing substantially to the purchase of the software tools mentioned in this paper and to ECP/PT group for providing a host site for their installation.

### **References**

- [1] IL.Mapelli et al., A scalable Data Taking System at a Test Beam for LHC, CERN/DRDC 90-64, CERN/DRDC 91-23, CERN/DRDC 92-1, CERN/DRDC 93-25, CERN/LHCC 95-47. See also the World Wide Web at URL <http://rd13doc.cern.ch/>.
- [2] OOIE - Object Oriented Methods A Foundation, James Martin and James J. Odell, Prentice Hall, ISBN 0-13-630856-2
- [3] Information on OMW and Kappa is available on WWW at <http://www.intellicorp.com/>
- [4] ITASCA Distributed Object Database Management System. See "Experience Using a Distributed Object Oriented Database for a DAQ system" in these proceedings.
- [5] B. Belliner, CVS II: II:parallelizing software development, Proceedings of the Winter 1990 USENIX Conference, 1990.
- [6] G. Bruno, Model Based Software Engineering; Chapman & Hall, 1995. ISBN 0 412 48670 9.
- [7] Booch, Grady, Object-Oriented Analysis and Design with Applications (2nd ed.), Benjamin/Cummings, Redwood City, CA, 1994.
- [8] P. T. Ward and S. J. Mellor, Structured Development for Real Time Systems, Yourdon Press, ISBN 0-917072-51-0.
- [9] D. J. Hatley and I. A. Pirbhai, Strategies for Real Time System Specification, Dorset House Publishing, ISBN 0-932633-11-0
- [10] K.P.Birman and Robert Cooper, The ISIS project: Real experience with a fault tolerant programming system. European SIGOPS Workshop, September 1990; also available as Cornell Univ. Computer Science Dept. Techn. Report TR90-1138.
- [11] Imperial Software Technology. X-Designer User Manual.