The Integration of Product Data with Workflow Management Systems Through a Common Data Model

Zsolt Kovács

A dissertation submitted in partial fulfilment of the requirements of the University of the West of England, Bristol for the degree of Doctor of Philosophy

This research programme was carried out in collaboration with CERN

Faculty of Computer Studies and Mathematics,
University of the West of England, Bristol
April 1999

Abstract

Traditionally product models, and their definitions, have been handled separately from process models and their definitions. In industry, each has been managed by database systems defined for their specific domain, e.g. Product Data Management (PDM) for product definitions and Workflow Management System (WfM) for process definitions. There is little or no overlap between these two views of systems even though product and process information interact over the complete life cycle from design to production. The integration of product and process models in a unified data model provides the means by which information could be shared across an enterprise throughout the system life cycle.

Existing PDM and WfM systems are based on rigid data models, mostly relational in nature, which have been designed for a specific phase of the software life cycle. In integrating these domains, an object oriented approach to data modeling is adopted in this thesis. A model has been developed that is sufficiently rich in semantics to cater for definitions which span the product and process domains of PDM and WfM. The model that has been developed is description-driven in nature in that it captures multiple layers of product and process definitions and it provides flexibility, reusability, schema evolution, complexity handling and versioning of data elements.

The integrating data model has been implemented in a system using component-based software: an object-oriented database, an Object Request Broker, Java user interfaces and C++ programmes. It has been tested in an application in which it is important to handle evolving definitions, both product- and process-based, over long time scales and in a distributed system which spans continents. The example studied is that of large-scale scientific detector construction for the CMS experiment at the European Centre for Particle Physics, CERN, Geneva.

In developing a data model that embodied both product and process description, design artifacts common to these two domains emerged. These ‘design patterns’ are also investigated in this thesis via the prototype system developed for this study and a discrete set of design patterns is identified for integrating product and process models.

This thesis concludes that adopting a description-driven approach to modeling, aligned with a use of suitable design patterns, can lead to an integration of PDM and WfM models which is sufficiently flexible to cope with evolving product and process definitions.

Acknowledgements

The research of this thesis was carried out during the years 1995 to 1998 at the University of the West of the England (UWE, Bristol) and at the European Centre for Nuclear Research (CERN) in the CRISTAL project.

Although it is impossible to mention all of the people who helped me to complete this thesis, I wish to express my greatest gratitude to the following friends and colleagues:

Dr. Richard McClatchey for giving me the opportunity to perform my research in his Research Centre at UWE. His trust, guidance and experience formed the driving force towards the completion of this work.

Dr. Jean-Marie Le Goff who made it possible for me to work in the exceptional research environment at CERN. This thesis was entirely formulated during my stay at CERN and without his foresight in triggering the CRISTAL project only a few of my ideas could have been realised.

Nigel Baker for giving me all of his deep understanding of distributed systems and their realisation by OMG.

Florida Estrella, Sophie Lieunard, Steve Murray, Alain Bazan, Thierry Le Flour, Guy Chevenier, Márton Zsenei, Giovanni Organtini and László Varga for working together in the CRISTAL project. Without their effort most of the ideas of this work would not have been implemented in software. Special thanks to Steve and Alain for ‘bringing me back down to earth’ when I was going too far from reality in creating the CRISTAL prototypes.

Hartmut Hillemanns and Christoph Koch for proof-reading the thesis. Hartmut read it as a physicist and played the role of an unprejudiced reader of a computer science PhD work. In doing so he gave very useful remarks to improve the understanding and readability of it. Christoph identified lots of points on the computer science background and coherence between the different chapters.

Finally, and most importantly, many thanks and kisses to my dearest ‘enemy’, Livia.

Geneva, Switzerland, April 1999
Zsolt Kovács

Table of Contents

List of Figures

1 Introduction

1.1 Preface

At a time when many companies are embracing Business Process Re-engineering (BPR) (Georgakopoulos et al., 1995) and are under pressure to reduce 'time-to-market' the ongoing management of product information from creative design through to production has become increasingly important. Traditionally design engineers have employed Product Data Management systems to coordinate and control access to documented versions of product designs. However, these systems provide control only at the collaborative design level and are seldom used beyond design. Workflow management systems, on the other hand, are employed to coordinate and support the more complex and repeatable work processes of the production environment. Most commercial workflow products cannot support the highly dynamic activities found both in the design stages of product development and in rapidly evolving workflow definitions. The integration of Product Data Management with Workflow Management could provide support for product development from initial CAD/CAM collaborative design through to the support and optimisation of production workflow activities (Kovacs et al., 1998). This is particularly important in large-scale production management systems and in scientific and engineering research environments where system design is likely to evolve over long time scales.

This thesis seeks to establish an infrastructure which enables seamless integration of Product Data Management (PDM) systems with Workflow Management (WfM) systems from the design to the production phases of the traditional system life cycle. Its primary goal is not to implement a system with full PDM and WfM functionality such as will eventually become available on the market. Rather this thesis concentrates on identifying a *design philosophy* that integrates these two management approaches and on implementing a prototype that provides core aspects of the two systems in a manner that maximises system *flexibility*. Building flexibility and expansibility into the prototype system is the key research issue of the present work. An additional objective of this work is to add to the volume of knowledge available on the subject of reusable computer system design artifacts, referred to as ‘design patterns’ in object-oriented system development.

1.2 The Research Environment

The research for this thesis was carried out at the European Centre for Particle Physics (CERN) during 1996 to 1998 in the computing section of a collaboration which is constructing a very large-scale scientific detector entitled the Compact Muon Solenoid (CMS, 1995). Historically CERN has developed ad-hoc computing solutions to support the production of its experiments as and when necessary. This has been largely due to the limited scale of experiments, to lack of resources and to lack of knowledge of industrial production management techniques. The next generation of experiments to take place at CERN at the so-called Large Hadron Collider (LHC, 1993) facility from 2005 onwards must operate over extended time scales and will be of such complexity that ad-hoc solutions to production management will no longer be sufficient.

In a scientific environment there is often no clear separation of the design and construction phases of experiment assembly (see Figure 1-1) and design evolution can often run in parallel to construction rather than being a necessary precursor. Due to the research nature of CERN, work on new experiments tends to be very iterative with new designs applied in practice as soon as possible. This contrasts with traditional industrial production lines where new product designs are normaly applied when prototyping of the product has been completed

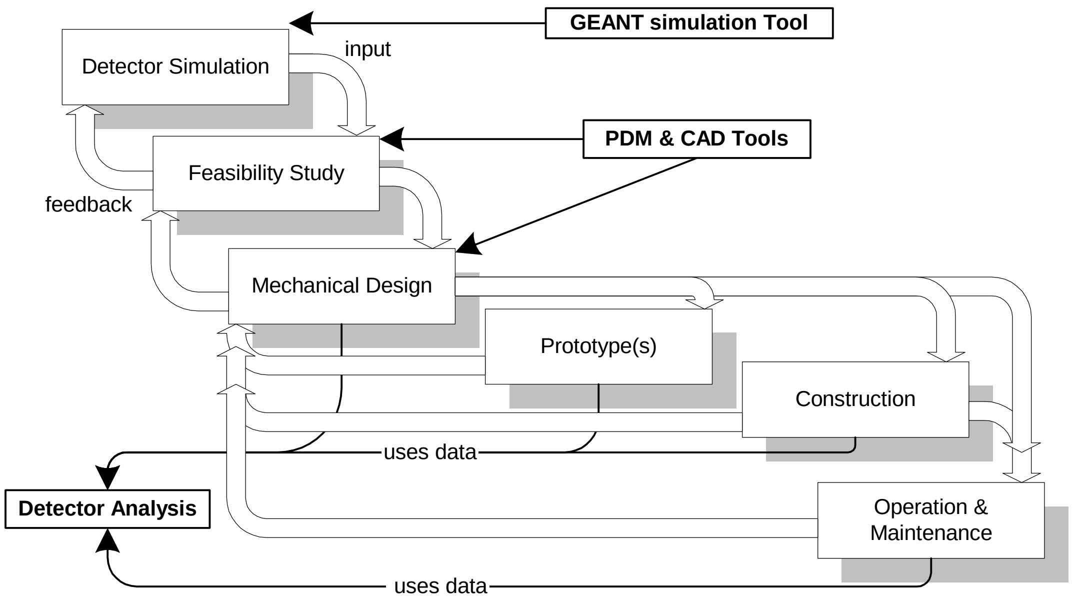


Figure 1-1. A typical product development life cycle at CERN

and the production line has been flushed. In industry design is traditionaly decoupled from production and different systems manage data separately for these two life cycle phases.

The output from the experiment construction process is a single experimental apparatus at a given point in time i.e. ‘one-of-a-kind’ production process (Hameri, A. and J. Nihtila (1998)). This apparatus typically comprises millions of detector elements (such as electronics channels, physical devices etc.) each of whose physics characteristics must be measured for the purposes of physics analysis. In addition, over time detector elements will be produced by multiple suppliers distributed worldwide and the allocation of specific detector elements to specific slots in the detector cannot be pre-specified especially since the detector is optimised for physics during construction. As the complexity of the experiment design increases the number of detector slots and candidate detector elements increases and an information management system becomes necessary to record the different design changes, to control releases of versions of designs, to manage the production, to perform the auditing of the construction process and ultimately to record the geometry of the experimental detectors. Commercial products presently provide limited support for these levels of system complexity - for example, existing PDM systems used at CERN cannot handle the error-free definition of millions of detector elements at the human level.

As a consequence of the constraints imposed by the development of systems for the LHC, which are investigated later in this thesis, it is necessary for future systems at CERN to be based on sound computing solutions and to provide additional facilities, which must be allowed to evolve over time, to cater for the specific needs of the research environment. Information systems are required to manage the vast quantities of information generated over extended time scales (1999-2005) during the design/construction life cycle of the new experimental detectors. This thesis seeks to identify appropriate computer science solutions to provide the flexibility needed for LHC experiment construction.

1.3 The Research Problem

In developing a system to cater for the production needs of CERN-based experiments the following issues must be resolved:

1. What is a suitable and convenient method for describing the CMS detector and to what extent could Product Data Management (PDM) tools be used?

2. What is a suitable and convenient method for describing the processes that must be carried out on CMS detector elements in design and production and to what extent could Workflow Management (WfM) tools be used?
3. How can the relationship between the detector element description and the detector process description be best described and what semantics can be associated with that relationship?
4. How can the representation of this relationship be designed to facilitate integration over the complete system life cycle?
5. How can such systems be designed to cater for partial specification and system evolution over time thereby allowing design evolution during system implementation?

In trying to answer these research questions this thesis will:

- Show the benefits of the complete integration of PDM and WfM descriptive approaches.
- Show that any integrated PDM/WfM solution for ‘one-of-a-kind’ systems (such as the construction of large scale detectors at CERN) should be description-driven in nature.
- Show that a reusable, component-based implementation approach is appropriate to this domain.
- Conclude that both PDM and WfM descriptive models share the same design ‘patterns’, will identify these patterns and will show that these patterns are the basis for integrating the two descriptive approaches.

As a conclusion this thesis will illustrate the extent to which the approach of integrating PDM and WfM could be applied in other phases of the product development life cycle and will indicate how technology developed using this approach could be transferred into the industrial domain.

1.4 Structure of the Thesis

Chapter 2 of this thesis reviews current related research in the fields of PDM and WfM. The roles of these disciplines are considered and issues concerning their underlying data models are introduced. Chapter 2 also considers how the standardisation process has influenced research in the disciplines of PDM and WfM and in particular the work of the Object Management Group (OMG) is highlighted.

Chapter 3 introduces the concept of so-called ‘description-driven’ systems, systems whose description has been captured in some form of a model (data and/or process models). Multi-layer architectures are considered as input to the research carried out in this thesis and the concept of descriptive ‘meta-objects’ is introduced. This chapter provides the theoretical background for the work carried out in later chapters and is based both on existing research and new concepts introduced by the author. In particular the role of patterns in developing description-driven systems is introduced in this chapter, existing patterns are studied for their role in integrating PDM with WfM, these patterns are enriched where required and new patterns are proposed to facilitate the integration.

The following chapters present the research prototype in which integration was studied. Chapter 4 describes the environment in which the research ideas expounded in this thesis were implemented and tested. The CERN environment is briefly described, its design constraints identified and the design approach used in developing the CRISTAL application software is presented in detail. Chapter 5 presents the overall data model developed for the CRISTAL application and extends this investigation by presenting a set of enriched design patterns that have emerged from the CRISTAL data modeling activities. Chapter 6 describes the implementation of the CRISTAL prototypes, the CRISTAL architecture and its underlying infrastructure and results of the early use of the prototypes. In the final Chapter 7 conclusions are drawn on the outcome of this research and pointers given as to where future research could extend the present work.

2 Current Status of PDM and WfM Systems

2.1 Introduction

In product development the management of product design information from design to implementation is paramount. This is especially true when manufacturers are having to optimise their process engineering so that product development times and ‘time-to-market’ are reduced. As the complexity of products increases - and these days composite products are being manufactured with hundreds of thousands of constituent products - so does the requirement for the use of computer-based management products. Furthermore, distributed production of products requires that product data and documents be available across local- and wide-area networks and that there is coordinated access to the product data.

Product Data Management (PDM) tools have been used for some time by manufacturing companies such as Mercedes-Benz and Ford to manage the data and documents accumulated in the design of their products. These systems are normally based on commercially available PDM systems such as MatrixOne, IBM PM or Sherpa. However, although PDM (Philpotts, 1996) systems provide good support for product documents and data particularly at the early stages of design, their use in supporting the unstructured processes inherent in product development is somewhat limited (Pikosz and Malmqvist, 1996). Also PDM systems provide few facilities for activity definition and no facilities for the enactment of production activities.

Workflow Management systems (WfM) (Georgakopoulos et al., (1995), Hsu (1995) and Schall (1996)), however, allow managers to coordinate and schedule the activities of organisations to optimise the flow of information or operations between the resources of the organisation. Commercial workflow management systems and research products are becoming available for the storage of workflow-related information and for the capture of audit trails of workflow operations. These systems seem to be appropriate tools for supporting the enactment of defined workflow operations. Workflow systems are weak at handling the dynamic evolution of process definitions which occurs during the design process and can occur even during the enactment of workflow processes. This chapter outlines current research in PDM and WfM systems and concludes that they need to be integrated to facilitate the support of the full product development life cycle in manufacturing from design through to operation.

2.2 Product Data Management

2.2.1 PDM Background

CIMdata (1998) defines Product Data Management as ‘a tool that helps engineers and others manage both data and the product development process’. PDM systems help keep track of volumes of information accumulated at stages in the overall system life cycle. PDMs integrate and manage data and documents for design (e.g. CAD/CAM diagrams, blueprints etc.), applications and information that define the products to be produced by an enterprise. Examples of products include manufactured products (e.g cars, aeroplanes, computers), projects (civil engineering projects, large assembly projects), facilities (e.g railway system, ports, warehouses), assets, plant etc. In other words PDMs help manage the information gathered during any product-related process. PDM systems can be used throughout the levels of an enterprise e.g at Director, Chief Engineer, Information Technology Manager, CAD/CAM manager or engineer and in operations, sales and marketing.

The features of PDM systems include an electronic data vault and document management, product structure management, project programme management and workflow definition management (Philpotts, 1996). PDMs have been successfully employed to control the data and documents emerging from the creative and collaborative stages of product design (e.g. CAD/CAM) where product structures tend to be hierarchical in nature and when access to documents needs to be controlled between groups of designers (using e.g. folder management). The advantages of using a PDM are well-documented elsewhere (Pikosz and Malmqvist, 1997). With a PDM the so-called ‘product breakdown structure’ (PBS, or product structure) data is centralised, versioned and can be used for tracking design in an environment which supports collaboration.

2.2.2 PDM Usage

Typically CAD/CAM systems are employed by mechanical engineers to specify the design of product components. As a consequence of this approach engineers tend to have a product-oriented view of the construction process. Conceptual design is a collaborative activity with designers checking-out and checking-in documents and diagrams of components perhaps under a policy of configuration management (Feiler, 1991). The database (and data vault) aspects of a PDM lend themselves well to this creative design process. Product breakdown (in industry often referred to as ‘Bill Of Materials’ (BOM) see CIMdata, 1998) is always strictly hierarchical in form and attributes can be assigned to each product or sub-product. Objects located in the product hierarchy can go through several stages of development so that ‘state’ can be assigned to a product and can be managed by the PDM.

PDM systems provide a change management service which can be used by engineering applications to assess, control and minimise the impact of material, product and process changes that occur in complex manufacturing life cycles. A release management service ensures that data achieves release status only after passing a pre-defined approval process, with user access to released information being based on project, password and other user-defined controls. PDMs comprise a set of integrated applications that improve the efficiency of people and processes involved in the design, production and assembly of system products.

On the one hand, the development life cycle of a large high-energy physics detector, which constitutes the research environment for this thesis, is much like any other large-scale construction activity in that it follows a design-prototype-implement cycle (see Figure 1-1 on page 14). On the other hand, the nature of experimental physics detector construction is highly dependent on state-of-the-art materials and techniques and therefore it does differ from industrial production in that it is highly iterative and consequently dynamic in execution. At the outset of the development a study is carried out (on the basis of some simulations) which assesses the feasibility of detector construction. The simulation studies provide the predicted behaviour of the detector materials under operational conditions and are thus dependent on the state of technology at the time of simulation. Similarly the mechanical design of the detector is somewhat dictated by the choice of materials as well as physics considerations. The overall performance of the detector is highly dependent on its design and therefore any changes in design need to be permeated through from conceptual design to physical construction as quickly as possible. The importance of rapidly reflecting design changes in production activities is typical of many examples of manufacturing engineering (such as micro-chip manufacture, telecommunications, aircraft manufacturing, computers) where reduction in the time between design and production is critical to reducing product development times and ‘times-to-market’.

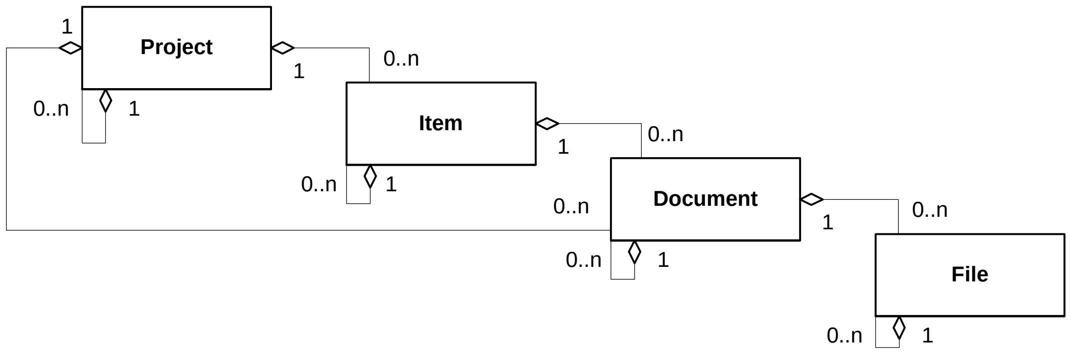


Figure 2-1. The Cadim data model, abridged.

A product breakdown structure (PBS) approach to design has been advocated by Bachy & Hameri (1995) in the design of the CERN-based LHC (LHC, 1993) accelerator. In principle PDM systems could capture not only product-descriptive data, such as the PBS, but could also capture process-descriptive data. According to Bachy and Hameri, any PDM system used for the engineering of large-scale one-of-a-kind facilities should hold the descriptions of both the PBS and the work breakdown structure (WBS) as well as the assembly breakdown structure (ABS), see upper part of Figure 2-2. The PBS tree saves information pertaining to projects, sub-projects, documents, items etc. The WBS tree holds information about the organisation of tasks (or activities) to be performed and the resources required for each task. The ABS tree holds data about how component products (and composite products) are assembled to form the overall final product. The ABS and WBS define the activities which enable the engineers to build the production line.

Current PDM offerings provide adequate support for product-related documents and some support for product data in applications where the PBS holds up to thousands of items. As yet no PDM can provide support for a PBS with millions of items, nor can PDMs cater for data related to distributed production. Furthermore, commercially available PDMs do not provide adequate support for large scale workflow process definition and execution. This is largely due to the fact that the data models on which these commercial products are based are simple tree structures which are insufficiently rich to cater for the integration of very large numbers of products and processes (Hameri, A. and J. Nihtila (1998)).

2.2.3 PDM Data Models

Traditionally, PDM products have been based on the relational data model and have largely been based on top of a relational database management system (RDBMS) such as ORACLE

(Oracle, 1998). One example of this is the Cadim (Cadim, 1998) product which is currently being used in the CEDAR (Hameri, 1996) project at CERN. Cadim is an industrial engineering document management system which will be used to handle all types of engineering data at CERN. In its introduction phase at CERN, Cadim is mainly used to manage projects and documents. The Cadim data model is based on a rigid hierarchy of data elements against which data can be stored and tracked - Projects, Items, Documents and Files - and all applications must fit into this structure (see Figure 2-1). For many small-to-medium scale projects this is suitable and the hierarchy of actual products can be captured (the familiar BOM). This hierarchy enables the design of individual product constituents and provides a structure for ordering and cost tracking. However, as the product becomes more complex in structure, such as in the construction of large-scale plant, a ‘parts explosion’ (i.e a dramatic increase in the number of database items) can take place in the PDM and data management with Cadim becomes a problem. As the complexity of the (composite) products increase it becomes simply infeasible to enter and manage products individually in a PDM based on a rigid hierarchy.

Object-oriented data models for databases (Cattell, 1994) arose as an attempt to overcome the inherent limitations of the relational model for databases. They bridge the gap in semantics between the user’s perception of a real-world application and the conceptual representation of it. Recently PDM products have begun to appear based on object-oriented data models and on Object Oriented DataBase Management Systems (OODBMS) such as Objectivity (Objectivity, 1998), one example being the MatrixOne (Matrix, 1998) product. In MatrixOne there is no pre-defined hierarchy of data elements and the developer is free to define appropriate data structures which can cater for complex products and product descriptions. To alleviate the parts explosion problem, it is possible to employ some form of *meta-data management* (see Section 3.2 on page 30) where a small number of *definitions* of products are captured in the model, representing an ‘as-designed’ view of the product, and instantiations of these definitions are used to form the overall product breakdown structure or ‘as-built’ view of the product. In this case, the BOM is not explicitly captured in the PDM, but is derived from the instantiation of the meta-data structure.

As an example consider using a PDM to manage a national electricity distribution network. The network will comprise of units (transformers, power lines, pylons etc.) repeated many times over the network. Each unit can be simply described and these descriptions catered for in a PDM. However to cater for the complete national network of units requires the storage of many hundreds of thousands of individual pylons, power lines and transformers i.e. a parts explosion. In other words capturing data about the type of unit rather than each individual unit could dramatically reduce the data management of the distribution network provided that a BOM is still derivable. This meta-data approach to designing the PDM data model is one of the main issues investigated later in this thesis.

2.3 Workflow Management

2.3.1 WfM Background

Whereas PDM systems assist in the tracking of product related data through the enterprise system life cycle, WfM systems track the execution and state of enterprise activities or processes. Hales and Lavery (1991) define workflow management software as ‘a proactive computer system which manages the flow of work among participants, according to a defined procedure consisting of a number of tasks. It co-ordinates user and system participants, together with the appropriate data resources, which may be accessible directly by the system or off-line, to achieve defined objectives by set deadlines. The coordination involves passing tasks from participant to participant in correct sequence, ensuring that all fulfil their required contributions, taking default action when necessary’.

The origins of workflow management lie in studies of enterprise process modeling and business process re-engineering, office automation (e.g billing systems) and database management and software process management. Workflow management is being applied in the applications of Computer Supported Cooperative Work (CSCW) and Groupware (Ellis et al. 1991), in Cooperative Information Systems (Papazoglou and Schlageter, 1998) and in many research prototypes (Jablonski and Bussler (1996), Wodtke et al., (1996), Mohan et al. (1995) and Sheth et al. (1996)). It has been applied to such diverse areas as software process modeling, mortgage request handling, manufacturing control and health care scheduling. Workflow management is still being researched, particularly in the fields of modeling, workflow transaction handling (Alonso et al., (1996)), work coordination and collaboration (Sheth and Kochut (1997)) and in the field of so-called *ad-hoc* workflow management where workflow process definitions can be defined ‘on-the-fly’ and where these definitions can evolve over time.

One further area that has received attention in the recent past is that of handling dynamic change within workflow systems (Ellis and Rozenberg, 1995). This is of particular relevance to the work carried out in this thesis since it is important, in the present study, for workflow execution to continue while the workflow definitions dynamically change. In other words, products can follow different versions of a workflow composition and the release of the workflow version must be possible while products are in the middle of a sequence of work-flow activities.

2.3.2 Scientific Workflow Management

Up to now, there have been relatively few examples of the application of workflow manage-ment outside the business domain. Workflow management allows the combination of a data-oriented view on applications, which is the traditional one for information systems, with a process-oriented one in which activities and their occurrences over time are modeled and supported properly. Since workflow management combines influences from a variety of dis-ciplines, including cooperative information systems, computer-supported cooperative work, groupware systems, or active databases, it has recently attracted the attention of non-business application domains. Two of these, the domain of scientific applications (in particular in the natural sciences) and that of engineering applications, seem particularly appropriate for the exploitation of workflow technology, since they involve processes in which humans and machines interact in considerable numbers, and could benefit from automation in the execu-tion of such processes (Weske, Vossen and Medeiros, 1996).

Scientific work is largely concerned with collecting, gathering and analysing large amounts of heterogeneous data. Merging data from various sources, performing analyses and carrying out sequences of tests are among the activities that could be tracked in a WfM system. What such applications have in common is the fact that the processes to be executed are frequently (sequences of) events with outcomes which can evolve as the experiment advances, so that the structure of the entire process is difficult to determine in advance. Nevertheless, model-ing, execution control, and documentation (for the purpose of reuse) are highly relevant (Wainer, J. et al., 1996).

In scientific applications, workflow execution requirements require features like:

- flexibility in structuring and modeling (versioned, open-ended, sometimes ad-hoc work-flow definition, allowing decision-making whilst a workflow is being executed);

- workflows with a complex (or nested) inner structure of individual steps (such that multi-level modeling becomes appropriate);
- workflows with a complex data structure of individual steps (e.g. the ‘outcome’ of a step is complex data);
- distribution of workflow execution;
- the treatment of failures, which can be more complex than dealing with ordinary cases like reliability and recoverability with respect to data and state;
- complete audit-trail of workflow execution;
- support for long-running activities with or without user interaction;
- application-dependent correctness criteria for executions of individual and concurrent workflows;
- integration with other systems (e.g., file managers, DBMSs, Product Data Managers, tools for analysis of data)

The design of scientific WfM systems therefore requires flexibility in the model used for definition and execution. WfM systems, like PDM systems, are based on a repository and on applications software which is driven by information stored in the repository. PDM systems hold information on product data whereas WfM systems gather information on the execution of processes or activities. It is the basic tenet of this thesis that basing both PDM products and WfM products on an integrated object model allows for the parts explosion problem to be alleviated, for flexibility to be provided and for the ability to cope with evolving workflow specifications. This statement is pursued and justified in the following chapters.

2.4 Standardisation for WfM and PDM

The Workflow Management Coalition (WfMC, 1996) is a standards body drawn from the community of Workflow Management System vendors. It has begun to identify the primitives from which any workflow management system should be built and the WfMC architecture is fast becoming a *de facto* industrial standard. The WfMC have identified a set of six primitives with which to describe flows and hence construct a work flow specification. With these primitives it is possible to model any workflow that is likely to occur.

The WfMC has produced standards on areas such as workflow Process Definition Interchange, on workflow Interoperability and workflow Client Application programming Interfaces (see WfMC, 1996) and have provided considerable input to the workflow workgroup of the OMG (Schulze et al., 1996). The OMG Business Object Management group has now proposed a Workflow Facility (OMG, 1998a) and this is currently becoming a fully fledged standard. This facility includes the definition of a workflow meta-model (*a la* Bussler, 1997), interfaces for workflow enactment, workflow monitoring and workflow audit trails. It also covers many of the aspects required by scientific workflow applications such as the nesting of workflows, support for ad-hoc workflows and workflow data security, which are not adequately covered by the WfMC.

At the same time the OMG Manufacturing Domain Task Force has been revising a proposal for standardising PDM Enablers (OMG 1998b) or services that should be provided by standard PDM products. This PDM Enablers proposal covers manufacturing-specific functions such as Engineering Change Orders, Document Management, Product Structure Definition and Configuration Management.

2.5 Integrating PDM and WfM Systems

Typically, in manufacturing systems, engineers use a PDM and production managers use Production Planning Systems and/or workflow management software. Design control and production control are separated and there is little or no cross-talk between the two. This is despite the fact that design changes need to be reflected quickly into the production environment to reduce development time. The provision of continuity from design to production through the provision of consistent product data is therefore a high priority. The integration of PDMs with workflow management software to provide consistency and continuity seems

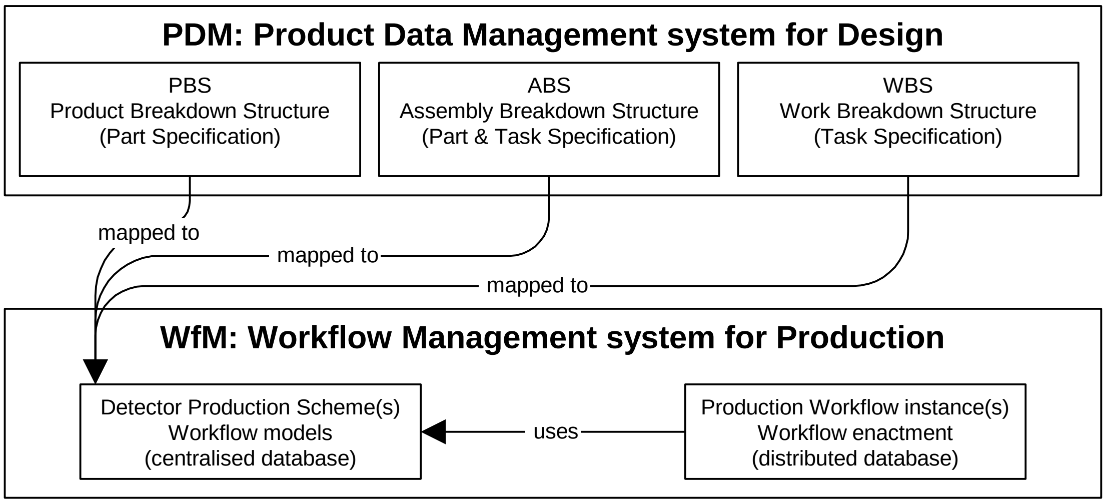

PDM: Product Data Management system for Design
PBS
Product Breakdown Structure
(Part Specification)
ABS
Assembly Breakdown Structure
(Part & Task Specification)
WBS
Work Breakdown Structure
(Task Specification)
mapped to
mapped to
mapped to
WfM: Workflow Management system for Production
Detector Production Scheme(s)
Workflow models
(centralised database)
uses
Production Workflow instance(s)
Workflow enactment
(distributed database)

Figure 2-2. The relationship between a PDM and a WfM from Design to Production

appropriate. In manufacturing systems the production line can be viewed as a collection of (versioned) workflows. The ABS and WBS hold the definitions of the production line and can be mapped onto workflows. The PDM can then manage the definitions of the product and workflow data and the Workflow software can cater for the instantiation, scheduling and enactment of those definitions (see Figure 2-2). Up to now the marriage of PDM and WfM has been proposed only for the capture of design information in manufacturing (Ramanathan, 1996) and, partially, in a civil engineering application (Stumpf, Ganeshan and Liu, 1996). Furthermore, Hameri & Nihtila (1998) conclude that current state-of-the-art PDM implementations do not support the whole product life-cycle and that new tools with an approach to support all organisational functions, including integrated product design and project management, are needed.

No research has been conducted into how the underlying data models of PDM and WfM could be integrated. A proposal for a common infrastructure for process and product models has been outlined by Manolescu and Johnson (1998) and some ‘design patterns’ have been proposed but that research is in its early stages. This thesis aims to show that adopting a description-driven approach to formalising a data model, based on design artifacts common to process and product models, will facilitate integration between product data management and workflow management, thereby providing consistency between design and production and speeding up the process of implementing design changes in a production system. The next chapter identifies how building a multi-layer architecture enables description to be captured in a system and how that description can be used to provide integration between PDM and WfM.

3 Description-Driven Systems

3.1 Introduction

For the purposes of this thesis ‘description-driven systems’ are defined as systems in which the definition of the domain-specific configuration is captured in a computer-readable form and this definition interpreted by applications in order to achieve the domain-specific goals. Description-driven systems are therefore similar to the already familiar ‘data-driven systems’ and to ‘meta-object systems’ which are coming into common parlance in computing. The expression description-driven systems is introduced to clarify common aspects of these approaches and to promote understanding of meta- concepts (Kerverhe & Gerbe, 1997 and Foote and Yoder, 1998).

In a description-driven system definitions (or descriptions) are separated from instances and managed independently to allow the definitions to be specified and to evolve asynchronously from particular instantiations (and executions) of those definitions. As a consequence a description-driven system requires computer-readable models both for definitions and for instances. These models are only loosely coupled in that coupling only takes place when instances are created or when a definition, corresponding to existing instantiations, is modified. The coupling is loose because the life cycle of each instantiation is independent from the life cycle of its corresponding definition.

Workflow management systems are one example of description-driven systems: the business process model acts as the definitions of the instantiated workflows and are managed separately from the instantiations. Up until recently PDM systems, such as Cadim (Cadim, 1998), were lacking this abstraction and there was no apparent similarity in the underlying structure of PDM systems and workflow management systems. In the recent past, however, modern PDM systems like Matrix (Matrix, 1998) have begun to follow this abstraction-based approach and to move towards supporting workflow definitions in addition to product data definitions.

Description-driven systems (sometimes referred to as meta-systems) are acknowledged to be flexible and to provide many powerful features including (see IEEE, 1996 & 1997, Crawley et al., 1997a and Baker and Le Goff (1997)):

- Reusability
- Complexity handling
- Version handling
- System evolution
- Interoperability

This chapter introduces the concept of description-driven systems, relates this to more familiar multi-layer architectures and describes how multi-layer architectures allow the above features to be realised. It then considers how description-driven systems can be implemented through the use of meta-objects and lists a set of patterns that provide the basis for realising the functionality required by description-driven systems. Finally the chapter states how the Object Management Group (OMG) is producing a so-called Meta-Object facility, which could standardise the development of future description-driven systems.

3.2 Layered Architecture of Description-Driven Systems

The concept of separating description from instantiation is well-known in computing (and particularly in the era of object-oriented computing). The ANSI IRDS standard (ANSI, 1988 see Figure 3-1), for example, followed the abstraction ideas in developing a multi-layered model in which instances are described by a model which is, in turn, described by a further model. The concept of models which describe other models has come to be known as ‘meta-models’[1] and is gaining wide acceptance in the world of object-oriented analysis and design.

One example of a system which uses a multi-layer architecture is that of a WfM system (Schulze, 1997). In WfM systems the workflow instances (such as activities or tasks) correspond to the lowest level of abstraction - the *instance layer*. In order to instantiate the workflow objects a workflow scheme is required. This scheme then describes these workflow instances and corresponds to the next layer of abstraction - the *model layer*. The information about a model is generally described as meta-data (see Maes and Nardi, 1988 and Kim,

1. According to the Oxford English Reference Dictionary ‘*Meta-*’ is “denoting position a) behind b) after or c) beyond of a higher or second order kind (like meta-language)”.

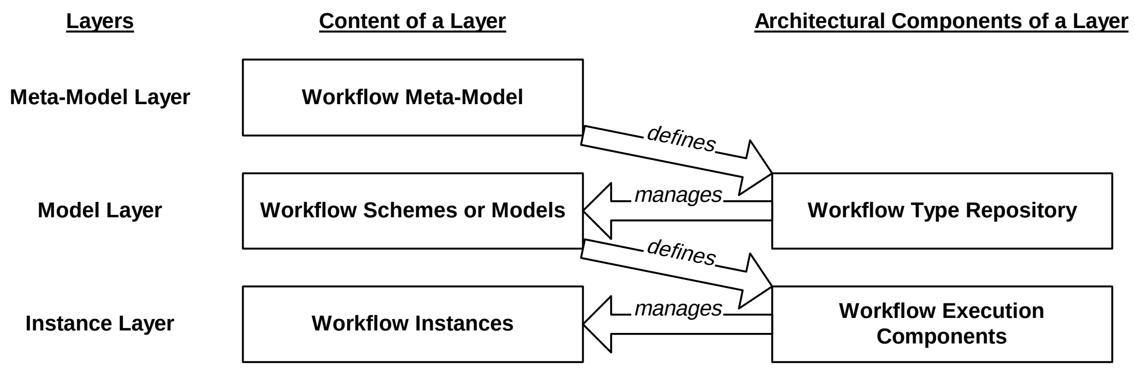


Figure 3-1. Workflow Systems in a 3-layer model architecture

1995). In order for the workflow scheme itself to be built, a further model is required to capture/hold the semantic for the generation of the workflow scheme. This model (i.e. a model describing another model) is the next layer of abstraction - the so-called *meta-model layer*. In other words a meta-model is simply an abstraction of meta-data (Schulze, 1997).

The semantics required to adequately model the information in the application domain of interest will in most cases be different. For example, the semantics for describing PDM systems (product types, product composition types etc.) is very different from those describing WfM systems (activity types, activity composition types, actor types, etc.). What is required for integration and exchange of various meta-models is a universal type language capable of describing all meta-information. The common approach is to define an abstract language which is capable of defining another language for specifying a particular meta-model, in other words meta-meta-information (c.f. Crawley et al. 1997b). In this manner it is possible to have a number of meta-model layers. The generally accepted conceptual framework for meta-modeling is based on an architecture with four layers (e.g Byrne, 1996). Figure 3-2 illustrates the four layer meta-modeling architecture adopted by the OMG and based on the ISO 11179 standard (ISO, 1998).

The *meta-meta-model layer* is the layer responsible for defining a general modeling language for specifying meta-models. This top layer is the most abstract and must have the capability of modeling any meta-model including those describing PDM (PDM 1998) and WfM systems (WFMC, 1996). It comprises the design artifacts in common to any meta-model. At the next layer down a (domain specific) meta-model is an instance of a meta-meta-model. It is the responsibility of this layer to define a language for specifying models, which is itself defined in terms of the meta-meta types (such as meta-class, meta-relationship, etc.) of the meta-meta modeling layer above. Examples from manufacturing of objects at this level

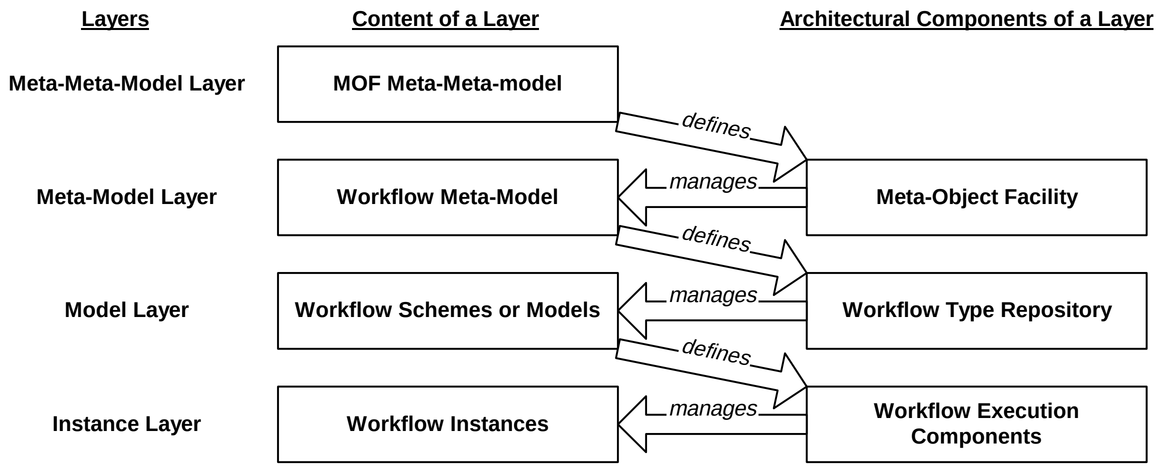


Figure 3-2. A 4-layer meta-modeling architecture

include workflow process description, nested subprocess description and product descriptions. A model at layer two is an instance of a meta-model. The primary responsibility of the model layer is to define a language that describes a particular information domain. So example objects for the manufacturing domain would be product, measurement, production schedule, composite product. At the lowest level user objects are an instance of a model and describe a specific information and application domain.

3.3 Features of Description-Driven Systems

The desirable features of description-driven systems, outlined in the introduction to this chapter, can be realised through the adoption of a flexible multi-layered architecture. This section examines each feature in turn and explains how a multi-layer architecture facilitates those features.

- *Reusability.* It is a natural consequence of separating definition from instantiation in a system that reusability is promoted. Each definition can be instantiated many times and therefore reused for multiple applications. For example, a single activity definition can be captured in a workflow management system and can be used for many workflow process specifications.
- *Complexity handling (scalability).* As systems grow in complexity it becomes increasingly necessary to capture descriptions of system elements rather than capturing detail associated with each individual instantiation of an element. Scalability can therefore be eased, and a parts explosion (see Section 2.2.3 on page 22) avoided, if descriptive information is held both at the model and meta-model layers of a multi-layer architecture and,

in addition, if information is captured about the mechanism for the instantiation of objects at a particular level. In a multi-layer architecture, as abstraction from instance to model to meta-model is followed, there are fewer data and types to manage at each layer but more semantics must be specified so that system complexity and flexibility can be simultaneously catered for. These semantics are always provided at the next higher (or descriptive) layer of abstraction. As an example of complexity handling consider the difference between describing the details of every single car of a given model produced by a company and describing the generic details of a model type. Each single instance of a car is derived from a given model type - description should be handled at the type level and details, such as the chassis number, specified only when required for a specific car instance.

- *Version handling*. It is natural for systems to change over time - new elements are specified, existing elements are amended and some are deleted. Element descriptions can also be subject to change over time. Separating description from instantiation allows new versions of elements (or element descriptions) to coexist with older versions that have been previously instantiated. For example, car models change over time and their production processes may need to be revisited as a consequence. Cars of different model versions must be handled over time and coexist with other cars of differing model versions. Separating details of model types from details of single cars allows the model type versions to take place asynchronously with the production of single cars.
- *System evolution*. When descriptions move from one version to the next the underlying system should cater with this evolution. However, existing production management systems, as used in industry, cannot cater for this. In the car example, it is not possible for a single production line to evolve *while* production is taking place. Rather the production line is flushed of cars following a particular model version before the production line is changed to reflect the requirements of the new model version. Production is therefore not continuous in nature and design changes take time to be rolled forward into production. This thesis shows that in capturing description separate from instantiation, using a multi-layer architecture, it is possible for system evolution to be catered for while production is underway and therefore to provide continuity in the production process and for design changes to be reflected quickly into production.

- *Interoperability.* A fundamental requirement in making two distributed systems interoperate is that their software components can communicate and exchange data. In order to interoperate and to adapt to reconfigurations and versions, large scale systems should become ‘self describing’. It is desirable for systems to be able to retain knowledge about their dynamic structure and for this knowledge to be available to the rest of the distributed infrastructure through the way that the system is plugged together. This is absolutely critical and necessary for the next generation of distributed systems to be able to cope with size and complexity explosions. A stronger aspect of interoperability is that distributed systems and components to be integrated should have common ways of handling and dealing with system objects such as events, security, systems management, transactions and faults. Software components must be able to plug into these common distributed services and facilities.

3.4 Implementing Description-Driven Systems

The concept of meta-data is not new - these ideas have been investigated in many domains and various technologies have been used as the implementation vehicle. A historical application of the use of meta-data is in database management systems where a schema provides a representation of the structure, constraints and use of data within the database. Relational database systems have been used to hold meta-data where data in tables describe other tables - for example the data dictionary tables of a relational database management system. Recently it has become clear that object-based systems provide greater expressivity, reusability and flexibility in the construction of complex computer systems (Jacobson, 1994) than previous systems. Object-oriented systems provide the mechanisms for the capture of system description at a high level of abstraction - descriptive objects themselves have state and methods - and are therefore suitable for building description-driven systems. When implementing a description-driven system based on objects, the descriptive element, which holds information about another object is called a ‘meta-object’.

One area of recent interest in systems design is that of ‘patterns’ (Alexander et al. 1977, Gamma et al. 1995). A pattern names, abstracts, and identifies the key aspects of a common structure that make it useful for creating a reusable object-oriented architecture. Foote and Yoder (1998) have advocated the development of patterns for meta-data and active object models. It is one of the objectives of this thesis in constructing one instance of a description-driven system, to identify where existing patterns may be employed, to identify where patterns are missing and to propose enrichment of existing patterns. The following sections introduce meta-objects and patterns in the context of a multi-layered architecture.

3.4.1 Meta Objects

For the purpose of this thesis a meta-object is defined an object which manages the description of another object. In other words meta-objects manage the meta-data required to implement description-driven systems. The ‘meta-’ prefix is used in the same manner, as it was used for meta-models, i.e. it describes the connection between objects of different layers of abstraction in description-driven systems. It is also important to emphasise that the usage of the term meta-object denotes that the system not only ‘stores’ the descriptive information but also manages it (i.e. it has data, methods and state).

3.4.2 Patterns

The concepts of patterns in object-oriented analysis and design emerged from the idea of a ‘pattern language’ or set of patterns, where each pattern describes how to solve a particular kind of problem. This idea was originally expounded in architecture by Christopher Alexander (Alexander et al., 1977). The pattern identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities. Each pattern focuses on a particular object-oriented modelling problem. It describes when it applies, whether it can be applied in view of other constraints, and the consequences and trade-offs of its use.

Patterns are a subject of intense research in computer science and an area which is rapidly maturing. Some design patterns e.g Composite and Iterator (Gamma et al., 1995) have been well-specified and are in general use. For example, Gamma defines the Composite pattern as “Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly”. According to Gamma,

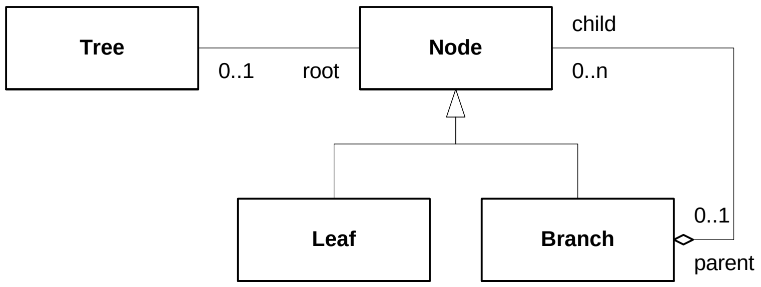


Figure 3-3. Complex Tree or Composite pattern

the Iterator pattern “provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation". Blaha and Premerlani (1998), have extended the OMT notation to help specify patterns. A number of patterns have been added to the OMT language to provide "a higher level of building blocks for models than the base primitives of class, association and generalisation". They also introduce cyclicity into the composite pattern. Of particular interest to the subject of this thesis are the Graph, Item Description and Homomorphism patterns of Blaha and Premerlani. These patterns are described in the following sections and are revisited and enriched in a later chapter of this thesis.

3.4.2.1 Item Description Pattern

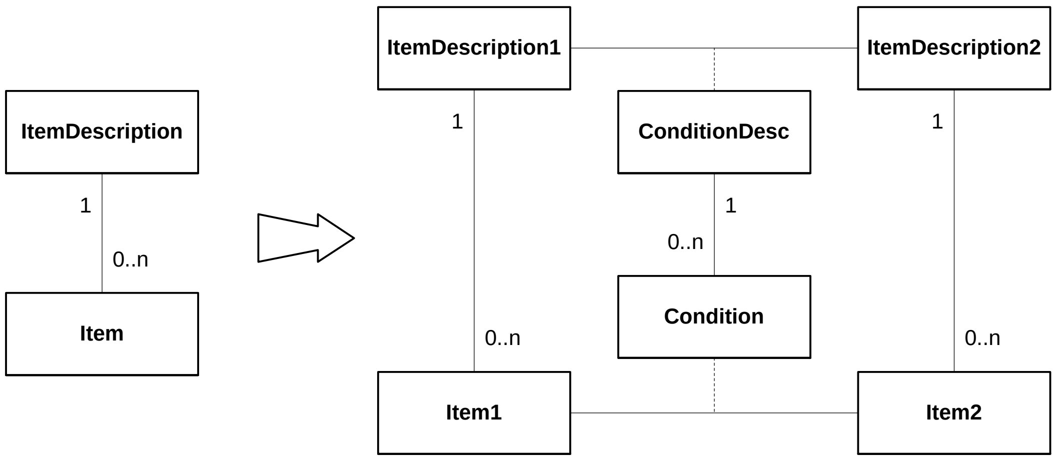


Figure 3-4. Item Description and Homomorphism pattern

Coad's (1992) Item Description pattern shows the association between descriptions and instances. In principle this pattern is the manifestation of the relationship between meta-objects and objects. Consequently this pattern describes consecutive layers of description-driven systems (see Section 3.2 and Section 3.4.1). The association between Items and Item-Descriptions can be an aggregate and support link attributes and qualifiers. In the car example of Section 3.3, individual cars (of a particular model) are Items which are built according to a single car model description. In other words, the association between car and model holds sufficient semantics for a particular instance of a car to be built according to a model definition. This mechanism is essential to the separation of instantiation from definition, as required by the multi-layer architecture of description-driven systems where semantics are required for the instantiation of Items from a ItemDescriptions. This pattern is heavily used in the data model described later in this thesis.

3.4.2.2 Homomorphism Pattern

Figure 3-4 also shows the Homomorphism pattern expounded in Rumbaugh et al. (1991). This figure shows two ItemDescriptions are themselves related so an association is defined between them. As a consequence of the Item Description pattern and the fact that semantics have been added to the association between ItemDescriptions, there will necessarily be semantics attached to the association of one (instantiated) Item to another (instantiated) Item.

According to Rumbaugh et al. (1991) "Homomorphisms are most likely to occur for complex applications that deal with meta-data". Section 3.4.2.1 stated that the Item Description pattern expresses the connection between layers in a multi-layer architecture. Since there are relationships between elements in each layer (e.g. relationship between ItemDescriptions) it is natural that the Homomorphism pattern appears between layers. The Homomorphism pattern is therefore fundamental to description-driven systems.

As an example of the use of this pattern, consider the two following Item Descriptions: the familiar car and car model and a production process and production process model. In this example, there are many instantiations of cars of a particular model and production activities of a particular process model. Also an association can be specified between a car model and a production process model - that is, the information which is specific to the execution of a production process model on a specific car model. When an instantiation of the production process is performed on a particular car, details such as the operational conditions must be specified. These operational conditions may be derived from the information on the association between car model and production process model. That is, the semantics of the association between the instantiated Items can be derived from the semantics on the association between the corresponding Item Descriptions.

3.4.2.3 Version Pattern

In description-driven systems it is important to keep track of versions of definitions and instantiations of these definitions. Figure 3-5 proposes a Version pattern that can facilitate individual and collective versioning. This pattern provides the functionality of both the CheckIn/CheckOut Model and Composition Models of configuration management (see Feiler (1991)). In this pattern each VersionedObject manages a set of individual versions of itself, each instance having a versionId and being referred to as a VersionedObjectProperty. Note that, in principle, a VersionedObject and a VersionedObjectProperty make up one object. Properties are separated from attributes in order to distinguish between meta-object data which is either versioned or not versioned respectively. Changing an object’s attributes does not version the object whereas changing an object’s properties will version it. In addition to handling versioning of an individual object, this pattern allows for versioning of a collected set of objects, called a Release. This is achieved by defining a class of objects called ReleaseManagers which are specialisations of VersionedObjects. ReleaseManagers are versioned and each version, the ReleaseManagerProperty class, manages a collection of versioned objects.

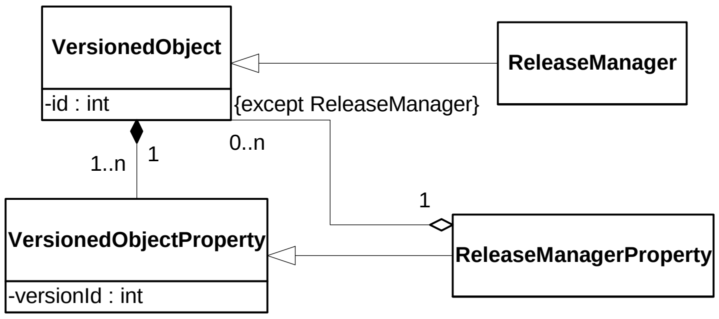


Figure 3-5. Version pattern

The ReleaseManager maintains a list of added or removed objects in a release. Static versioning is therefore handled by this pattern. The propagation of changes in a release to dependent objects (i.e the notification of changes to dependent objects and the nature of the dependencies) is described in the following section.

This Version pattern has, at the time of writing, not been identified by the patterns community. Work is in progress in identifying a so-called ‘History’ pattern (Johnson & Oakes, 1999), however identification of the Version pattern is an unique contribution of this thesis.

3.4.2.4 Publisher/Subscriber Pattern

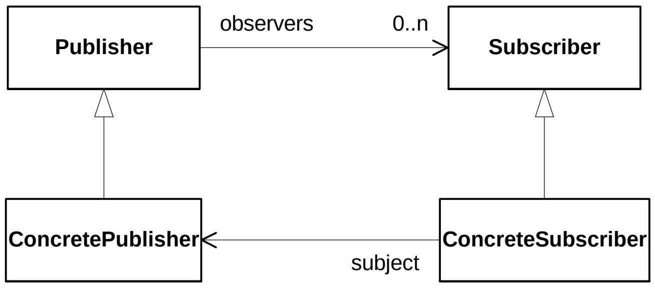


Figure 3-6. Publisher/Subscriber pattern

To facilitate dynamic version management, which cannot be handled by the Version pattern alone, use can be made of Gamma et al’s (Gamma et al., 1995) Observer pattern, otherwise referred to as the Publish/Subscribe pattern. Figure 3-6 shows this pattern. In this pattern a publisher Item (or meta-object) sends out notifications which will reach all subscriber Items without the Publisher knowing who the Subscribers are and how many Subscribers there are. (In UML (Fowler & Scott, 1997) this can be represented by a directed association as shown by the arrow between Publisher and Subscriber in Figure 3-6). This pattern is useful in handling versions of meta-objects when there are dependencies between the meta-objects but they are not tightly coupled, as discussed later in this chapter.

3.4.2.5 Graph Pattern

Graphs can be directed (cyclic or acyclic) or undirected and simple or complex in nature. They are sets of nodes which can be either leaf or branch nodes. The general form of a graph is shown in Figure 3-7 where nodes are linked to other nodes. One example of graphs is the graph/relationship service of the Object Management Group's OMA (OMG, 1992b).

In an undirected graph an edge connects any two nodes, whereas in a directed graph an edge

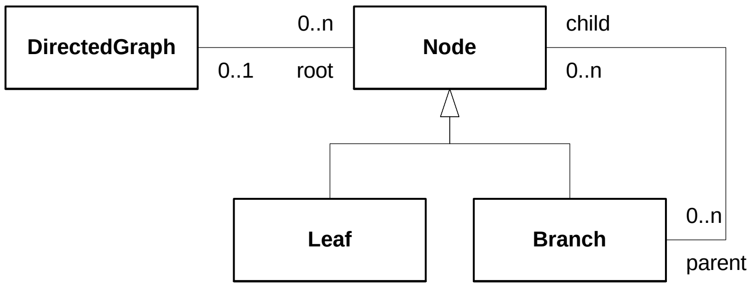


Figure 3-7. Complex Directed Graph pattern

connects a source node to a sink node. In addition, a directed graph can have nodes with any number of edges. Complex graphs make a distinction between branch and leaf nodes, whereas simple graphs do not. The example quoted by Blaha and Premerlani to describe complex directed graphs is that of the Unix file structure: files are either data files or directory files and a directory file contains named files which are identified by a filename that is unique in the context of a directory file. In the Unix file system a file can belong to multiple directories via symbolic links and a file may have a different name in each directory where it is referenced - this means the structure is a graph. All files have a parent directory except the root file as shown in Figure 3-8 (from Blaha & Premerlani, 1998). The graph is complex since distinction is drawn between datafiles and directory files - datafiles being leaf nodes and directory files being branch nodes.

In an acyclic graph, when the graph has been traversed repetitively from parent to child nodes, there are no instances where a traversal leads to a node being a child of itself. Cyclic graphs can allow this form of recursion. Therefore the complex directed graph of the Unix file system example is acyclic in nature, since a directory file cannot contain a reference to itself at any level.

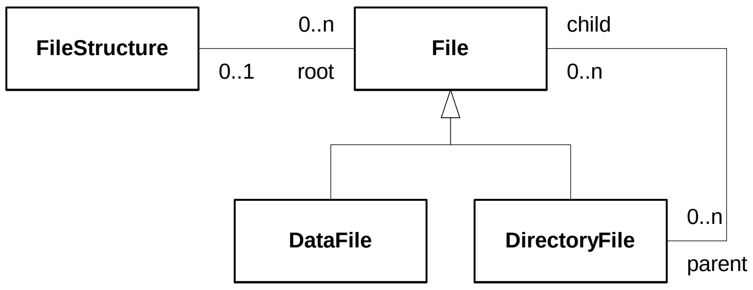


Figure 3-8. File directory as an example Graph pattern

The complex Directed Acyclic Graph pattern of Blaha and Premerlani (1998) does not allow semantics to be added to the association between nodes as branches (see Figure 3-7), consequently there is no way of identifying, and associating attributes or methods to, a particular instance of the link. Later in this thesis this Directed Acyclic Graph pattern is enriched to cater for this functionality and it is shown that this provides complexity handling in large systems.

3.4.3 Patterns and Frameworks

Elsewhere patterns have been discussed in the context of so-called frameworks (Johnson and Foote, 1988), which are defined as sets of reusable and customisable classes and software components for specific application domains. PDMs and WfMs constitute two application domains in which frameworks can be defined. The frameworks used in building PDMs and WfMs may have some software components in common. These software components can be built using patterns. Consequently using the language/patterns of multi-layer architectures, frameworks are simply the software components which result from the meta-model, model and instance layers for a specific application domain i.e an instance of a complete description-driven system.

Work in the area of patterns (Coad et al., 1995, Foote and Yoder, 1998 and Roberts and Johnson, 1998) and frameworks (Baumer et al., 1997, Devos and Tilman, 1998 and Riehle and Gross, 1998) is directly relevant to the ideas expounded in this thesis. Foote and Yoder (1998) have applied the concepts of pattern representations to the domain of data description. They conclude that candidate patterns are required to describe meta-data structures and their inter-relationships. Patterns are thus needed in object-oriented modeling to describe meta-model to build meta-objects. Similar conclusions are being drawn by Devos & Tilman (1998) in the field of design frameworks, where the framework behaviour is driven by repository-based descriptions and where descriptions of an organisation’s business operation is separated from the business application.

CORBA 2 is an example of a framework to build general distributed software applications using patterns. The ORB is composed of classes and components, whereas CORBA Services are implementation of patterns (R. Malveau & T. J. Mowbray (1997)).

3.5 Providing Description-Driven System Features Using Meta-Objects

3.5.1 Handling Complexity

In Section 3.3 of this chapter, it was stated that scalability can be eased, and a products explosion avoided, if descriptive information is held both at the model and meta-model layers of a multi-layer architecture. The Item Description pattern combined with the Directed Acyclic Graph pattern provides the mechanism by which this can be achieved.

Figure 3-9 shows a combination of the Item Description and Directed Acyclic Graph patterns. The combination of the patterns is established by decomposing an ItemDescription into its constituent ItemDescriptions. In other words, an ItemDescription can be either elementary or composite in nature and therefore some ItemDescriptions can be made up of other ItemDescriptions. Consider the car and car model example of earlier. A particular description of a car model is composed of other descriptions: e.g descriptions of the engine, the chassis, the drive-system (front-axle system, rear-axle system, wheels, tyres etc.). Some of these descriptions are elementary e.g wheels and some composite e.g drive-system. The associa-

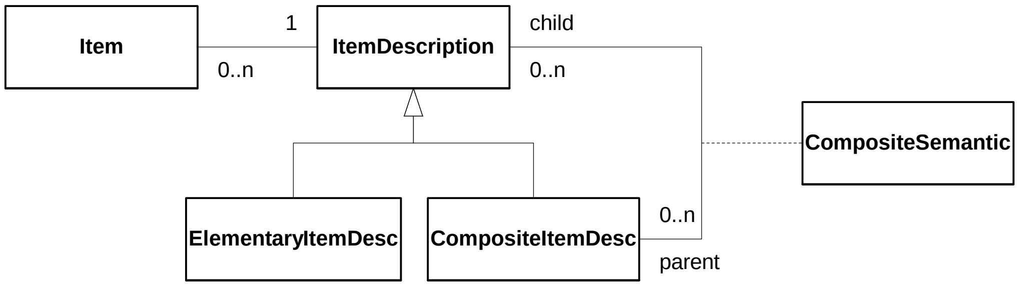


Figure 3-9. Combination of the Item Description and Directed Acyclic Graph patterns

tion between a CompositeItemDescription and its children will hold semantics such as the number of constituent descriptions of a common type (e.g 4 wheels of 1 wheel description).

However, as stated earlier, a simple combination of the Item Description and Directed Acyclic Graph patterns as described in Blaha and Premerlani (1998) does not enable the identification of a particular constituent ItemDescription within its CompositeItemDescription. In the example, it is not possible to determine which wheel is located at which wheel position.

Consequently the combined patterns require enrichment by the introduction of another meta-object which captures the membership of an ItemDescription within its CompositeItemDescription(s). Figure 3-10 shows the Enriched Directed Acyclic Graph and Item Description combined pattern. An ItemDescription can be part of many different CompositeItemDescriptions. For example, one wheel description could be employed in both the front-axle system and the rear-axle system. One instance of the CompositeMember meta-object will hold the full semantics of the membership of a particular ItemDescription in a single CompositeItemDescription. In other words, it is now possible to determine which wheel is located in which axle system and in which location in that axle system.

When a particular ItemDescription is instantiated into an Item the composition of that Item

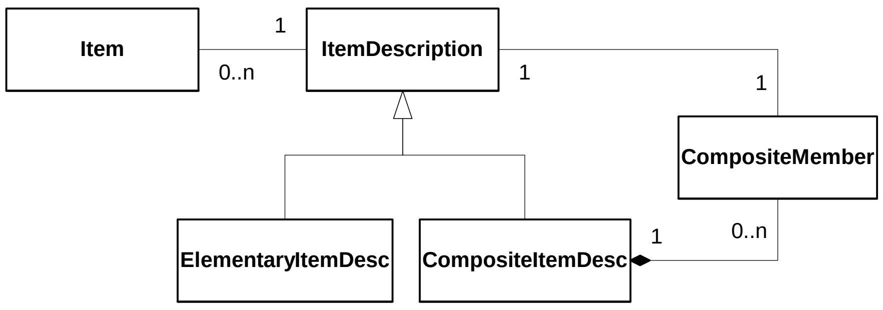


Figure 3-10. Enriched Directed Acyclic Graph pattern

is determined by traversing the graph of its ItemDescription. The result will be a hierarchy of Items organised as a tree in which each node is of a particular ItemDescription. In the car example the car is made up of a chassis, an engine, a drive-system (comprising front- and rear-axle systems each of which is composed of 2 wheels etc.). The tree is as deep as there are layers in the directed acyclic graph and each composite node will have a number of constituent nodes equal to the number of CompositeMember meta-objects in the ItemDescription corresponding to that node (see Figure 3-11). Note that when an instance of a CompositeItemDescription is deleted its corresponding CompositeMember is also deleted.

The complexity of the overall model of Items is therefore handled through the reuse of ItemDescriptions. The reuse can, in addition, take place at any point in the traversal of the directed graph so long as the graph is acyclic. For the car example, the number of Items and the number of levels of compositeness is not great and complexity handling is not a major issue. As either the complexity of the Item and the number of levels of composition increases the role of CompositeMember meta-objects becomes essential. Later in this thesis the Enriched Directed Acyclic Graph and Item Description patterns are used to manage the complexity inherent in the construction of a large high energy physics experiment.

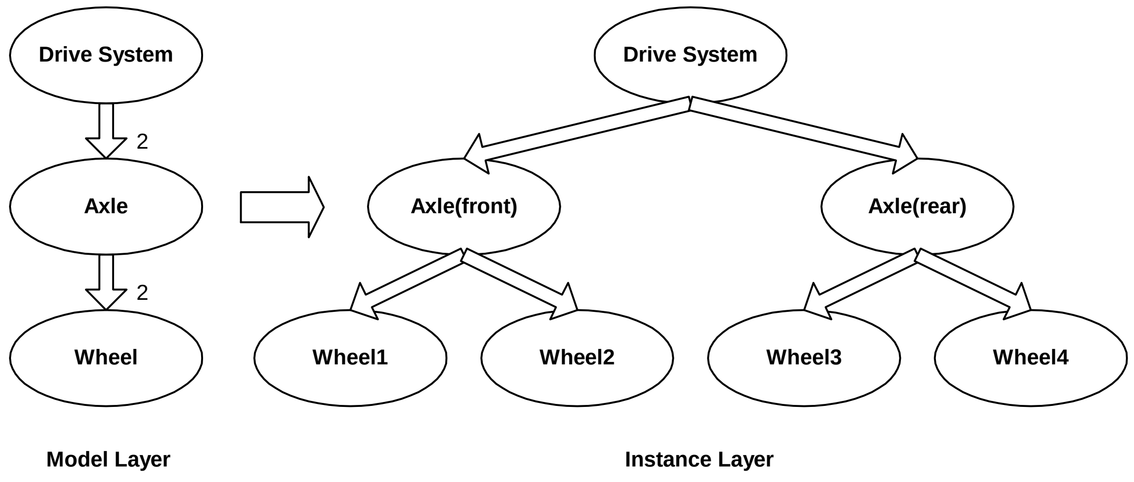


Figure 3-11. Car example for Graph handling complexity

3.5.2 Integrating Product & Process Models

Having discussed the role of a directed acyclic graph to describe Items and their constituents in the previous section, it is now proposed that any product or any process can be modeled in terms of an Enriched Directed Acyclic Graph pattern combined with an Item Description pattern.

In manufacturing, models are used to support the design life cycle of a particular product (see for example Lee, Sause and Hong, 1998 and Haugen, 1998). Products can evolve over time, their designs may change or the production process may be improved. In Chapter 2 it was stated that PDM systems have been employed to manage product data in the design life cycle. PDM systems traditionally employ hierarchies to capture product composition (so-called ‘Bill Of Materials’, BOM) and therefore, as the complexity of the product grows PDM systems suffer from a parts explosion. Basing a PDM model on an Enriched Directed Acyclic Graph pattern combined with an Item Description pattern, handles the parts explosion. The consequence is that the BOM is only available once the product composition tree has been generated by traversal of the complete graph structure (as shown in Figure 3-11).

Products are subject to many processes in the manufacturing life cycle such as design processes, assembly processes, test processes, maintenance processes etc. Each of these processes can be complex and composite in nature. Ideally the description of these processes should be captured in a model and instances of these processes managed in some repository. One example of a process management system is a WfM, which, as stated earlier, can be described as a description-driven system. Process models to support systems such as WfMs must cope with process composition, process sequence, parallelism of processes and synchronisation of processes. Basing a WfM model on an Enriched Directed Acyclic Graph patterns combined with an Item Description pattern supports processes of arbitrary complexity including composition and sub-process reuse. Furthermore, using the CompositeMember meta-objects of the Enriched Directed Acyclic Graph pattern allows the capture of process sequence, parallelism and synchronisation.

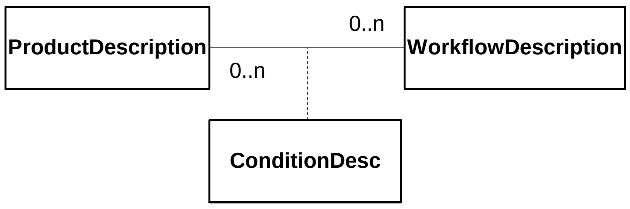


Figure 3-12. Product, Workflow and Condition Descriptions.

As stated in Chapter 2 there is an increasing movement in manufacturing to integrate product and process models for the purposes of life cycle data management. Therefore any system which can manage both product and process information in a common model is very desirable to the manufacturing community. Basing both a product and a process model on the above patterns and associating product descriptions with process descriptions, provides a uniform model for manufacturing. The association between the two descriptions carries semantics in that it describes how a particular process description is applied to a particular product description and any conditions or constraints on how the process acts on the product (see Figure 3-12). This association of process description to product description is very powerful - it allows different associations to be defined between a product and different processes that can take place throughout its life cycle e.g. design, assembly, testing, maintenance etc. For example, the association of a maintenance process to a product will require quite different conditions to be captured from those that are captured when a design process is carried out on that same product. The integration of PDM with WfM as outlined later in this thesis demonstrates the power of a unified product and process life cycle model.

3.5.3 Handling Evolution

Production systems should cater for the evolution of product or process descriptions regardless of the current state of the production and even while the production continues. Since, as discussed earlier, layers in a description-driven system are only loosely coupled, modifications in the meta-model layer can be carried out asynchronously from the application of those modifications in the model layer (which is itself defined in and generated from the meta-model layer). Similarly, modifications in the model layer can be asynchronously applied from their instantiations in the instance layer.

Even though the modifications are asynchronously applied in each layer, notification of the modification is required to provide traceability in the production systems. This mechanism can be handled through a combination of the Publish/Subscribe pattern, described in Section 3.4.2.4, the Item Description pattern of Section 3.4.2.1 and the Version pattern of Section 3.4.2.3. In the combination of these patterns, an ItemDescription is a concrete Publisher and any Item associated with this ItemDescription is a concrete Subscriber. A modification in the ItemDescription (at the model layer) is then notified to its Subscribers (at the instance layer) which can apply their modifications when appropriate.

The application of the Subscribers' modifications follows the Homomorphism pattern, described in Section 3.4.2.2. The Homomorphism pattern provides linkage between versions of Items and ItemDescriptions. Consequently an Item can determine the consequences to itself of moving to a new version of an instantiation of its ItemDescription.

3.6 Meta-Objects and Standardisation

In distributed object-based systems, object request brokers, such as the Object Management Group's CORBA (OMG, 1992a) provide for the exchange of simple data types and, in adition, provide location and access services. The CORBA standard is meant to standardise how systems interoperate. OMG's CORBA Services (OMG, 1994) specify how distributed objects should participate and provide services such as naming, persistent storage, life cycle, transaction, relationship and query. The CORBA Services standard is an example of how self describing software components can interact to provide interoperable systems.

Recently a considerable amount of interest has been generated in meta-models and meta-object description languages (Laddaga and Veitch, 1997). Work has been completed within the OMG on the Meta Object Facility (MOF, OMG 1997) which is expected to manage all kinds of meta-models relevant to the OMG Architecture. This meta-modeling approach will facilitate further integration between product data management and workflow management thereby providing consistency between design and production and speeding up the process of implementing design changes in a production system.

The purpose of the OMG MOF is to provide a set of CORBA interfaces that can be used to define and manipulate a set of interoperable meta models. The MOF is a key component in the CORBA Architecture as well as the Common Facilities Architecture. The MOF uses CORBA interfaces for creating, deleting, manipulating meta objects and for exchanging meta models.

The intention is that the meta-meta objects defined in the MOF will provide a general modeling language capable of specifying a diverse range of meta models (although the initial focus was on specifying meta models in the Object Oriented Analysis and Design domain). It has been designed to support:-

- *Generality*: it should be capable of describing a range of meta models.
- *Extensibility*: it is a core model and is capable of extension by inheritance and composition
- *Reuse*: when developing meta-data for a new application it should be possible to reuse meta-data from other similar applications
- *Reflection*: it should be capable of being able to represent itself (see Maes, 1987).

The usage of the MOF will depend very much on viewpoint. From a systems designer's viewpoint, who will be looking down the meta architecture layers, the MOF is used to define an information model for a particular domain of interest. Another viewpoint is that of a systems programmer who is looking up the meta levels. The concern here is for CORBA clients to obtain information model descriptions to support reflection and interoperability. The four-layer OMG meta-model has been discussed in Section 3.2 and was shown in Figure 3-2.

4 Test Application Environment

4.1 Introduction

An application has been developed to test the concepts put forward in this thesis. The test application that will form the basis of this study has been developed to manage the construction of a large scale high energy physics detector. The detector under construction is called the CMS Electromagnetic Calorimeter (ECAL) and the construction will take place across multiple test and assembly centres distributed worldwide over the period 1998-2005. The ECAL detector is very complex comprising hundreds of thousands of individual products, many of which are composite in nature, and each product must undergo a series of measurements and tests during assembly. An information system is required to track the status of assembly both from a product standpoint and from a process standpoint. This information system must be built on a data model which facilitates product and process tracking and is sufficiently flexible to cater for evolution of the product and process definitions over the period of detector construction. It is therefore an ideal vehicle in which to study the integration of PDM and WfM.

This chapter describes the domain of CERN where the current study of integration between workflow management and product data management has been conducted. Firstly, an introduction is given to the working environment at CERN which dictates some important design constraints on any software that is used to support activities at CERN. Secondly the peculiarities of designing systems for operation at CERN’s Large Hadron Collider (LHC, 1993) are introduced before the specifics of the Compact Muon Solenoid (CMS, 1995) experiment and its Electromagnetic Calorimeter (ECAL, 1997) production is detailed. Finally, having investigated the environment for software design and operation in CMS, the chapter discusses aspects of scientific workflow management and establishes a set of constraints which must be satisfied in the design of any workflow and product data management software used by experiments at CERN. It concludes by proposing the design approach followed in constructing the CRISTAL prototypes and indicates the method followed in carrying out the present study.

4.2 The Test Environment

4.2.1 The European Laboratory for Particle Physics (CERN)

CERN, the European Laboratory for Particle Physics, exists primarily to provide European physicists with accelerators that meet research demands at the limits of human knowledge (see Figure 4-1). In the quest for higher energies which its experiments need to conduct their research, the Laboratory has played a leading role in developing colliding beam machines. Notable ‘firsts’ were the Intersecting Storage Rings (ISR) proton-proton collider commissioned in 1971, and the proton-antiproton collider at the Super Proton Synchrotron (SPS), which came on the air in 1981 and produced the massive W and Z particles two years later, confirming the unified theory of electromagnetic and weak forces (for more detail see the CERN Web pages: http://www.cern.ch). The main impetus at present is from the Large Electron-Positron Collider (LEP), where measurements unsurpassed in quantity and quality are

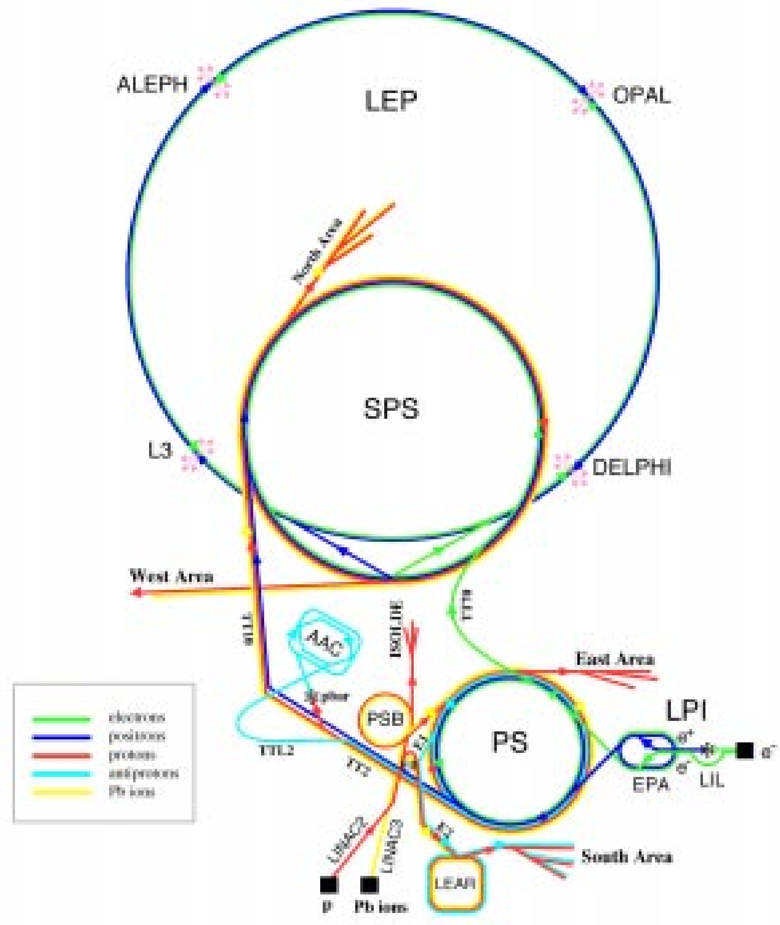


Figure 4-1. Current setup of accelerators at CERN

testing the best description of sub-atomic Nature, the Standard Model, to a fraction of 1% soon to reach one part in a thousand. By 1996, the LEP energy was doubled to 90 GeV per beam in LEPII, opening up an important new discovery domain. More high precision results are expected in abundance throughout the rest of the present decade, which should substantially improve the present understanding. The LEP/LEPII missions will by then be largely completed.

4.2.2 The Large Hadron Collider Project (LHC)

LEP data are so accurate that they are sensitive to phenomena that occur at energies beyond those of the machine itself; rather like the delicate measurement of earthquake tremors far from an epicentre. This gives us a ‘preview’ of exciting discoveries that may be made at higher energies, and allow us to calculate the parameters of a machine that can make these discoveries. All evidence indicates that new physics, and answers to some of the most profound questions of our time, lie at energies around 1 TeV (1 TeV = 10^{12} electron-Volts). To look for this new physics, the next research instrument in Europe's particle physics armoury is the LHC (LHC, 1993). In keeping CERN's cost-effective strategy of building on previous investments, it is designed to share the 27-kilometre LEP tunnel, and be fed by existing particle sources and pre-accelerators. A challenging machine, the LHC will use the most advanced superconducting magnet and accelerator technologies ever employed. LHC experiments are, of course, being designed to look for theoretically predicted phenomena. However, they must also be prepared, as far as is possible, for surprises. This will require great ingenuity on the part of the physicists and engineers. The LHC is a remarkably versatile accelerator. It can collide proton beams with energies around 7-on-7 TeV and beam crossing points of unsurpassed brightness, providing the experiments with high interaction rates. Joint LHC/LEP operation can supply proton-electron collisions with 1.5 TeV energy, some five times higher than presently available at HERA in the DESY laboratory, Germany. The research, technical and educational potential of the LHC and its experiments is enormous.

The LHC is an accelerator which will bring protons into head-on collision at higher energies (14 TeV) than ever achieved before to allow scientists to penetrate still further into the structure of matter and recreate the conditions prevailing in the Universe just 10^{-12} seconds after the ‘Big Bang’ when the temperature was 10^{16} degrees. The LHC luminosity will reach L = 10^{34} cm^{-2} s^{-1} (a quantity proportional to the number of collisions per second). This will be achieved by filling each of the two rings with 2835 bunches of 10^{11} particles each. The time between two bunch crossing is 25*10^{-9} seconds resulting in approximately 10^{9} interactions per second i.e. 20 interactions per crossing on average. The resulting large beam current (of around 0.53 A) is a particular challenge in a machine made of delicate superconducting magnets operating at cryogenic temperatures. When two bunches cross in the center of a physics detector only a tiny fraction of the particles collide head-on to produce the wanted events.

4.2.3 The Compact Muon Solenoid Detector (CMS)

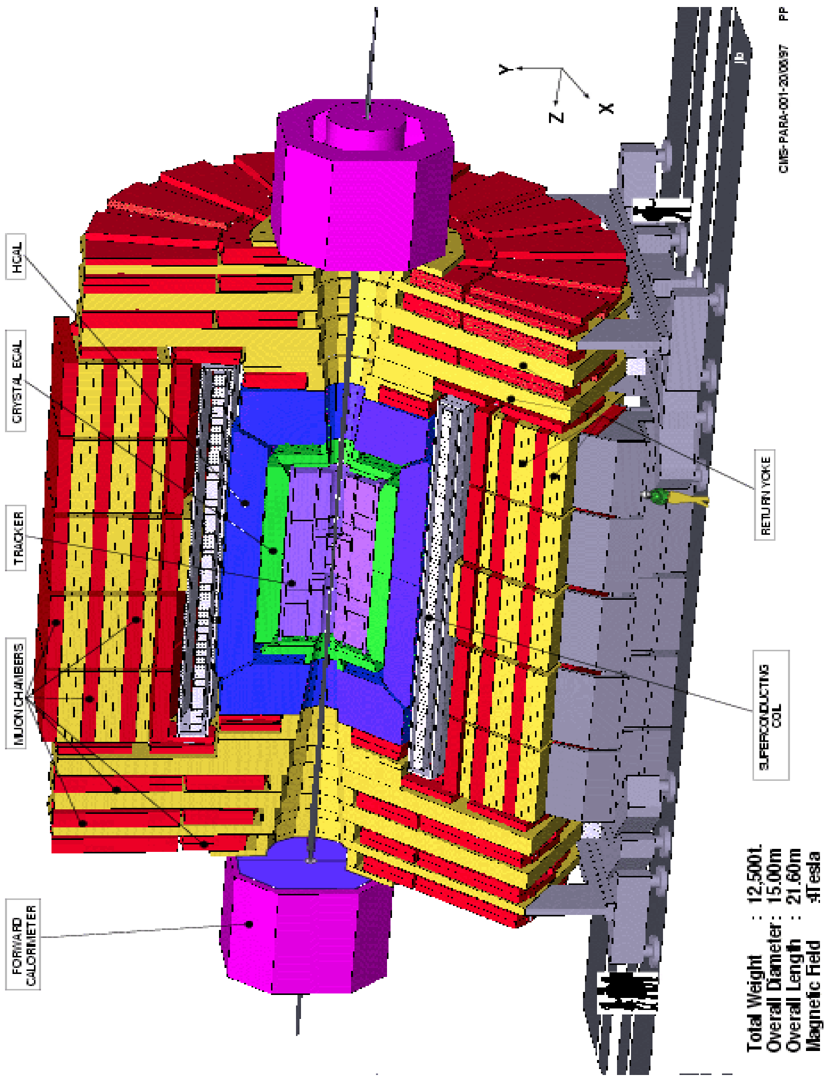


Figure 4-2. An artist's impression of the layout of the CMS high energy physics detector

The CMS detector (CMS, 1995 - see Figure 4-2) is one of two general purpose detectors to be installed at the future Large Hadron Collider. The CMS detector can be subdivided into one barrel region and two identical end-cap regions. The central element of the CMS detector is a 13 meter long, 6 meter diameter superconducting solenoid generating a uniform magnetic field of 4 Tesla. The magnetic flux is returned through a 1.5 meter thick saturated iron yoke instrumented with muon chambers. Located within the solenoid bore are the central tracker (TRACKER 1998) and both the electromagnetic (ECAL, 1997) and hadron (HCAL, 1997) calorimeters. The overall length (excluding the very forward calorimeters) and width of the detector are respectively 22 and 14.6 meters respectively; the total weight will be about 14500 tons.

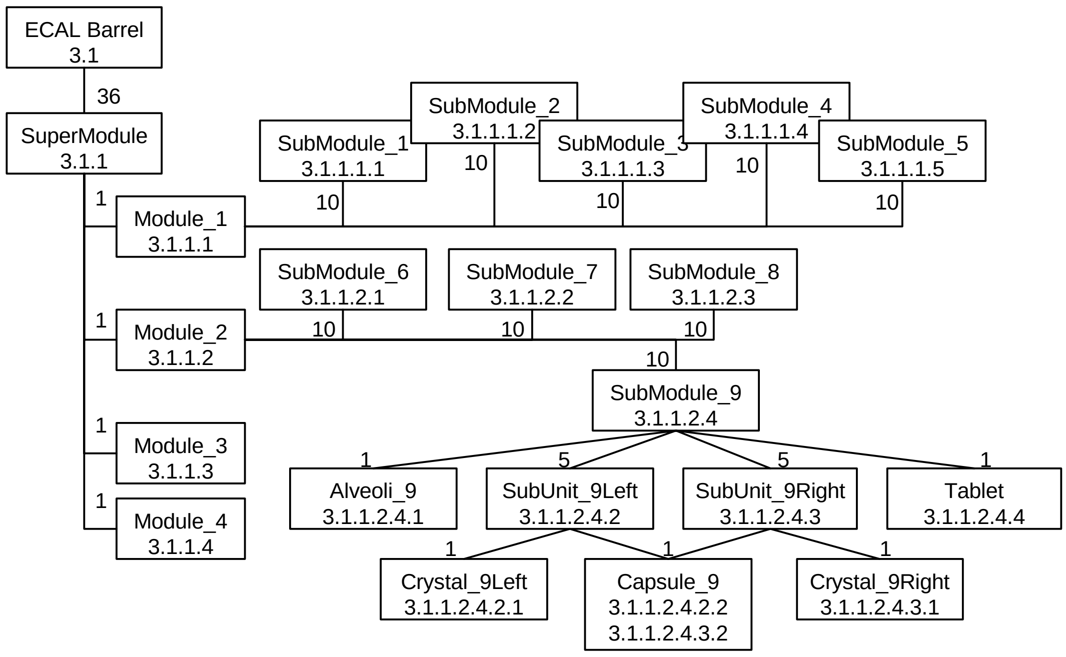


Figure 4-3. The Product Graph of the CMS Electromagnetic Calorimeter (ECAL) Barrel

The central tracker is designed to reconstruct high transverse momentum muons, electrons and hadrons with very good accuracy. To achieve the requested accuracy it has to be subdivided into approximately 10^8channels for each which individual location has to bee known precisely.

The primary function of the ECAL is the precise measurement of the energy of both electrons and photons, and, in conjunction with the HCAL, the measurement of jets. The ECAL subdetector is composed of about 61,200 Lead Tungstate crystals in a Barrel structure (see Figure 4-3) and 21,528 crystals in two EndCap structures. Each Barrel crystal is equipped with twin avalanche photodiodes and each EndCap crystal is equipped with vacuum phototriodes to convert the light induced in the crystals by the incoming particles into a meaningful electronic signal that can be processed by the readout chain. The total weight of the crystals is 67.4 tonnes occupying 8.14 cubic metres of volume (see Figure 4-4).

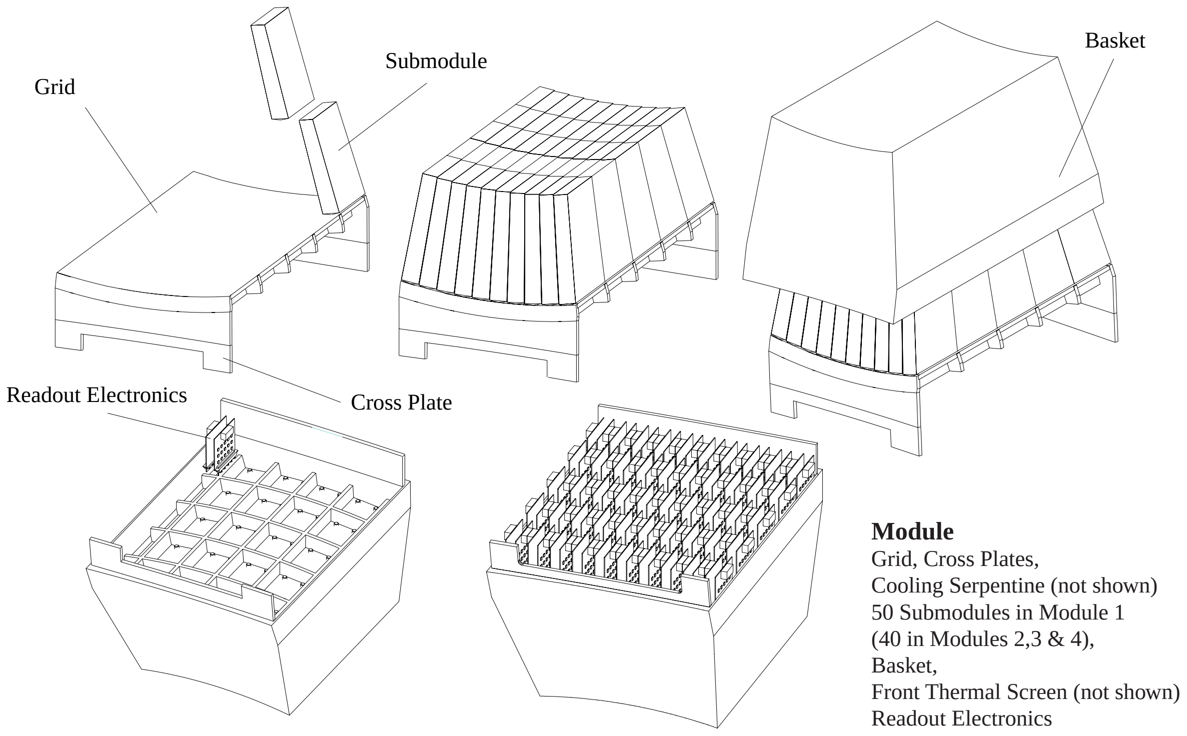


Figure 4-4. A mechanical engineering drawing of an ECAL Barrel Module.

Each individual Barrel crystal (of which there are 34 types) undergoes a series of quality assurance tests followed by dimensional measurements and determination of transverse and longitudinal transmissions and light yields (which must be stored for future reference) prior to being glued onto the photodiode capsule. Once glued the assembly is known as a sub-unit and further tests are performed to determine the quality of the gluing and to ensure that the light produced in the crystal can be measured in its associated electronics. Ten sub-units, plus support and cooling structures are assembled together to form a sub-module on which more tests are performed. Modules are constructed from 40 (or 50) sub-modules and supermodules are comprised of four modules of different types plus readout electronics, support structures, monitoring devices and cabling. There are 36 supermodules in the cylindrical ECAL Barrel. The EndCaps follow a similar structure. In total, excluding cabling, the ECAL comprises about 400,000 products of about 500 product types.

The HCAL subdetector is organized in 18 identical wedges corresponding to approximately the same number of channels as the ECAL. Finally, the Muon system is organized in four stations. Each station consists of 12 planes of drift chambers leading to a total of about 10^6 channels.

Each sub-detector channel will be readout individually by a very fast electronic chain then digitised and analysed for physics event selection. After event selection, the resulting amount of data collected will be of the order of 1 MByte per event.

4.3 The CRISTAL Application

The construction process of detectors for the Large Hadron Collider (LHC) experiments is long scale, heavily constrained by resource availability and evolves with time. As a consequence, changes in detector component design need to be tracked and quickly reflected in the construction process (Lebeau, Lecoq and Vialle, 1995). As stated in Chapter 2, with similar problems in industry engineers employ Product Data Management (PDM) systems to control access to documented versions of designs and managers employ Production Schedulers or Workflow Management software (WfM) to coordinate production work processes. The scale of LHC experiments, like CMS, demands that industrial production techniques be applied in detector construction.

However, as noted in Chapter 2, no commercial products provide the integrated PDM and WfM capabilities required for the construction of large scale high energy physics detectors. A research project, entitled CRISTAL (Cooperating Repositories and an Information System for Tracking Assembly Lifecycles) has been established to provide integrated product and process management, based on a common data model, during the construction of CMS. This is the first time industrial production techniques have been deployed to this extent in high energy physics detector construction. This section outlines the major functions and applications of the CRISTAL (Baker et al., 1998 and Bazan et al., 1998) system developed for CMS. CRISTAL is the vehicle in which the integration of PDM and WfM techniques in managing large scale physics detector construction is studied for this thesis.

The CMS detector will be built of many subdetectors such as ECAL, HCAL, Tracker etc. Each of these sub-detectors will be based on quite different physics principles, will be constructed for different purposes and will be designed and realised by autonomous groups. As a consequence the production of these sub-detectors will be started independently and carried on in parallel with that of other sub-detectors and integration with other subdetectors will take place late in CMS construction. Consequently each subdetector will use an independent installation of CRISTAL and will provide their own specification of their construction procedures.

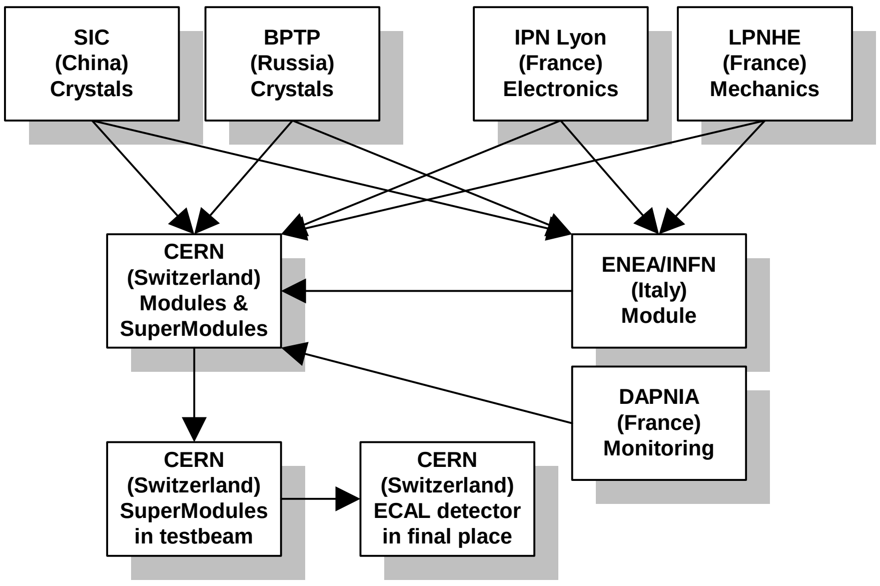

SIC (China) Crystals
BPTP (Russia) Crystals
IPN Lyon (France) Electronics
LPNHE (France) Mechanics
CERN (Switzerland) Modules & SuperModules
ENEA/INFN (Italy) Module
DAPNIA (France) Monitoring
CERN (Switzerland) SuperModules in testbeam
CERN (Switzerland) ECAL detector in final place

Figure 4-5. The movement of physical parts between centres.

Each CMS sub-detector will be constructed from a large number of precision parts (CMS, 1995) and will be produced and assembled during the next few years by *centres* distributed worldwide (see Figure 4-5). These centres will include Shanghai Institute of Ceramics (SIC) and Bogodoritsk Plant of Technochemical Products (BPTP) crystal production centres, IPN Lyon capsule testing centre, LPNHE alveoli centre, CERN and ENEA/INFN assembly centres and DAPNIA monitoring centre in Saclay, France. Each constituent product of each detector will be measured and tested locally, prior to its assembly and integration in the experimental area at CERN. Much of the information collected during this phase will be needed not only to construct the detector, but for its calibration, to facilitate accurate simulation of its performance and to assist in its maintenance over the lifetime of the detector. The construction process is heavily dependent on many areas of research (materials science, electronics, computing) so that systematic tracking of the evolution of individual detector parts and quality control of the assembly process are essential for the subsequent calibration and maintenance of the detector. Furthermore, coordinating operations required for construction can be complex given the number of products requiring characterisation, particularly when the operations are distributed over geographically separated centres.

Large-scale industrial production systems (such as aeroplane manufacture) have similar requirements and often employ product data management tools to manage the data and documents accumulated in the design of product(s). These systems are normally based on commercially available PDM products and successfully support the creative and collaborative

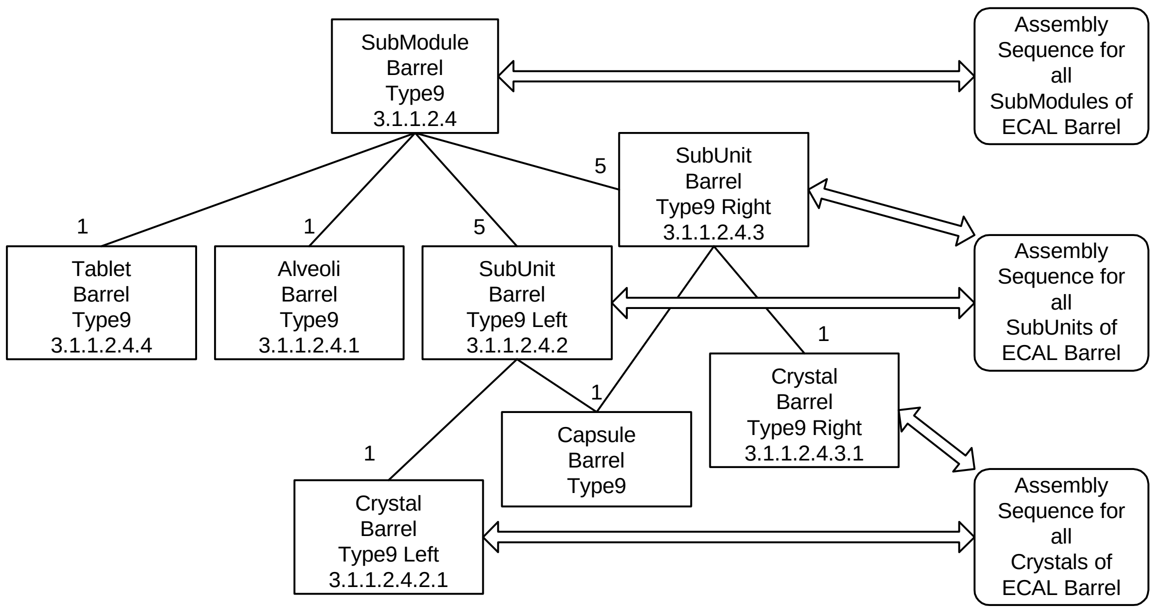

SubModule Barrel Type9 3.1.1.2.4
Assembly Sequence for all SubModules of ECAL Barrel
SubUnit Barrel Type9 Right 3.1.1.2.4.3
Assembly Sequence for all SubUnits of ECAL Barrel
Tablet Barrel Type9 3.1.1.2.4.4
Alveoli Barrel Type9 3.1.1.2.4.1
SubUnit Barrel Type9 Left 3.1.1.2.4.2
Crystal Barrel Type9 Right 3.1.1.2.4.3.1
Capsule Barrel Type9
Crystal Barrel Type9 Left 3.1.1.2.4.2.1
Assembly Sequence for all Crystals of ECAL Barrel
1
1
5
5
1
1
1

Figure 4-6. The Product Graph and assembly sequences of a Barrel sub-module.

stages of product design where access to documents needs to be controlled between groups of designers. However, their use in supporting the unstructured processes inherent in product research and development is somewhat limited – they provide limited facilities for activity definition and no facilities for the enactment of production activities. Workflow management systems, on the other hand, do provide utilities which facilitate the coordination and scheduling of the activities of organisations to optimise the flow of information between resources and there are many such tools becoming available commercially. However, WfMs alone are weak at handling the dynamic evolution of process and activity definitions which occur during design and during the enactment of workflow processes.

No commercial products provide the workflow and product data management capabilities required by CMS. A research project, entitled CRISTAL has therefore been initiated, using component software technologies where possible, to facilitate the management of the engineering data collected at each stage of production of CMS. CRISTAL uses the so-called ‘as designed’ view of the detector (potentially stored in a Product Data Management system) to build a distributed production scheme which spans construction centres. As the detector construction evolves during assembly (see Figure 4-6) and also during the lifetime of the experiment, versions of the production scheme are dispatched from CERN to the centres where the so-called ‘as built’ view of the detector is gathered, as a consequence of following the execution of pre-defined activities. On execution of these activities, CRISTAL captures all the physical characteristics of detector components, which are, later, required for detector construction, calibration and maintenance. In Figure 4-6, the product composition graph of an ECAL Barrel submodule is displayed. For each sub-module component (e.g Sub-Unit, crystal, capsule etc.) an assembly and testing sequence of workflow activities can be specified for execution in the crystal, capsule, sub-unit and sub-module construction process. Each of these workflow activities can be composite in nature and a composite workflow activity will have a particular workflow activity layout, as shown in Figure 4-7 for crystal assembly.

Ultimately, workflow activities are elementary or atomic in nature: i.e. they carry out a single task which produces a specific outcome. Elementary activities have descriptions, can be applied at particular centres, have duration and can be executed by Operators, by Instruments or by User-Supplied software. Conditions can be assigned to these elementary workflow activities. These production conditions may be a pre-requisite to the execution of the workflow activity (so-called Start Conditions) or be a test for successful completion of the workflow activity (so-called End Conditions) as shown in Figure 4-8. These conditions are discussed further in the following chapter.

In the first instance CRISTAL is being used to monitor and control the production and assembly process of the CMS Electromagnetic Calorimeter (ECAL, 1997). The software employs workflow and task management techniques and is generic in design and will therefore be reusable for other CMS detector groups or indeed by groups with similar requirements outside of CMS. At the time of writing, the Tracker (TRACKER, 1998) subdetector

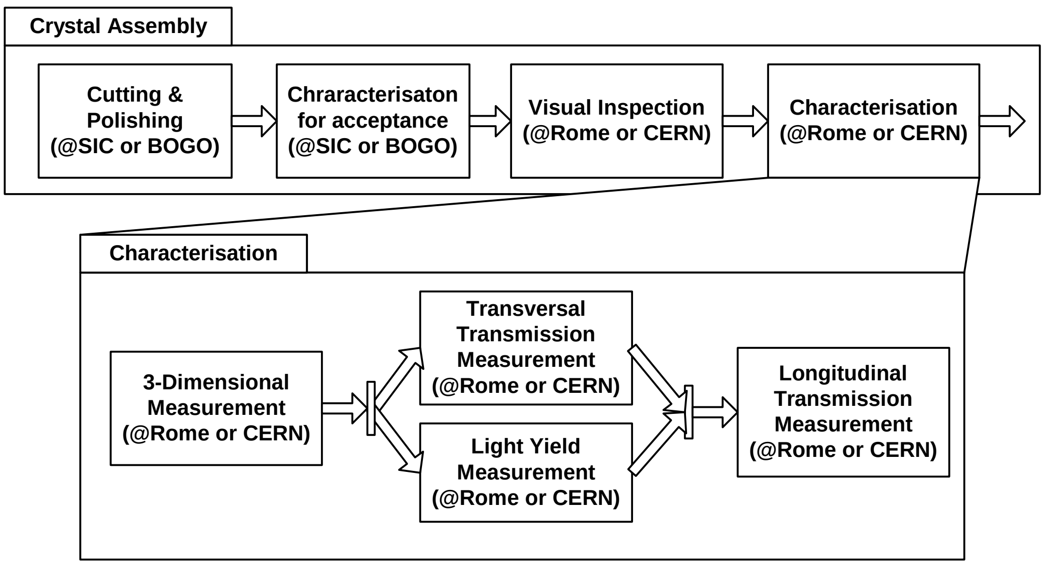

Crystal Assembly
Cutting & Polishing (@SIC or BOGO)
Chraracterisaton for acceptance (@SIC or BOGO)
Visual Inspection (@Rome or CERN)
Characterisation (@Rome or CERN)
Characterisation
3-Dimensional Measurement (@Rome or CERN)
Transversal Transmission Measurement (@Rome or CERN)
Light Yield Measurement (@Rome or CERN)
Longitudinal Transmission Measurement (@Rome or CERN)

Figure 4-7. Crystal assembly workflows and the Characterisation sub-workflow.

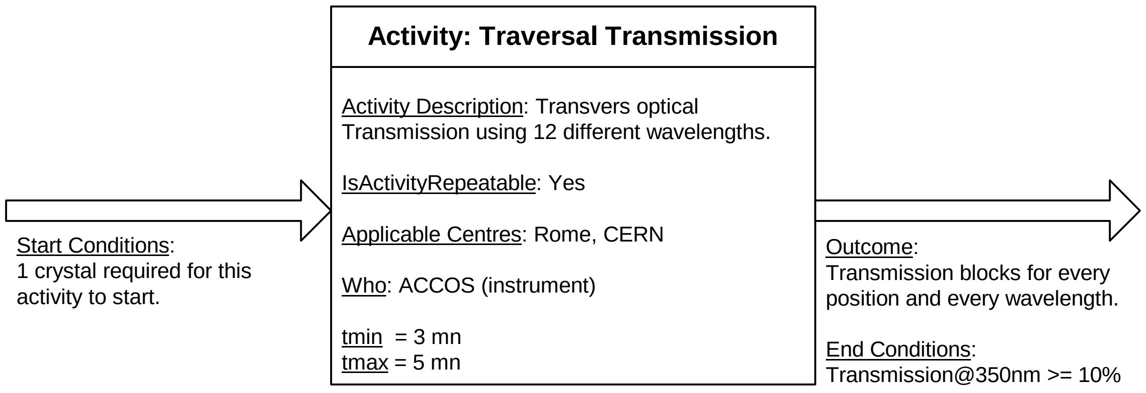

Activity: Traversal Transmission
Activity Description: Transvers optical Transmission using 12 different wavelengths.
IsActivityRepeatable: Yes
Applicable Centres: Rome, CERN
Who: ACCOS (instrument)
tmin = 3 mn
tmax = 5 mn
Start Conditions: 1 crystal required for this activity to start.
Outcome: Transmission blocks for every position and every wavelength.
End Conditions: Transmission@350nm >= 10%

Figure 4-8. A workflow activity definition including Start and End conditions.

group has committed to the use of CRISTAL and the Hadronic calorimeter (HCAL, 1997) group is evaluating its usage.

Following the growth and characterisation of the ECAL Barrel crystals in Production Centres in China and Russia, assembly of the crystals with their Capsules (produced in France) and associated support structures (alveoli) will take place in Local Centres located in Italy, the UK and CERN. Each of the component products (crystals, electronics, alveoli) will have their physical characteristics individually measured and recorded to facilitate calibration and to ensure consistency of the production process (see Figure 4-5).

Since the overall costs and time scales of ECAL crystal production must be strictly controlled, the efficiency of the production process will be paramount. Quality control must be enforced at each step in the fabrication process. The CRISTAL system must support the testing of detector parts, the archival of accumulated information, controlled access to data and the on-line control and monitoring of all production and assembly centres.

CRISTAL needs to be both a production management and workflow management facility that tracks products through the manufacturing, assembly and maintenance life cycle. Ultimately, it will provide the *as-built* view to information stored during ECAL construction, which is required by physicists as the basis for all detector-related analyses, and is capable of providing support for other views of the construction data.

These views include the following as examples:

- A Calibration View. Where physicists will want to view and access product characteristic data for experiment calibration and event reconstruction purposes.

- A Maintenance View. Where engineers will refer to the production processes for assembly and disassembly procedures, update information collected through maintenance operations and design modifications throughout the experiment’s lifetime.
- An Experiment Systems Management View. Where the so-called ‘slow control’ system can view the product production history for configuration and fault management purposes.

In summary, the CRISTAL project aims to implement a prototype distributed engineering information management system, which will control the construction process of CMS ECAL. Its specific objectives are to: -

- provide an information system to control quality in the construction of detectors
- monitor the global production process across distributed centres
- capture and store crystal data during detector production and construction
- integrate instruments used to characterise products
- provide controlled, multi-user access to the production management system and
- provide access to engineering and calibration data for CMS users

4.4 Design Constraints

The CERN CRISTAL project combines many of the requirements of second generation workflow systems. This section uses the CRISTAL project as a vehicle in which to demonstrate aspects of design constraints for scientific workflows. Firstly, the environment at CERN is research-based and both workflow and product-related definitions tend to evolve rapidly over time. The CRISTAL software must cater for the development of a High Energy physics detector (CMS, 1995) which will take place over an extended period of time (1999-2005) and whose design will naturally advance as time elapses. As a consequence of this CRISTAL must also support long-running and potentially nested workflow activities, with natural consequences for transaction handling.

Secondly, the construction of CMS is a one-of-a-kind process. In other words, the evolution of workflows and product data must be allowed to take place as the production continues - versions of workflow activities and product definitions must co-exist in the production process for the duration of CMS construction. This is in contrast to industrial production lines where the process is seldom one-of-a-kind; normally the production line produces many copies of products that follow a fixed set of repeatable processes during production. If the sequence of processes (i.e the workflow) requires updating, then the production line is flushed of products prior to the change taking place. In CMS, the production processes can be changed while production continues. This implies the coexistence of products (of potentially different versions) in the same production line, which could be following different versions of a workflow. In addition, users of CRISTAL must ultimately be able to cater for the ad-hoc definition and execution of workflows, as and when required in CMS construction.

Thirdly, the CMS construction process is highly distributed. Production of (versions of) CMS products will take place in areas as disparate as China and Russia, their testing will be undertaken in Rome, CERN and the United Kingdom and assembly will largely take place at CERN. Each of these ‘Centres’ must cater for multiple versions of evolving workflow definitions in an autonomous manner but be centrally coordinated from CERN. As a consequence of the autonomous operation of each centre and the requirement that product management must follow physical product transportation, it will be necessary to provide product ‘flow’ capability, tracked by a ‘central system’. In other words, production activities can be initiated at one centre and continued at another. Therefore, data collected at the source centre must be made available at the destination centre so that the production flow can be managed without interruption. This implies that product-related data needs to follow the shipped product between centres and that a mechanism is required to resume production which previously had been suspended at another centre.

Finally, the data collected in the CRISTAL database must be reliably secure (since many processes cannot be undone or redone) and available for a variety of purposes. In other words, many different users require access to the CRISTAL data from a variety of *viewpoints*: construction engineers interpret data using an assembly-oriented view whereas physicists see the detector in terms of a set of electronically-decoded channels and mechanical engineers view the detector in terms of constituent 3-dimensional volumes aligned in space.

These design constraints cannot currently be satisfied by any commercial offering. The

CRISTAL design team were therefore free to develop an open and flexible solution. The team adopted an approach in which a model was developed that provided generality through the use of meta-data, describing general structures, rather than specifying a restricted view onto the workflow or product-specific data. This is in contrast to many workflow solutions which take specific data and attempt to abstract to the more generic.

The particular constraints of time, evolution and flexibility quickly led to the adoption of an object-oriented approach to product and process modeling. The result has been a detailed UML model, presented in the next chapter.

4.5 Design Approach

Due to the specialised environment in which this research was carried out a design approach was adopted which:

- was object-oriented in nature and supported reuse of software components
- handled the distributed nature of CRISTAL usage
- was research-based and catered for evolving user requirements
- resulted in models which were adaptable and supported system versioning
- supported the deferral of decisions on enabling technology to later in the software life cycle
- was sufficiently flexible and ‘open’ in nature to provide interoperability with other systems

During the analysis phase of the CRISTAL project it was clear that user requirements would be difficult to specify since the product and process descriptions themselves were the subject of research and since PDM and WfM techniques had not been rigorously applied in the environment before. Because of these constraints a rapid prototyping (Davis et al. (1988), Gordon and Bieman (1991) and Lowell (1992)) philosophy was followed in constructing a series of prototypes which were used to feedback design decisions to the user community for approval. Rather than following a rigid phased approach to systems development (*a la* Waterfall Model (Pressman, 1992)) in which design followed analysis and preceded coding, an evolutionary approach to systems development, using UML, was followed where multiple CRISTAL prototypes were produced over time and the final prototype was delivered in a step-wise fashion. In this way, each release incorporated the experiences of earlier releases in a manner analogous to the 'Spiral Model' of software development suggested by Boehm (1988) and each release was available for verification with end-users prior to commitment being given to the next stage of prototype development.

At the outset of the research it was clear that an object model of the users requirements was needed and that this model should capture the essence of the system description. At that stage it was also clear that it would not be possible to use a fully fledged pattern-based approach to building the object model since the users' requirements were liable to evolve, perhaps significantly, over the period of design. In other words, it was not possible to use existing design frameworks (or even single design patterns) as the starting-point for the development of an overall CRISTAL object model. Rather the object model emerged iteratively following detailed discussion with knowledgeable end-users of each prototype's functionality.

In developing an object model which was sufficiently flexible to subsume design amendments as they emerged over time, that maximised reusability of design components and interoperability of the overall system and, ultimately, produced an adaptable model, it was necessary to study closely the structure of the model that emerged. Each change in the model was discussed with the user community and then its effect on the existing model was considered closely prior to its incorporation in the model. As is perhaps expected with description-driven systems, the model that emerged developed a certain symmetry and even an element of relative simplicity in its structure.

As the model stabilised over time its essential structures became rooted in the fundamental design of the CRISTAL system and those structures displayed an elegant symmetry with respect to the capture of description for product and process-based models. It became clear that the design approach was producing a model with repeating structures which could be abstracted into design meta-objects that captured a high level of systems description. Having completed the data model for the description-driven CRISTAL system, including meta-objects, it was intended to compare the resultant structures with those proposed by the patterns research community, so that their design patterns could be enriched and improved as a result of the study of description-driven systems for the integration of PDM and WfM. The following chapter describes the detail of the CRISTAL model and shows that repeating design structures (or patterns) resulted from the meta-modeling process that was employed.

5 Prototype Data Model

5.1 Introduction

As the CMS construction process gets under way production schemes and product specifications will continue to evolve. Clearly, these changes in definition must be folded into any data which is derived from the construction data system. One way of achieving this is for the system to make available a representation of itself for manipulation. A system which can make modifications to itself by virtue of containing a description of its own computation is called a reflective system (Maes, 1987 and Peters and Ozsu 1994). In order to inter-operate in an environment of future systems and in order to adapt to reconfigurations and versions of itself the CRISTAL system should adopt some aspects of reflective systems. This chapter presents the self-describing data model that has been developed as the basis of the CRISTAL prototypes and which provides a degree of reflection.

5.2 The CRISTAL Description-Driven Architecture

The main feature of the CRISTAL data model resulting from the use of a generic design approach was its multi-layer architecture. Following the principles described in Section 3.2 on page 30 (and Figure 3-1 on page 31), a three-layer architecture has been developed: the instance layer, the model layer and the meta-model layer (see Figure 5-1). The instance layer comprises Items such as Product and Workflow Activity objects and the architectural components which manage this layer are known as the CRISTAL Execution components. The model layer comprises the Item classes and the associations between Item classes. Together these constitute a model from which Items are instantiated, and which define the CRISTAL Execution components. In addition this model layer contains ItemDescription instances (such as Product Definitions and Workflow Activity Definitions) which are managed by an architectural component known as the CRISTAL Specification component. (Description of the architectural components is deferred to the next chapter which discusses the physical prototype implementation of CRISTAL).

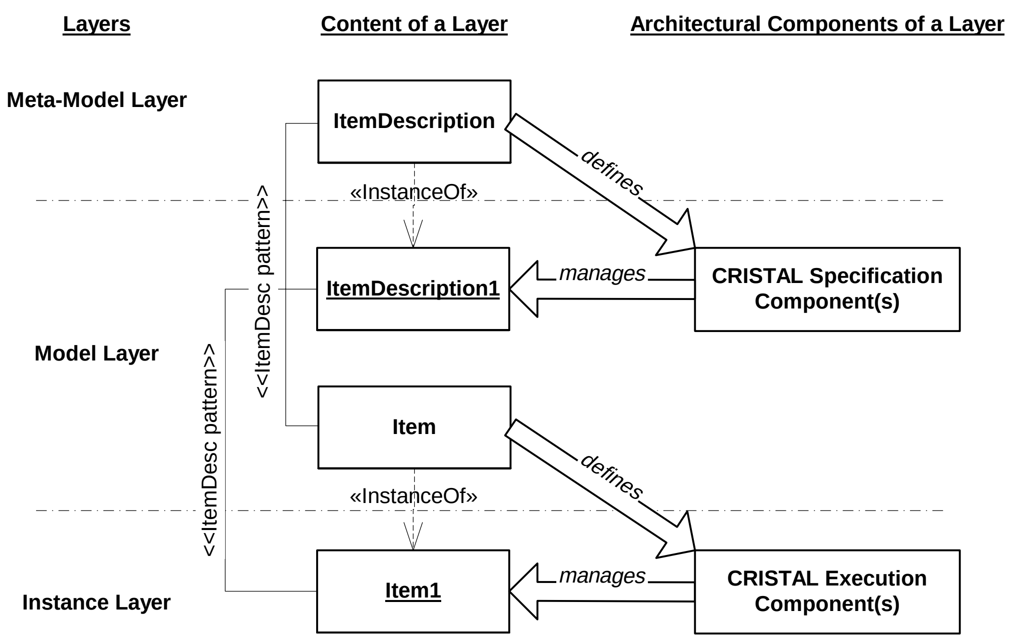


Figure 5-1. The CRISTAL three-layer architecture.

The meta-model layer comprises the ItemDescription classes and associations between Item-Description classes. Together these constitute a meta-model from which ItemDescriptions are instantiated and which define the architectural components used for specifying CRISTAL.

In Figure 5-1 the Item Description pattern, of Section 3.4.2.1 on page 36, has been employed to provide the semantic that relates the Item class (of the model layer) with the ItemDescription class of the meta-model layer. As a consequence the pattern can also be applied to relate an Item instance (at the instance layer) with its corresponding ItemDescription instance (at the model layer). The multi-layer architecture, which as described in Chapter 3 forms the basis of a description-driven system, is therefore a direct consequence of the use of the Item Description pattern. Figure 5-2 shows this use of the Item Description pattern at the level of collections of classes, called packages. Here a Descriptions Meta-Model of ItemDescription classes is modeled separately from an Items Model which contains collections of Item classes related to ItemDescription classes of the meta-model.

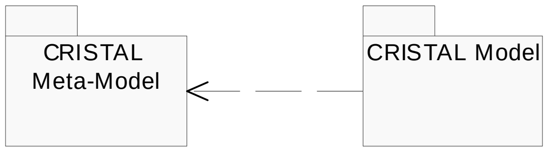


Figure 5-2. CRISTAL packages in the multi-layer architecture.

The following sections describe the CRISTAL Meta-model package and the CRISTAL Model package in detail. The first section describes the CRISTAL Meta-Model package, which corresponds to the meta-model layer of a multi-layer architecture (as described in Chapter 3) including Descriptions of Products, Workflow Activities, Executors and Data Formats. The second section describes the CRISTAL Model package (corresponding to the model layer of a multi-layer architecture) which has been instantiated from the descriptions in the Meta-Model package. The CRISTAL Model package contains classes of Products, Workflow Activities, Executors and instances of Data Formats.

5.2.1 CRISTAL Meta-Model Package

All of the major aspects of CRISTAL functionality, such as product model (PBS/ABS), workflow model (WBS), production conditions, data formats, execution conditions and executors are specified in the form of packages (see Figure 5-3). Packages are a convenient mechanism to partition the functionality of CRISTAL, they deal with logical subsets of the complete CRISTAL model and are themselves collections of object classes. There are dependencies between the packages, such as between the PBS/ABS, WBS and production conditions packages and these are shown in Figure 5-3 by arrows connecting the packages. It is these dependencies or assignments which provide the linkage between the object classes

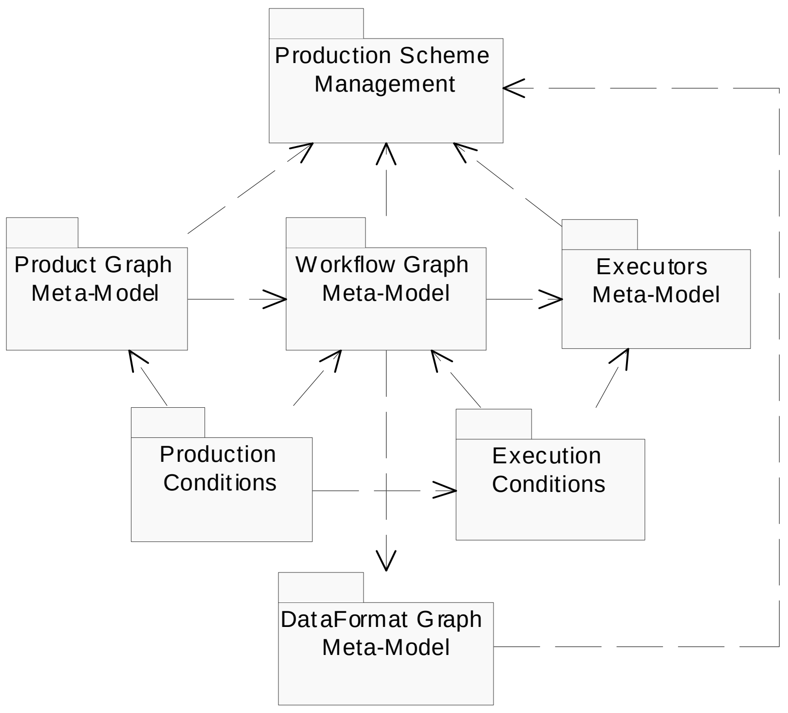


Figure 5-3. CRISTAL Meta-Model Packages

within different packages and allow integration between the PBS/ABS (or PDM-related) world and the WBS (or WfM-related) world. Together the packages and their inter-dependencies are referred to as the Meta-Model layer for CRISTAL.

The CRISTAL Meta-Model layer is composed of three main packages. The Product Graph Meta-Model package provides semantics to describe the composition and the layout of the detector elements, the Workflow Graph Meta-Model package defines the language to build workflow descriptions and the Executors Meta-Model package provides semantics to describe the different executors which are capable of carrying out a workflow activity. In other words these three packages hold the description of ‘What is to be produced’, ‘How it should be produced’ and ‘What is going to produce it’. The reason for the separation between these packages is so that they can evolve independently, given the different nature of their purpose. Without such a distinct separation the relationships and the dependencies between the packages would be difficult to identify and capture. To describe the dependencies between these packages, two more packages have been introduced: the Production Conditions package and the Execution Conditions package. The Production Conditions package describes any product-related conditions pertaining to the association of a type of workflow activity to a specific type of product e.g any product type instances required as prerequisites to the initiation of a workflow type instance (so-called ‘Start Conditions’). The Execution Conditions package describes any conditions pertaining to the association of a type of Executor to a specific type of workflow activity.

A further package, the Production Scheme Management package, has been created to show how the different definitions are held and managed to provide a sequence of consistent releases of the detector production specification. Finally, the Data Format Meta-Model package has been introduced as an abstraction of all the data structures that are used in CRISTAL to collect data.

The dependencies between packages show that in CRISTAL everything is organised around Product Definitions: the Product Graph Meta-Model is dependent on the Workflow Graph Meta-Model which, in turn, is dependent on the Executors Meta-Model. Dependencies also exist between the Production Conditions package and the Execution Conditions package.

The following sections give a detailed description of the packages (and the patterns which emerge from these packages) which are essential for the integration of PDM and WfM i.e the Production Scheme Management, the Product Graph Meta-Model, the Workflow Graph

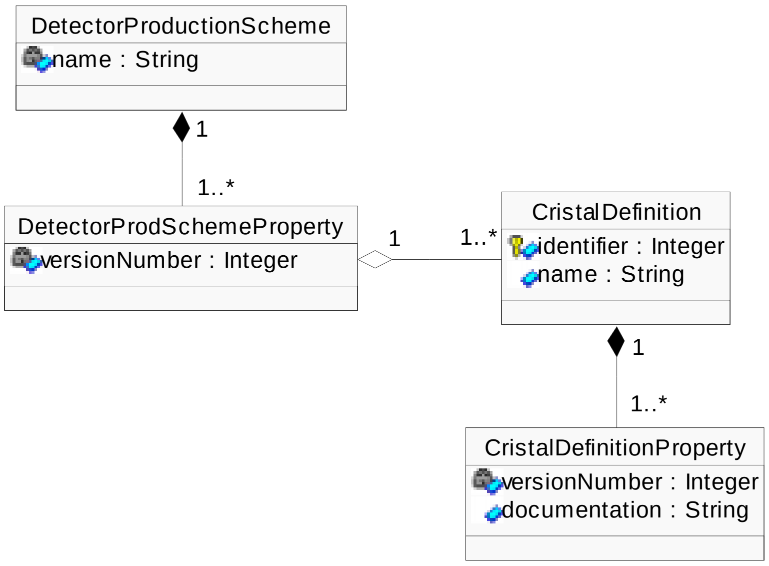


Figure 5-4. Production Scheme Management package

Meta-Model and the Production Conditions packages. In addition, the Data Format Meta-Model package will also be described since similar patterns emerge from its specification. The Executors Meta-Model and Execution Conditions package are not described since they do not have a direct impact to the subject of this thesis i.e. the integration of PDM and WfM systems.

5.2.1.1 Production Scheme Management Package

This package describes the elements for overall production/construction scheme management and shows the adopted versioning policies in CRISTAL. A DetectorProductionScheme (DPS) describes and manages the complete life cycle of all the CristalDefinitions used to specify the sub-detector as designed. Each version of the DPS, called a DetectorProdSchemeProperty, contains a collection of CristalDefinitions. CristalDefinitions themselves can be versioned so that one version of a CristalDefinition, called a CristalDefinitionProperty, could be used in one or more DPS versions. A CristalDefinition becomes versioned whenever it is modified. A DPS is versioned when a new production scheme has been released or when a CristalDefinition is added or removed from the current DPS release.

Each DPS version contains a list of references to CristalDefinitions but no information as to the version of each CristalDefinition. Each CristalDefinition manages its own version history - in other words, given the DPS version number each CristalDefinition is able to select the proper version from its ‘history’. This allows individual CristalDefinitions to be interrogated (e.g. for version evolution) whereas in a traditional configuration management system versioning is carried out only at the release level (i.e at the DPS level). Using the terminology of the Version pattern of Section 3.4.2.3 on page 37, CristalDefinitions are VersionedObjects and DPS is a ReleaseManager. Dynamic version handling is catered for through the use of the Publish/Subscribe pattern as described in Section 3.4.2.4 on page 38.

5.2.1.2 Product Graph Meta-Model Package

This package describes the PBS/ABS aspects of a sub-detector as-designed - it holds all details of all versions of the composition and layout of the designed sub-detector. The package follows the versioning strategy for all CristalDefinitions as described in the previous section, utilising the Version pattern of Section 3.4.2.3 on page 37, in conjunction with the Enriched Composite Item Description pattern of Section 3.5.1 on page 41.

Each PBS/ABS instance is represented in this package as a ProductDefinition object that has been instantiated from the ProductDefinition class. The Version pattern emerges from this package in that a ProductDefinition class is itself a specialisation of the CristalDefinition class and therefore it manages its own versions (i.e it has a CristalDefinitionProperty).

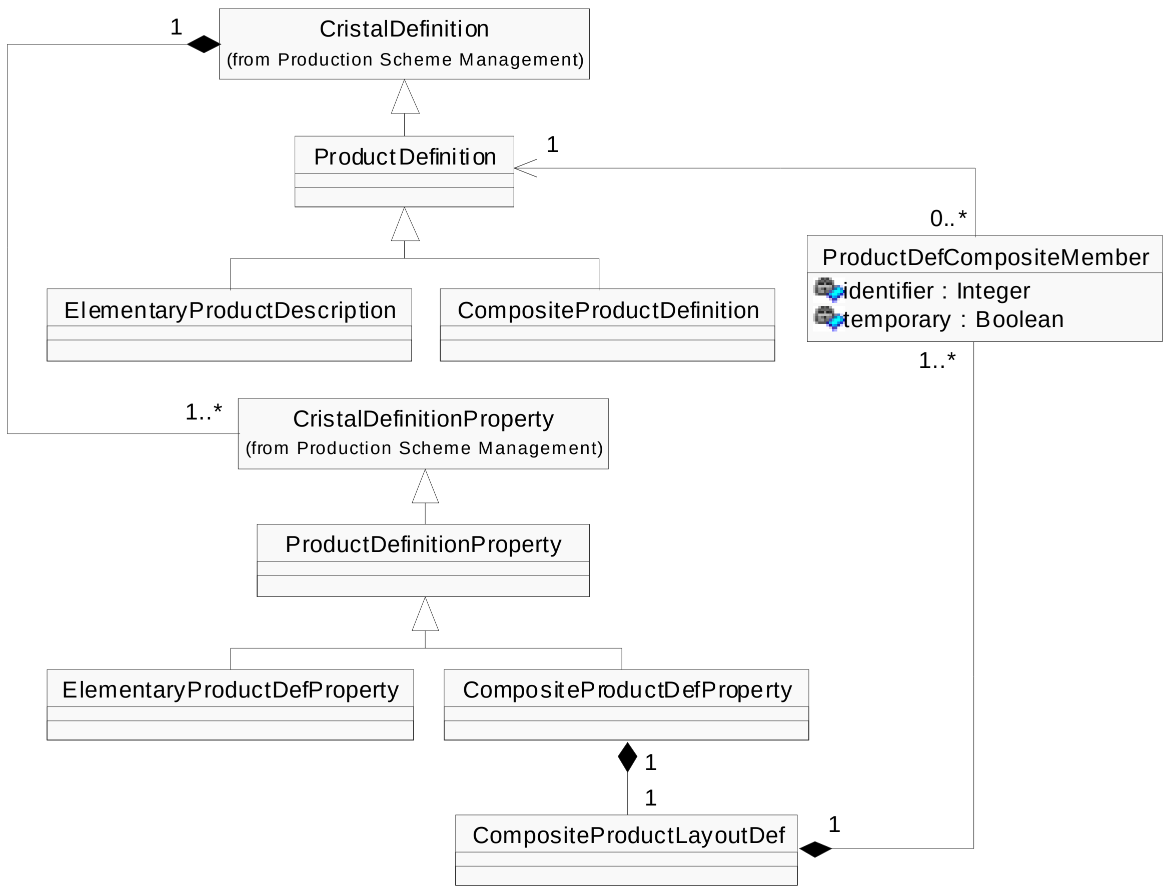

1
CristalDefinition
(from Production Scheme Management)
ProductDefinition
1
0..*
ProductDefCompositeMember
identifier : Integer
temporary : Boolean
1..*
ElementaryProductDescription
CompositeProductDefinition
1..*
CristalDefinitionProperty
(from Production Scheme Management)
ProductDefinitionProperty
ElementaryProductDefProperty
CompositeProductDefProperty
1
1
CompositeProductLayoutDef
1

Figure 5-5. Product Graph Meta-Model package

The Enriched Composite Item Description pattern emerges from this package in that:

- A ProductDefinition can be either Elementary or Composite in nature.
- Composite ProductDefinitions are made up of other ProductDefinitions each of which has a number and a location in the composite i.e. each Composite ProductDefinition has a three-dimensional layout.
- The ProductDefCompositeMember identifies and locates a ProductDefinition in the context of its Composite ProductDefinition. (As a result of the use of a ProductDefCompositeMember object, the separation of PBS from ABS is no longer required in this model).

The conjunction of the Version and Enriched Composite Item Description patterns provides great flexibility in versioning products. Firstly, the layout of a composite product can be amended without changing its composition in a single product version. Secondly, the number of members of a composite product can be changed without affecting the product to which it refers. Thirdly, versioning a product definition does not affect the different composites in which it may be a constituent.

The flexibility in versioning that results from the use of the Version and Enriched Composite Item Description patterns maximises the reusability of ProductDefinition instances and provides a powerful environment for handling complexity. For example the ECAL Barrel subdetector is composed of 36 SuperModules, which are themselves composed of four modules (of different types) and 170 front-end electronic units, each containing 10 channels. To define this set-up the following seven product definitions are required:

- SuperModule
- ModuleType1, ModuleType2, ModuleType3 and ModuleType4
- Front-End Electronic Unit
- Channel

These seven product definitions together with the number of their occurrences in each composition will be sufficient to describe this ECAL example and, once computed, to produce a Bill of Materials comprising a tree of four levels and 61,344 nodes (i.e 36 * [4 + 170 * 10] nodes).

The role of the CompositeProductLayoutDef object in this package is to separate the definition of compositions from any physical representations of the constituent products in the composite product. For example, a module of type1 is composed of 50 sub-modules (10 each of five different types). The location of each individual sub-module, in the context of this module, is independent of the position of the module in its SuperModule. The design of the sub-modules can therefore evolve provided that the overall module dimensions remain unchanged. (If the overall dimensions of the module change, then clearly the change must propagate to the SuperModules in which the module is constituent). This provides considerable design flexibility in an environment in which sub-detector detailed specification is research-led and apt to change with time.

5.2.1.3 Workflow Graph Meta-Model Package

This package describes the WBS aspects of a sub-detector as-designed - it holds all details of all versions of the workflow activities and layouts required to construct the sub-detector. Like the PBS/ABS package described above, this package follows the versioning strategy for all CristalDefinitions as described earlier, utilising the Version pattern of Section 3.4.2.3 on page 37, in conjunction with the Enriched Composite Item Description pattern of

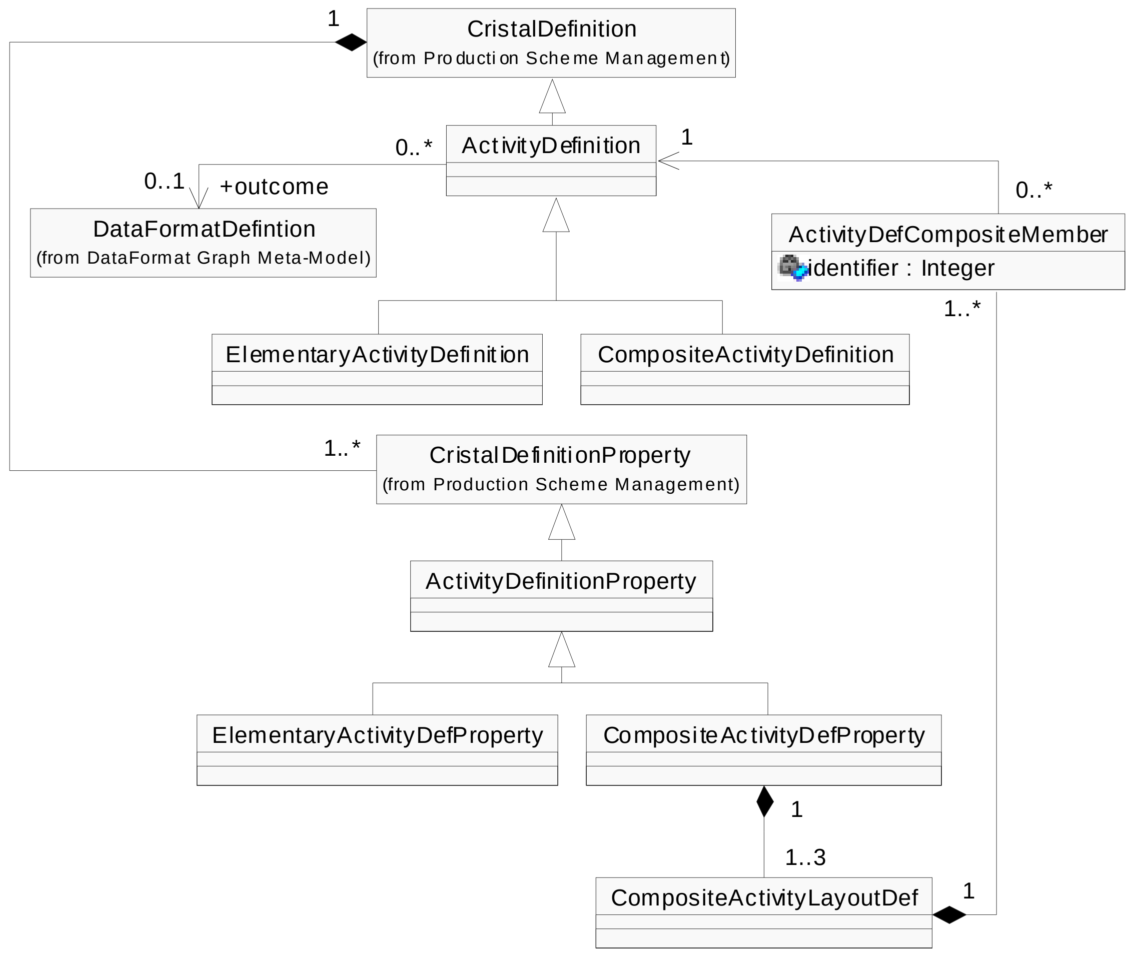

CristalDefinition
(from Production Scheme Management)
1
ActivityDefinition
0..*
1
0..1
+outcome
0..*
DataFormatDefintion
(from DataFormat Graph Meta-Model)
ActivityDefCompositeMember
identifier : Integer
1..*
ElementaryActivityDefinition
CompositeActivityDefinition
1..*
CristalDefinitionProperty
(from Production Scheme Management)
ActivityDefinitionProperty
ElementaryActivityDefProperty
CompositeActivityDefProperty
1
1..3
CompositeActivityLayoutDef
1

Figure 5-6. Workflow Graph Meta-Model package

Section 3.5.1 on page 41. Consequently, exactly the same conclusions can be drawn, from this package, regarding complexity handling and reusability of process definitions.

This package differs from the Product Graph Meta-Model package in the interpretation of the CompositeActivityDefProperties and the CompositeActivityLayoutDefs objects (as opposed to the CompositeProductDefProperties and the CompositeProductLayoutDef objects). Whereas the CompositeProductDefProperties define the composition of a product definition, here the CompositeActivityDefProperties defines the sub-processes which are the constituents in each composite activity. Also the CompositeActivityLayoutDefs of this package describe how the constituent workflow elements (workflow activities, splits and joins) are connected in a particular workflow activity type whereas the CompositeProductLayoutDefs of the Product Graph Meta-Model package describes the position of a product definition in its composite.

5.2.1.4 Production Conditions Package

The Production Conditions package provides for the integration of the Product Graph and Workflow Graph Meta-Model packages. This is achieved by adding a directed association between an instance of a ProductionDefinition to an instance of an ActivityDefinition. This association effectively captures the conditions, in an object called ProductionConditions, required for an instance of the WBS to be applied to a specific instance of the PBS/ABS.

A version of these ProductionConditions (called a ProductionCondProperty, following the Version pattern) comprises StartConditions, EndConditions, ApplicableCentres and Commands. StartConditions are the required individual product instances for the execution of (an instance of) a workflow activity definition on a composite product. For example, a sub-unit is constructed from a crystal and a capsule which are glued together to form the sub-unit. The StartConditions for the gluing process are therefore the instances of the crystal and the capsule. As the result of the execution of a workflow activity on a product a collection of data may be stored. To ratify the outcome of the workflow activity execution, EndConditions (or tolerances) are compared to the collected data. If the data lie within tolerances the following activities in the workflow are enabled, if not the process stops awaiting a decision on the course of action for the product. ApplicableCentres are simply the valid locations for a workflow activity to take place on a product. Commands are used when the executor of the workflow activity is a physical instrument and are the list of instructions required by that instrument for the execution of the workflow activity on a product.

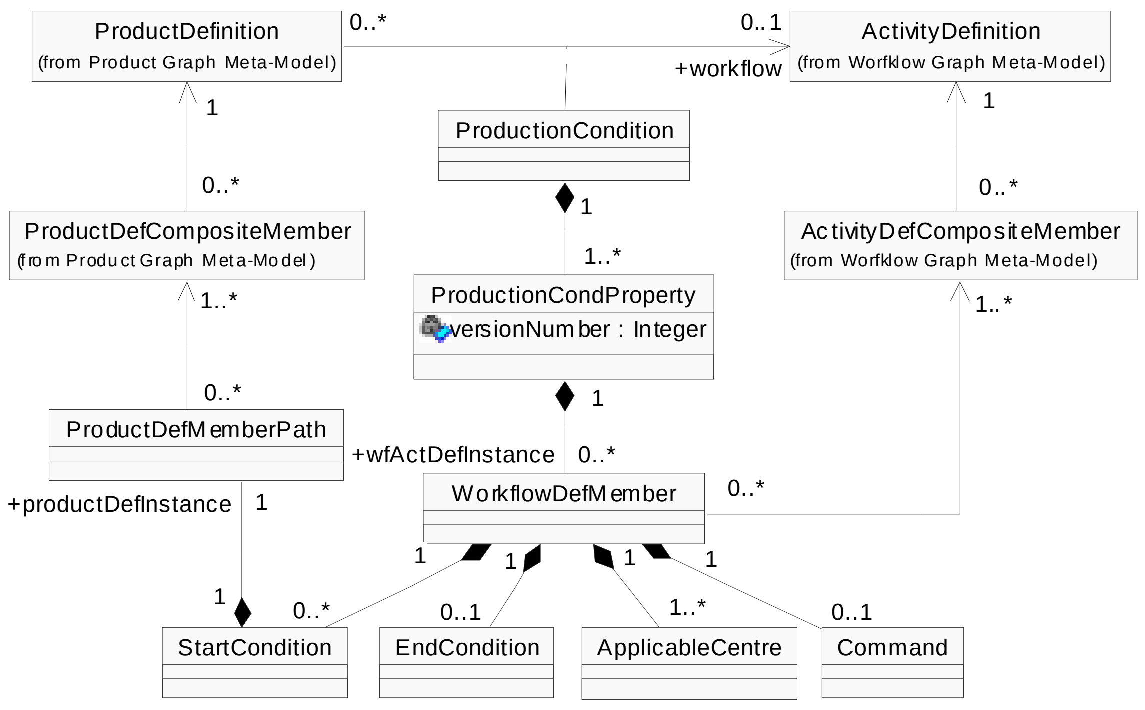


Figure 5-7. Production Conditions package

As can be seen in Figure 5-7, the Production Conditions Package makes heavy use of the CompositeMember construct of the Enriched Composite Item Description pattern (see Figure 3-10 on page 42). The CompositeMember object allows the identification of a specific instance of a product from a collection of those products in a composite product. For example, a sub-module is composed of 10 sub-units each located in a position in a mechanical support structure (called an alveoli). The order in which a particular sub-unit is placed in the sub-module, and the position in which the sub-unit is located in the sub-module is handled by the StartConditions for the ‘Place sub-unit in sub-module’ activity and these conditions refer to individual CompositeMembers. Therefore, it is possible to determine not only the composition of a product but also how it was populated with constituent products. The same principle is used to identify ProductionConditions for the execution of a specific workflow activity on a product when multiple instances of the same activity are constituents of a composite activity. *This use of the Enriched Composite Item Description pattern provides deep integration between the PBS/ABS (or PDM-related) world and the WBS (or WfM-related) world and is one of the major unique contributions of this research.*

The isolation of the production conditions for the execution of (an instance of) a specific workflow activity on an (instance of an) identified product allows for maximising reuse of definitions in the CRISTAL application. For example, in ECAL there are 34 different types of crystals and each crystal type may follow a common workflow process. The model is flexible enough to capture different ProductionConditions for the execution of each instance of a workflow process on each instance of a crystal type.

Furthermore, this isolation allows for asynchronous versioning of product definitions and of workflow activity definitions. That is, product definitions can be versioned separately from workflow activities and connections between asynchronous versions are made only when an instance of a workflow activity definition is applied to an instance of a product definition for construction, according to some ProductionConditions. The detail of this mechanism is explored in later sections of this chapter.

The technique of assigning application-specific semantics to the association between product instances and process instances can be generalised for other applications. For example, the association of a maintenance workflow activity to a product will require quite different conditions to be captured than when the detector was constructed. Also, the association of a cal-

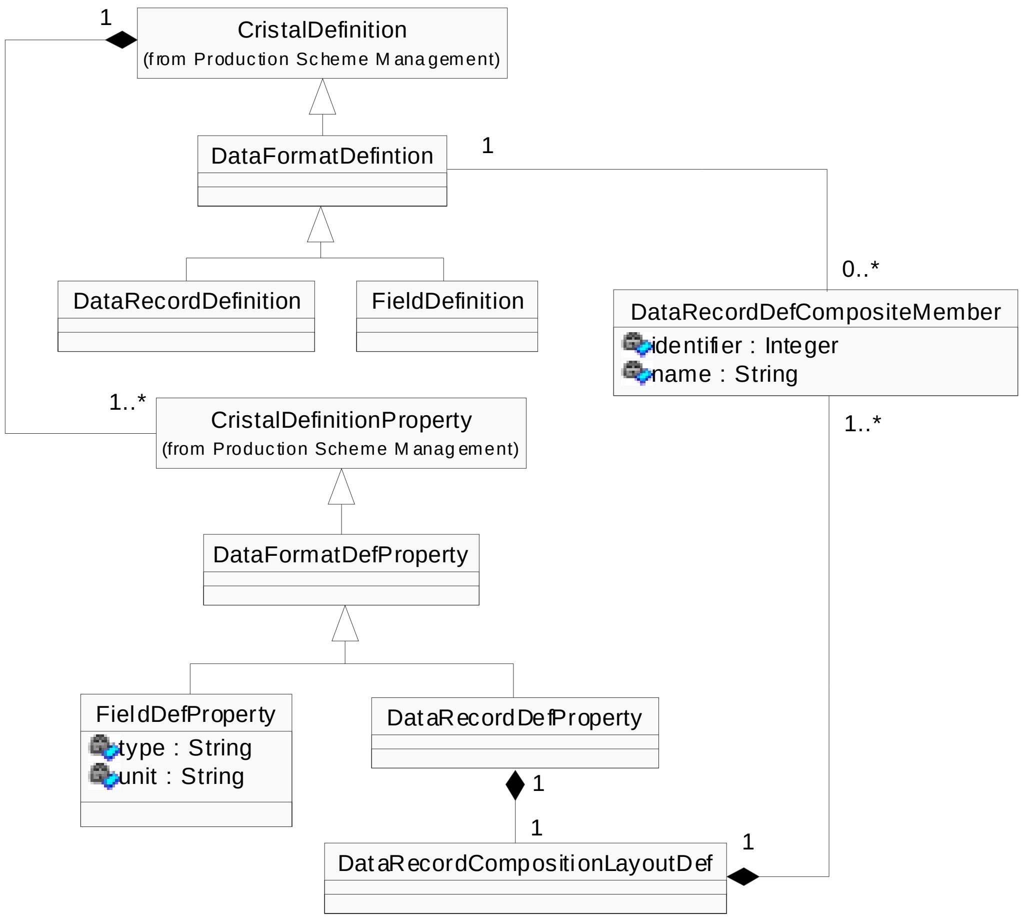


Figure 5-8. DataFormat Meta-Model package

ibration workflow activity to a product would require calibration-specific conditions to be captured. In other words, the identified association between the process and product description worlds carries rich semantics. It allows many other links to be made between aspects of the overall CRISTAL data model: the same mechanism can be used to assign agents to workflow activity definitions for the purposes of enactment or the assignment of agents to product definitions for the purposes of resource management.

In the Homomorphism pattern, of Section 3.4.2.2 on page 37, the semantics in the association between two ItemDescriptions dictates the semantics in the association between two Items. For example, the description of the production conditions which associate a Product Definition with a Workflow Definition will dictate the production conditions to be applied when a Workflow instance is executed on a Product instance.

5.2.1.5 DataFormat Meta-Model Package

This package describes all the data structures that are used in CRISTAL to collect data. Data format definitions (i.e either data record or field definitions) are kinds of CristalDefinitions which can, of course, be versioned. They can also be composite in nature and the CompositeMember construct can be invoked again to uniquely identify members in the composition in exactly the same manner as in the preceding sections. Therefore a further use of the Enriched Composite Item Description and the Version patterns can be seen in the DataFormat Meta-Model package. Furthermore, use of the Enriched Composite Item Description pattern provides an abstraction of the familiar C++ typedef structure, since a single FieldDefProperty can be used with different names in the same data format composition.

5.2.2 CRISTAL Model Layer

Just as for the CRISTAL Meta-Model, the CRISTAL model has been partitioned into packages: the Product Tree Model and Workflow Tree Model packages and the Production Management package, which are explained in detail in the following sections. As explained in Section 3.2 on page 30 and as shown in Figure 5-1, the CRISTAL Model is derived from the CRISTAL Meta-Model through the use of the Item Description pattern.

Section 3.4.2.2 on page 37 detailed the Homomorphism pattern and indicated that it is used in the generation of a model from its meta-model. In Figure 5-9 the Homomorphism pattern emerges in that the Product and Workflow Graph Meta-Model packages represent the Item Description elements and the Product and Workflow Tree Model packages represent the Item elements (see Figure 3-4 on page 36). Similarly, the Production Conditions packages represents the ConditionDesc element of this figure and the Production Management package represents its Condition element. It is the responsibility of the ProductionManagement package to establish when versions of product and workflow activities should be evolved and to carry out that evolution.

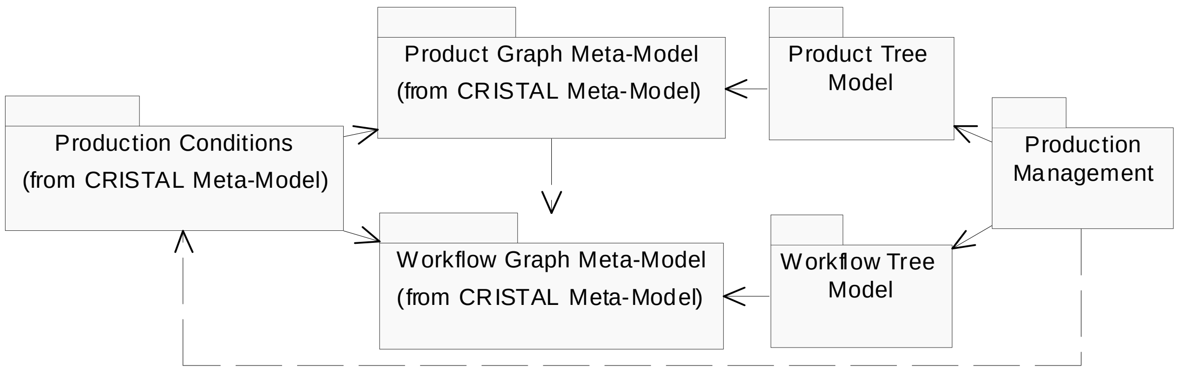


Figure 5-9. The relationship between the CRISTAL Meta-Model and Model.

5.2.2.1 Product Tree Model

The Product Tree Model package is constructed from the Product Graph Meta-Model package by the navigation of an Enriched Composite Item Description pattern. During construction of the product tree, product nodes are linked to parent nodes by firstly associating Products with their Product Definitions, secondly by locating the associated parents in the graph and the parents' corresponding node locations in the product tree and finally by linking a selected parent node to the original product node.

Consequently the detector 'as-built' tree is constructed from the detector 'as-designed' graph and the 'Bill Of materials' required for engineering purposes results from the navigation of

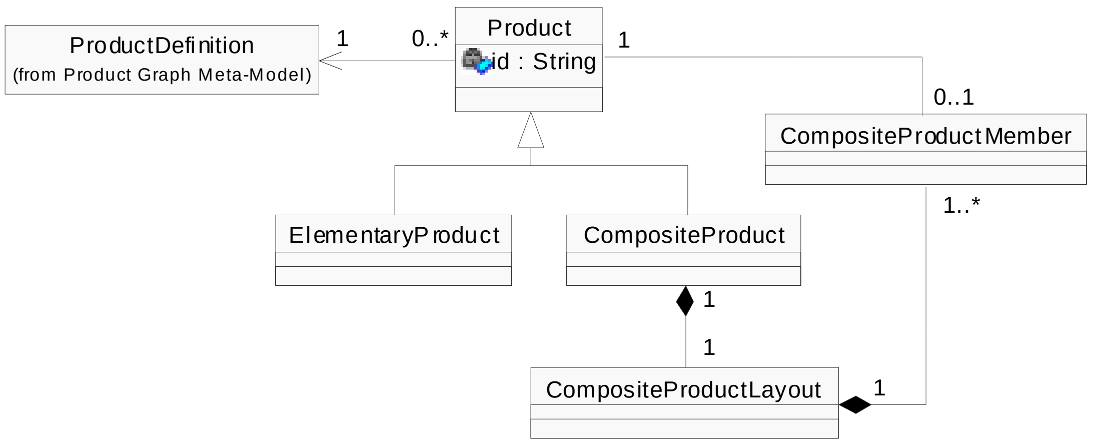


Figure 5-10. Product Tree model

the graph and from instantiating and executing the tree construction process. The Tree pattern that emerges is, in fact, an enriched tree (see Figure 5-10) and holds more semantics than the complex tree of Figure 3-3 on page 35 in that, as was found for a normal tree:

- A Product can be either Elementary or Composite in nature.
- A CompositeProduct is made up of other Products each of which has a number and a location in the composite i.e. each CompositeProduct has a three-dimensional layout.

but in addition:

- The CompositeProductMember identifies and locates a Product in the context of its Composite Product.

The role of the CompositeProductLayout object in this package is to capture the physical representations of the constituent products in the composite product.

Since the 'as-built' tree is established only during construction when products are assigned to composite products, there is, at any one time, only a single active version of the product composition. This version of the product composition is determined when the composite product is instantiated according to the most recent version of its CristalDefinition. Consequently, the Product Tree Model package differs from the Product Graph Meta-Model package since in the latter ProductDefinitionProperties are used for versioning and these are not required in the former. In other words, the meta-model caters for all versions of product definitions, but there is only a single product tree whose nodes can be derived from different versions of the same product definition.

5.2.2.2 Workflow Tree Model

The structure and construction of the Workflow Tree Model package and its relationship with the Workflow Graph Meta-Model package is exactly analogous with those of the Product Tree Model and Product Graph Meta-Model packages. The CompositeActivityLayout object captures the layout of individual activities in their parent (composite) activity - i.e. the layout of CompositeActivityMembers in the corresponding CompositeActivity.

This package differs from the Product Tree Model package in the interpretation of the CompositeActivityMember and the CompositeActivityLayout objects (compared to the CompositeProductMember and the CompositeProductLayout objects). Whereas the CompositeProductMember defined the composition of a product, here the CompositeActivityMember defines the sub-processes which are the constituents in each composite workflow

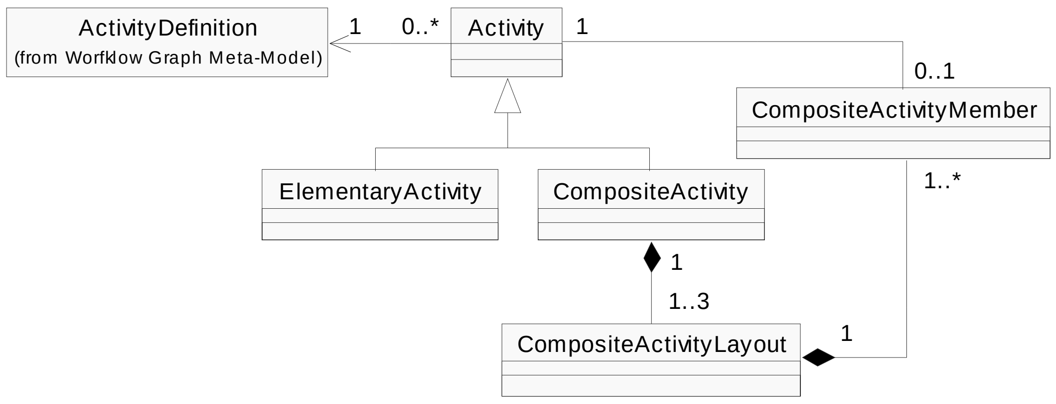


Figure 5-11. Workflow Tree Model package

activity. Also the CompositeActivityLayout of this package describes how the constituent workflow elements (workflow activities, splits and joins) are connected in a particular workflow activity whereas the CompositeProductLayout of the Product Tree Model package describes the position of a product in its composite.

As with the Product Tree Model there is no need for explicit versioning objects to be shown on the Workflow Tree Model. However, the workflow tree is not established at construction time (as was the case for the product tree) rather it is established when products are first instantiated and is thereafter amended. This amendment happens when it is deemed appropriate for a new version of the workflow to become active. This is carried out by the ProductionManagement package which has information on the current state of both products and activities and can determine when a new version of an activity should be invoked, as explained in the following section.

5.2.2.3 Production Management Package

The Production Management package carries out the physical integration of products and workflows. A product and its associated workflow activities manage their own 'state' with records being stored in a database of the progress a product makes through its workflow. The product and workflow management is kept separate and only a Product Manager (PM) object performs their integration. The PM uses the present state of products and workflow activities with the prevailing production conditions to determine the next step that a product takes in its workflow.

In effect the Product Manager acts like a mediator (as defined in the Mediator pattern by Gamma et al. (1995)) liaising with both workflow activity and product objects, which, themselves, have no knowledge about each other's state. By using the Mediator pattern, workflow

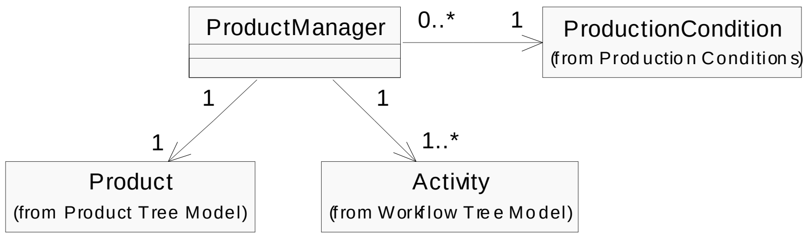

ProductManager
0..*
1
ProductionCondition
(from Production Conditions)
1
1
1
1..*
Product
(from Product Tree Model)
Activity
(from Workflow Tree Model)

Figure 5-12. Production Management Package

activity objects are decoupled from product objects thereby allowing interaction with these objects to be handled separately through a single 'mediator' object. By so doing the protocol for interaction with both workflow activity and product objects is simplified and only implemented through the mediator object.

Figure 5-13 shows an enriched Homomorphism pattern replacing the Condition element of Figure 3-4 on page 36 by a Mediator class. This enriched pattern describes both the relationship between the Meta-Model and Model layers of CRISTAL, through the familiar Item Description pattern and the role of the PM as the mediator between Items of the Model layer (in this case between products and workflow activities).

The role of the CRISTAL execution component introduced in the three-layer model of Figure 5-1 is to manage Items which have been instantiated from ItemDescriptions, using the Item Description pattern. The PM mediates between Items, as described above, and therefore it acts as the CRISTAL execution component in the multi-layer model.

5.3 Conclusions

The design of the CRISTAL prototype was dictated by the requirements for adaptability over extended time scales, for schema evolution, for interoperability and for complexity handling and reusability. These constraints meant that a description-driven solution was required. In

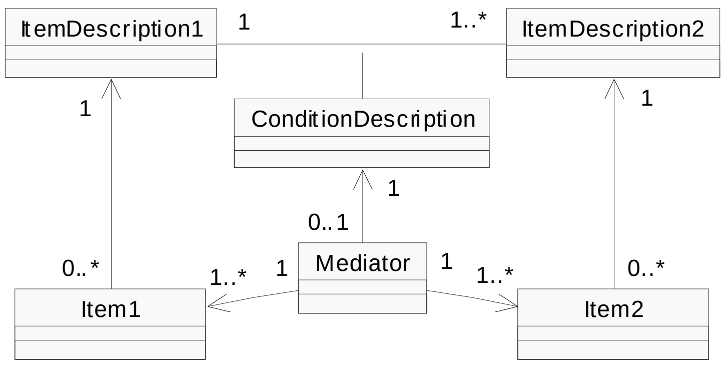


Figure 5-13. Enriched Homomorphism pattern

adopting a description-driven design approach, a separation of object instances from object descriptions instances was needed. This abstraction resulted in the delivery of a meta-model as well as a model for CRISTAL. Having completed the data model for the description-driven CRISTAL system, including meta-objects, the resulting structures were compared with those proposed by the patterns research community. As a result existing design patterns could be enriched and new patterns proposed. These patterns are sufficient to cater for the integration of PDM and WfM.

In conclusion to this chapter, it is apparent that the meta-object or description-driven approach handles system complexity by promoting object reuse, i.e. the same meta-object can be used to build different compositions, and by translating complex hierarchies of object instances into graphs of object definitions. A meta-model of the schema can be stored in the database which describes the actual objects and allows changes to be made without the need to alter the database schema itself. This also makes it possible to store different versions of objects concurrently in the same database. A model can be derived from this meta-model which is sufficient to perform the PDM-WfM integration. It is a reflection on the power of the data model produced to provide this functionality that only a very small set of (enriched) design patterns are required to specify the complete CRISTAL meta-model and instantiated model.

6 Prototype Implementation

6.1 Introduction

Designing and implementing large-scale software systems, especially for the research community, is often best achieved through the efforts of a small, well-motivated group of software engineers working to tight deadlines and in close contact with the prospective end-users of the system. The prototype studied in the present work was built by a group of software engineers and evolved over a period of months after being designed and coded over a period of 30 months. This chapter describes the work undertaken in implementing functioning prototypes of the CRISTAL system and identifies the contribution of the author in this work. It also describes some of the technologies used in implementing the prototype, describes its architecture and how it was evaluated and tested with the end users and closes with a discussion on the relevance of the prototype to the user community.

6.2 Project Organisation

The user motivation for the CRISTAL development came from a CMS Technical Note (Lebeau, Lecoq and Vialle, 1995) in which the need for a database was identified ‘to keep the history of each (assembled CMS ECAL detector) unit, to have a very high degree of security...and to facilitate all the necessary selections and analyses of stored data’. The database envisaged was required to be distributed geographically over several continents, to be up to 1 Terabyte in size, to provide schema evolution facilities, to be object-oriented in nature and to be accessible with rapid response over time scales lasting several years to physicists who may not be experts in computing. In addition, the database system had to be interfaced with an automatic measuring instrument, which performed a series of defined tasks on ECAL detector components (Peigneux et al., 1997). As no such database system had been employed hitherto by physicists, the user requirements for system specification were difficult to elicit. Furthermore, the extended time scales of the project required that the system be capable of repeated updating and constant review. Flexibility was clearly the central principle on which the system was to be developed and close liaison with the end-users was paramount in view of the speculative nature of its development.

The final CRISTAL prototype system was analysed, designed and developed over a period of three years (from early 1996 to 1999) by a group of engineers working at CERN, Geneva and, nearby, at LAPP, Annecy. The group was, on average, made up of seven members during this period of time: a project leader, a software designer, two experienced software engineers and three students. A further two developers worked part-time on developing the system. The author was involved from the early stages of system analysis, through detailed system design, technology evaluation and system architecture specification to the final implementation and coding of a subset of the prototype system.

The author was largely responsible for developing the meta-object design which underpins the CRISTAL data model. The author introduced the idea of integrating the product and process model using common design artifacts (or ‘patterns’) and specified a series of packages, as described in Chapter 5, for the CRISTAL Data Model. The author also conducted an evaluation of object-oriented technologies before advising on a set of products and approaches which were adopted in the architectural design of the CRISTAL prototypes. In particular, the author evaluated the CORBA and ODBMS standards. The author was involved in verifying each major design decision in the implementation of CRISTAL and in the selection of suitable enabling technology. As a consequence of this it was appropriate that the author should have been responsible for the implementation of much of the prototype code including the Product Manager, the Data Duplication Manager, the Instrument Agent and the Product Browser. The practical implementation of the Product Manager embodies aspects both of product data management and workflow management and is the vehicle for their integration in CRISTAL. Its delivery is central to the operation of the CRISTAL prototype and its functionality is detailed later in this chapter.

6.3 Underlying Technologies

The CRISTAL system uses industry-standard commercial products whenever possible to minimise support problems and to facilitate systems integration. Further functionality is provided, where needed, in a manner which maximises the flexibility of the prototype. The products and workflow activity data are stored in a commercial object-oriented database product, Objectivity (Objectivity, 1998). Orbix (Orbix, 1998), an implementation of CORBA (Common Object Request Broker Architecture, OMG 1992a), is used for communication between the different kinds of distributed CRISTAL components, e.g. the database, the instruments which automatically measure the properties of sub-detector parts. Users access CRISTAL through Java (Java, 1998) GUI applications.

6.3.1 CORBA

6.3.1.1 The CORBA Standard from OMG

The Common Object Request Broker Architecture (CORBA), is the Object Management Group's answer to the need for interoperability among the rapidly proliferating number of hardware and software products available today. Simply stated, CORBA allows applications to communicate with one another no matter where they are located or who has designed them (see Figure 6-1). CORBA 1.1 was introduced in 1991 by the Object Management Group (OMG) and defined the Interface Definition Language (IDL) and the Application Programming Interfaces (API) that enable client/server object interaction within a specific implementation of an Object Request Broker (ORB). CORBA 2.0, adopted in December of 1994, defines true interoperability by specifying how ORBs from different vendors can interoperate.

The (ORB) is the middleware that establishes client-server relationships between objects. Using an ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request, pass it parameters, invoke its method, and

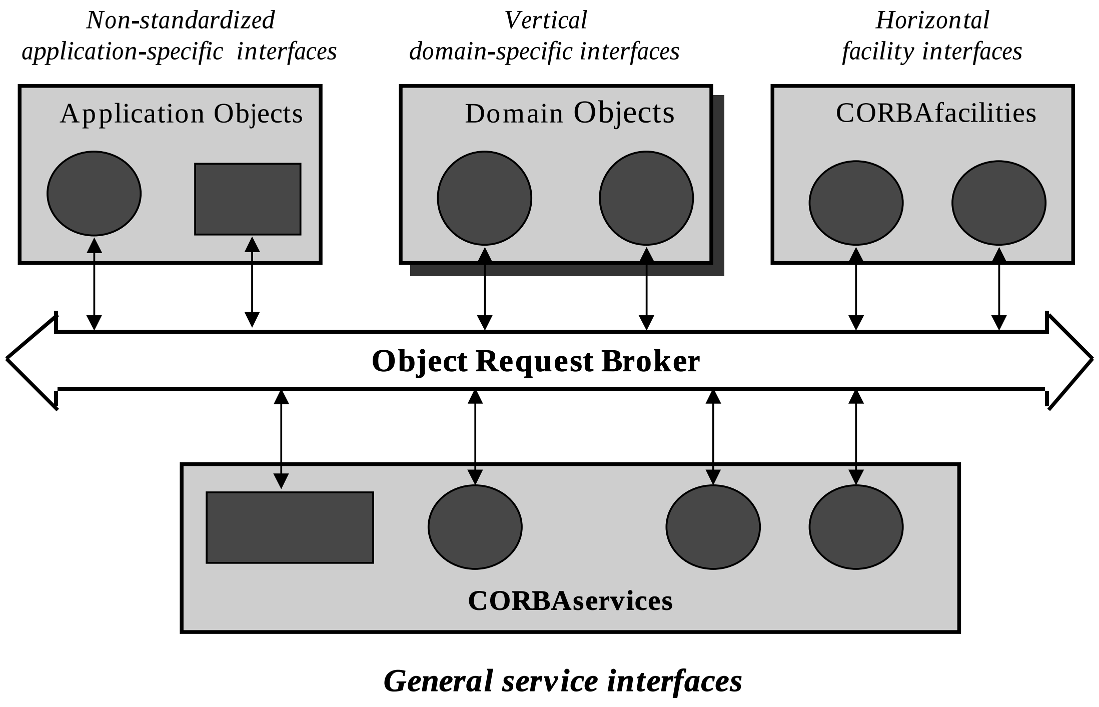

Non-standardized application-specific interfaces
Vertical domain-specific interfaces
Horizontal facility interfaces
Application Objects
Domain Objects
CORBAfacilities
Object Request Broker
CORBAservices
General service interfaces

Figure 6-1. The OMG Object Management Architecture (OMA)

return the results. The client does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of an object's interface. In so doing, the ORB provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems.

In fielding typical client/server applications, developers use their own design or a recognized standard to define the protocol to be used between the devices. Protocol definition depends on the implementation language, network transport and a dozen other factors. ORBs simplify this process. With an ORB, the protocol is defined through the application interfaces via a single implementation language-independent specification, the Interface Definition Language (IDL). ORBs provide flexibility - they let programmers choose the most appropriate operating system, execution environment and even programming language to use for each component of a system under construction. More importantly, they allow the integration of existing components. In an ORB-based solution, developers simply model the legacy component using the same IDL that they use for creating new objects, then write 'wrapper' code that translates between the standardized bus and the legacy interfaces.

CORBA is a step on the road to object-oriented standardization and interoperability. With CORBA, users gain access to information transparently, without them having to know what software or hardware platform it resides on or where it is located on an enterprises' network. Being the communications "heart" of object-oriented systems, it is claimed that CORBA brings true interoperability to today's computing environment.

The key to understanding the structure of the CORBA architecture is the Reference Model, which consists of the following components:

- the ORB, which enables objects to transparently make and receive requests and responses in a distributed environment. The ORB is the foundation for building applications from distributed objects and for interoperability between applications in heterogeneous and homogeneous environments (OMG, 1992a).
- Object Services, a collection of services (interfaces and objects) that support basic functions for using and implementing objects. Services are necessary to construct any distributed application and are always independent of the application domains. For example, the Life Cycle Service defines conventions for creating, deleting, copying and moving

objects; it does not dictate how the objects are implemented in an application. A specification of an Object Service usually consists of a set of interfaces and a description of the service's behaviour. The syntax used to specify the interfaces is the OMG Interface Definition Language (IDL). The semantics that specify a service's behaviour are, in general, expressed in terms of the OMG Object Model. The OMG Object Model is based on objects, operations, types and subtypes. It provides a standard, commonly understood set of terms with which to describe a service's behaviour. The following Object Services specifications are now available: Naming, Event, Persistent Object, Life Cycle, Concurrent Control, Externalization, Relationship, Transaction, Query, Licensing Property, Time, Security, Trading, Collections. Details about these services can be found in OMG, (1994).

- Common Facilities, a collection of services that many application may share, but which are not as fundamental as the Object Services. For instance, a system management or electronic mail could be classified as a Common Facility. Information about Common Facilities can be found in CORBAfacilities: Common Facilities Architecture (OMG, 1993).
- Application Objects, which are products of a single vendor or in-house development which control their interfaces. Application Objects correspond to the traditional notion of applications, so they are not standardized by OMG. Instead, Application Objects constitute the uppermost layer of the Reference Model.

6.3.1.2 Orbix: a CORBA implementation for the CRISTAL prototype

Orbix from IONA technologies is the most popular implementation of CORBA. The Orbix product family offers seamless standards-based integration of software from desktop to server, mainframe to web browser. Orbix and OrbixWeb are C++ and Java-based CORBA application development and deployment products. Orbix comprises all the CORBA functionality, as well as Microsoft COM integration, naming services, and firewall traversal functionality (Orbix, 1998).

Most of the CORBA Object Services have not been implemented by IONA. In particular at the time the CRISTAL prototype was designed, the event, relationship and lifecycle services were not supported by Orbix. Consequently, only the Orbix ORB and Naming Service have been used in the prototype implementation.

6.3.2 Object Database Management System

6.3.2.1 The ODMG standard

According to the Object Database Management Group (ODMG) an Object DataBase Management System (ODBMS) is a database management system that integrates database capabilities with object-oriented programming language capabilities (Cattell, 1994). An ODBMS makes database objects appear as programming language objects, in one or more existing programming languages. The ODBMS extends the language with transparently persistent data, concurrency control, data recovery, associative queries, and other database capabilities.

The primary goal of ODMG is to put forward a set of standards allowing an ODBMS customer to write portable applications, i.e., applications that could run on more than one ODBMS product. The data schema, programming language binding, and data manipulation and query languages must be portable.

The major components of the ODMG architecture are:

- Object Model: The common data model to be supported by ODBMSs. ODMG has used the OMG Object Model as the basis for this model. The OMG core model was designed to be a common denominator for object request brokers, object database systems, object programming languages, and other applications. In keeping with the OMG Architecture, ODMG has designed an ODBMS profile for their model, adding components (e.g., relationships) to the OMG core object model to support their needs.
- Object Specification Languages: One is the object definition language, or ODL, to distinguish it from traditional database data definition languages, or DDLs. ODMG uses the OMG interface definition language (IDL) as the basis for ODL syntax. Release 2.0 adds another language, the object interchange format, or OIF, which can be used to exchange objects between databases, provide database documentation, or drive database test suites.
- Object Query Language: ODBMS defines a declarative (nonprocedural) language for querying and updating database objects. The relational standard SQL has been used as the basis for OQL, where possible, though OQL supports more powerful capabilities.
- C++ Language Binding: ODMG specifies how to write portable C++ code that manipulates persistent objects. This is called the C++ OML, or object manipulation language. The C++ binding also includes a version of the ODL that uses C++ syntax, a mechanism to invoke OQL, and procedures for operations on data-bases and transactions.

- Smalltalk Language Binding. ODMG defines the binding in terms of the mapping between ODL and Smalltalk, which is based on the OMG Smalltalk binding for IDL. The Smalltalk binding also includes a mechanism to invoke OQL, and procedures for operations on databases and transactions.
- Java Language Binding.ODMG defines the binding between the ODMG Object Model (ODL and OML) and the Java programming language as defined by Version 1.1 of the Java_ Language Specification. The Java language binding also includes a mechanism to invoke OQL, and procedures for operations on databases and transactions.

With an ODMG compliant ODBMS it is possible to read and write the same database from C++, Smalltalk, and Java, as long as the programmer stays within the common subset of supported data types. Unlike SQL, ODBMS data manipulation languages are tailored to specific application programming languages, in order to provide a single, integrated environment for programming and data manipulation.

6.3.2.2 Objectivity: an ODBMS for the prototype implementation

Objectivity/DB is a distributed ODBMS which manages transparently data to high-end applications (Objectivity, 1998). Objectivity/DB integrates easily with application software and allows users to directly store and manage objects through standard language interfaces including C++, Smalltalk and SQL using traditional programming techniques and tools. Objectivity/DB has recently proposed a limited Java binding which they intend to extend and improve in the next releases of their software.

Objectivity/DB partially complies to the ODMG standard and the current 5.1 version handles data replication across Local Area Networks (LAN). The choice of Objectivity/DB for the prototype implementation has been dictated by the need to conform to the CERN standardisation procedure.

6.4 Prototype System

6.4.1 Overall System Architecture

Essentially the CRISTAL system provides the ability to record and track changes in product and workflow definitions for a large-scale engineering environment and for versions of these definitions to be supplied to remote centres for instantiation. It also provides the ability to execute instances of these definitions at centres and to record the outcomes of the associated workflow activities (McClatchey et al., 1997). In addition, the system logs a history of

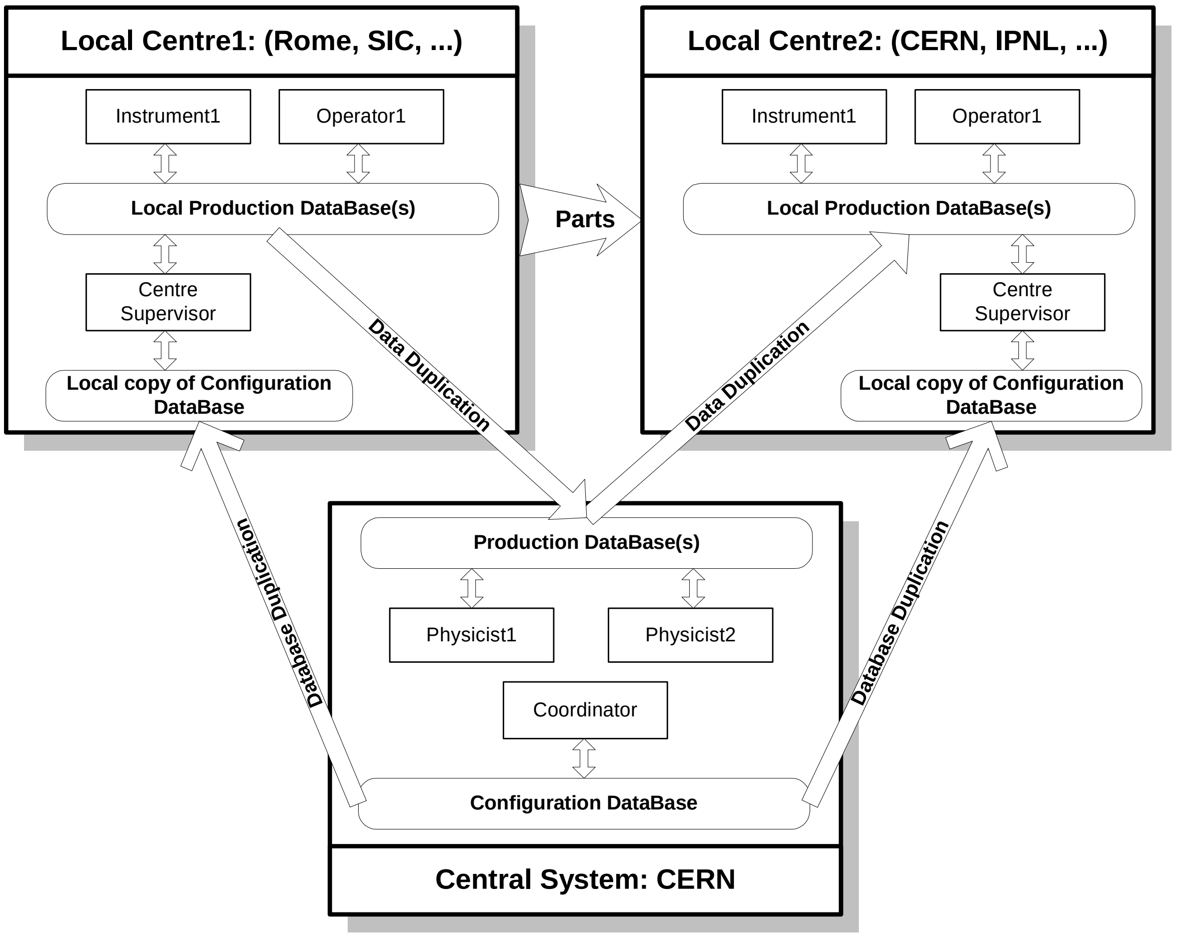


Figure 6-2. Multi-Centre Architecture of CRIStAL

'events' (or significant occurrences) as they occur for each individual product instantiation. All product data that is accumulated at the centres is highly volatile in nature. In other words, workflows are often executed 'once-off' and cannot be repeated or undone. The system must therefore provide reliable local capture of data and replication of that data to a secure central store. The central store therefore provides a snapshot of the status of the sub-detector construction. Furthermore, to optimise technical resources centres have responsibilities for carrying out specific tasks - workflows can therefore be split between centres for execution and products must accordingly be moved between centres. Each centre must be capable of standalone data gathering, so that overall production is not dependent on inter-centre networks, and each centre must be provided with a minimal set of resources (e.g disk space, memory, CPU, instruments) to minimise centre costs.

A CRISTAL system comprises one or more data gathering centres, each of which is *federated* into the system (Bazan et al., 1998). These centres are a (single) Central System and (one or more) Local Centres in which the CRISTAL software will run. CRISTAL software runs at every centre (either production, assembly or test centres). It is not feasible to store all of the data redundantly at all sites since the approximate size of the main repository at CERN will total around 1 TB. However, production sites also have local repositories which must store only the data belonging to the products that are currently located at that Centre (see Figure 6-2). When products are shipped, the associated data migrate between the sites.

The CRISTAL system uses a set of roles (e.g. Coordinator, Centre Supervisor, Operator, Physicist) to define user access to the software and data. The overall system Coordinator, located at the Central System, provides the product and workflow definitions for a version of the sub-detector construction scheme. New versions of product and workflow definitions are defined centrally at CERN and are farmed out to each Centre by the Coordinator. Different centres of one sub-detector are coupled very tightly because they are part of the same construction scheme specification. If two centres can carry out the same workflow activity on the same product type the construction scheme specifications must be identical since that is the only way to ensure the integrity of the sub-detector. It is for this reason that releases of the construction scheme specification are provided centrally and can be edited by only one person (the Coordinator) at a given time.

The sub-detector construction scheme is supplied to each Local Centre where it can be applied by the (local) Centre Supervisor. Products of a given definition follow a series of workflow activities (of a given definition) which are performed by workflow executors. Each Local Centre will have a set of Instruments defined in the database in terms of the commands that each instrument uses and the data formats expected as outcomes from the execution of workflow activities by instruments (Auffray et al., 1998). Workflows are executed either by Instruments, by Operators or by automatically launched User-Provided Code. Should the outcome of a workflow execution be non-conformant then the Centre Superviser is responsible for taking appropriate exceptional action (such as product rejection, workflow activity re-execution or schedule a specific ad-hoc workflow). Physicists (who may be distributed geographically) access only the Central System and perform analyses on the accumulated product-specific data that have been gathered (at outlying centres) during the construction process. All workflow executors (e.g. Operators) have, consequently, a process-centred view of production whereas Physicists have a product-centred view of production and the system must provide the ability to view construction from both standpoints.

Figure 6-2 also shows the database architecture selected for CRISTAL. It is based on the federated database concept in which multiple autonomous databases co-operate. The Central database holds the federation configuration files and any changes in the schema take place centrally and are dispatched to the Centres via a small, specification database (*the Configuration DB*). Remote Centres must also be allowed to continue data gathering even when the network connection to the central database is down. Data is duplicated from the Centres to the central database when network connections allow. Once written into the central database, the data is thereafter extracted by read-only processes running centrally. Data extraction takes place only at the central database. One example of accessing the construction database is for the extraction of sub-detector calibration data.

6.4.2 Local Centre Architecture

Figure 6-3 shows the software architecture of a Local Centre. The software comprises a set of Instruments and Instrument Agents, a set of User-Provided Code Agents, a set of Product Managers for handling all data to/from the database, a Local Centre Manager (LCM) which supervises the data gathering in a centre, a set of Desktop Control Panels (DCPs) which handle user interaction with the system and a Data Duplication Manager which handles all duplication of data between the Local Centre and the Central System (for secure back-up). The Desktop Control Panels (DCP) are interpreted Java (Java, 1998) code, which provide the user interface to CRISTAL (see Figure 6-3). By swiping the barcode of an unique product

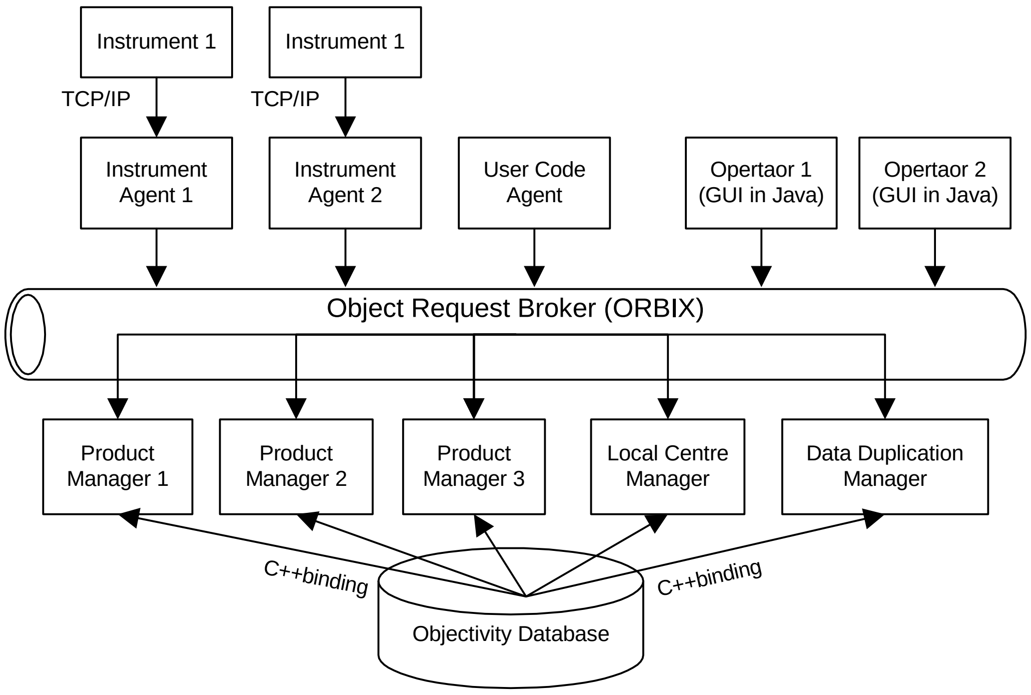


Figure 6-3. The software architecture of a Local Centre

the Operators are guided, via the DCP, through the possible workflow activities that the product can follow and are instructed to carry out the corresponding tasks on the swiped product.

Each instrument in the Centre has an associated Instrument Agent which receives commands from the operator, via the corresponding DCP and Product Manager. The Instrument Agent communication protocol is ASCII based and is implemented either using Orbix or using a sockets interface driver depending on the instrument's computing capabilities. The measurements from the instrument are converted by the Instrument Agent from ASCII into an object-based format, that the Product Manager can use to store in the database.

6.4.3 The Product Manager

Product Managers (PM) provide the mechanism by which products are tracked through workflows. They manage concurrency of workflow activity executions, they manage data storage and carry out all book-keeping of events. Any significant occurrence that happens to a product in a workflow is recorded by its PM which effectively keeps track of the 'state' of the product in production. As the PM handles all database activity for specific products, the DCPs are protected from any changes occurring in the database schema. As a consequence, the PM lies at the heart of the CRISTAL system and controls access to versions of product and process data. It is responsible for the application of any changes to products and workflows resulting from updates of the centrally-defined construction scheme i.e it handles all aspects of dynamic changes in workflow execution.

As stated in Chapter 5 the PM is an implementation of the Mediator pattern of Gamma et al. (1995). In practice, this means that the product (PDM) and process (WfM) information can be managed separately but can be combined by the Mediator process which appears externally as a single entity. Product and process data are stored as database objects, managed by the PM. All events associated with these product and process objects are also stored in a chronological order as database objects (the so-called ProductionEvents of the ProductionPermanentData). The PM interrogates the database for product and process 'state' information and combines this with the ProductionConditions to determine the next viable workflow activities (including 'shipping' requests) that can be performed on the product under consideration. The PM data model is shown in the Production Management package of Figure 6-4.

Not only does the PM manage static product and process information, it also handles evolving product and process information. When a new release of the product specification becomes available at a centre, the PM will compute when and how the change can be applied.

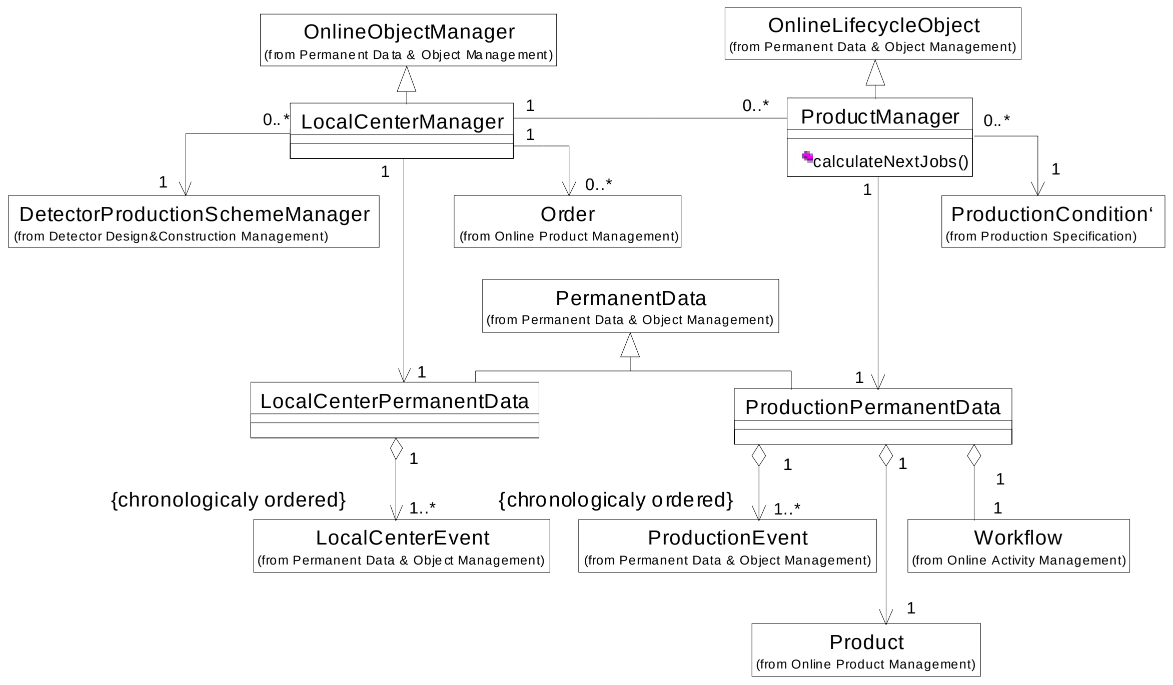

OnlineObjectManager
(from Permanent Data & Object Management)
OnlineLifecycleObject
(from Permanent Data & Object Management)
LocalCenterManager
ProductManager
calculateNextJobs()
DetectorProductionSchemeManager
(from Detector Design&Construction Management)
Order
(from Online Product Management)
ProductionCondition'
(from Production Specification)
PermanentData
(from Permanent Data & Object Management)
LocalCenterPermanentData
ProductionPermanentData
{chronologicaly ordered}
{chronologicaly ordered}
LocalCenterEvent
(from Permanent Data & Object Management)
ProductionEvent
(from Permanent Data & Object Management)
Workflow
(from Online Activity Management)
Product
(from Online Product Management)
0..*
1
1
0..*
0..*
1
1
0..*
1
1
1
1
1
1..*
1
1..*
1
1
1
1

Figure 6-4. The Production Management package, showing the Product Manager.

Changes can be of different types (e.g product, workflow activity, data format) and as a Mediator the PM will forward the handling of the change to the appropriate database object. As a consequence of its role in determining when a change can be applied, it is possible for products of the same definition to be following quite different versions of the production scheme at any one time in a centre.

It is the responsibility of each products's PM to keep track of the product in its workflow. Products can move between centres, so PMs in both the source and destination centres must be able to follow the same workflow i.e to suspend and resume workflow activities when products are shipped between centres. Products are shipped between centres by the issuing and the management of 'Orders'. These orders are specified at the Central System (by the Coordinator) and contain a required number of products (of one product definition) and the source and the destination centre names. Orders are dispatched to centres and managed locally by the LCM. At the source centre orders are filled when the product's PM determines that the next workflow activity cannot be carried out at the source centre and the product needs to be shipped. The production for that product is then suspended and only resumed when a PM at the destination centre determines that the next workflow activity can be carried out at the destination centre. It is the responsibility of the LCM at the destination centre to provide the environment in which suitable PMs are launched, following reception of an order.

The Product Manager has been implemented as a CORBA object using the CORBA lifecycle service. In its role as manager of the Local Centre the LCM manages the life cycle of each PM instance, including when a product is physically transported from one centre to another (‘shipped’).

6.4.4 Prototype Evaluation.

To ensure that the requirements of the CRISTAL user community were being adequately captured and ultimately catered for, it was essential to have user involvement in every stage of the CRISTAL development. Therefore at the outset of the analysis, user commitment was established for analysis, design, implementation and testing. This included, by definition, the evaluation of the system design and of each ‘prototype’ that was constructed.

Due to the research nature of the project, and particularly since new technologies were being evaluated and integrated for CRISTAL, two prototypes were developed. The initial prototype was essentially a ‘throw away’ prototype, developed in the first year of the project with very limited functionality to:

- evaluate the ODBMS and CORBA technologies
- allow training in the use of UML and Java and
- to help in establishing and verifying user requirements.
- to evaluate a ‘rapid prototyping’ design approach.

The first prototype was based on a simplified data model which allowed the description of products and the capture of product-related characteristics but did not cater for WfM capabilities. A basic UML model was constructed and a ‘rough and ready’ system was rapidly implemented. The prototype was built using Objectivity V4.0 and Orbix Version 2.1 and programming was in C++ and Java JDK 1.1. It was successful in terms of providing the necessary experience in the use of UML, Java, Orbix and Objectivity and in providing a tool for exploring detailed user requirements. The first prototype established a set of basic user requirements (both mandatory and desirable) which were then used as the starting-point for the development of the later, working prototype.

The later prototype was developed over the second and third years of the project and was delivered as a series of evolving versions. By this time the enabling technologies were well understood and the required user functionality could be implemented in a step-wise fashion. In delivering these prototype versions, a form of Spiral Model (Boehm 1988) development approach was followed in that the users were consulted for new requirements and a risk analysis was carried out prior to the development of each new prototype version. This working prototype was then tested by the users and further improvements emerged through evaluation of its functionality. These new requirements provided the input to the next cycle of prototype development (or the next loop in the spiral model). This iterative or evolutionary development process continued until there was convergence of the prototype, after a number of prototype versions, with the required and desirable functions of the user environment. The final prototype was delivered early in 1999 to the ECAL user community. The prototype was built with Objectivity version 5.1, Orbix Version 2.3 and Java JDK 1.1.7 and Swing.

6.4.5 Prototype Usage and Tests

As proof of the prototype, a number of ECAL detector crystals were made available for characterisation (i.e. measurement, testing and pre-assembly). This sample of over 400 pre-production crystals was used to evaluate the functionality and performance of the final prototype. Over a period of three months, these crystals were characterised while the prototype evolved through several development cycles (or loops in the Spiral Model). In total eight versions of amended process specifications were released during this period. Each release provided greater functionality and/or greater performance than the previous release as demanded by the user community. Due to the description-driven nature of the prototype, the design of the system was robust enough to cope with the rapid prototyping that resulted from evolving user requirements, which naturally emerged as the prototype was used in earnest.

Allowing description (of products) to be captured and managed separately from instances of those descriptions has provided the complexity handling required by the user community. The prototype system had to handle only 400 product-specifications to describe the detector ‘as-designed’ rather than the 250,000 products that would otherwise have to have been captured by a conventional PDM tool in order to provide the bill of materials. In addition, the description-driven nature of the prototype has facilitated the reuse of product and workflow definitions. For example, workflow activity definitions have been defined once in the CRI-

STAL repository and reused repetitively for a single crystal type and over multiple crystal types. In fact, there are 34 crystal types and one definition of, for example, the Characterisation workflow activity, has been repeated both in a workflow for one crystal type, e.g crystal 6Left, and reused over the 34 crystal types.

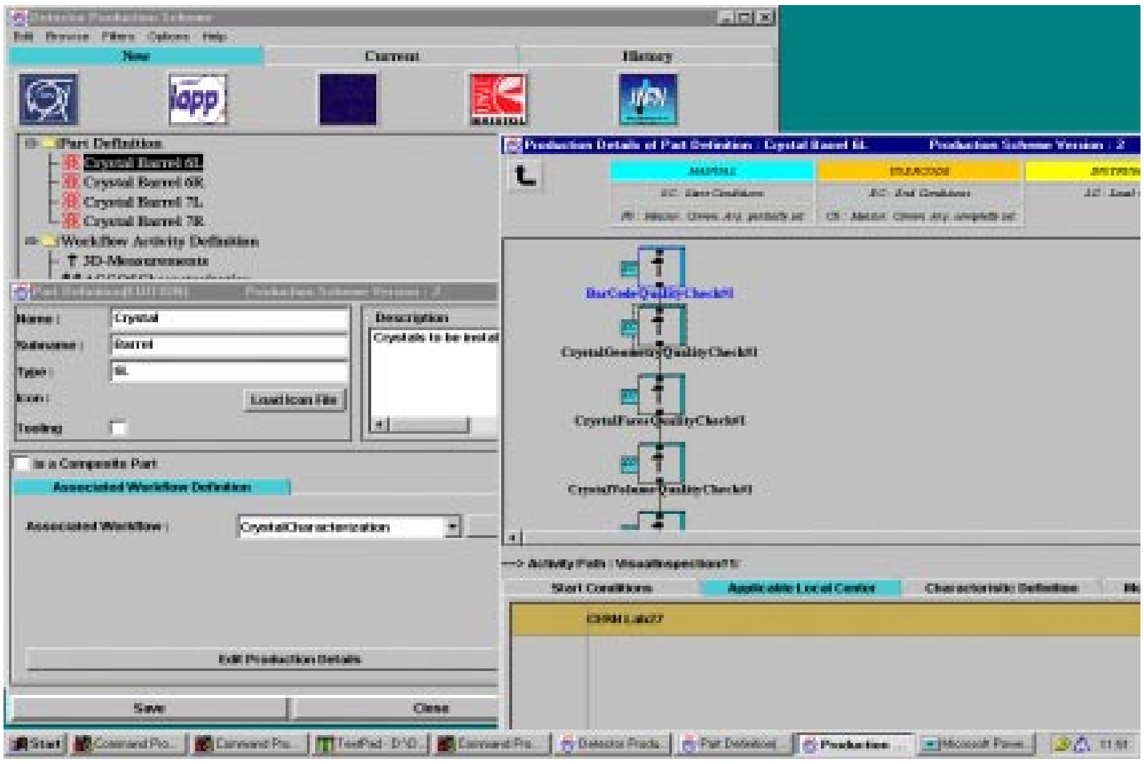
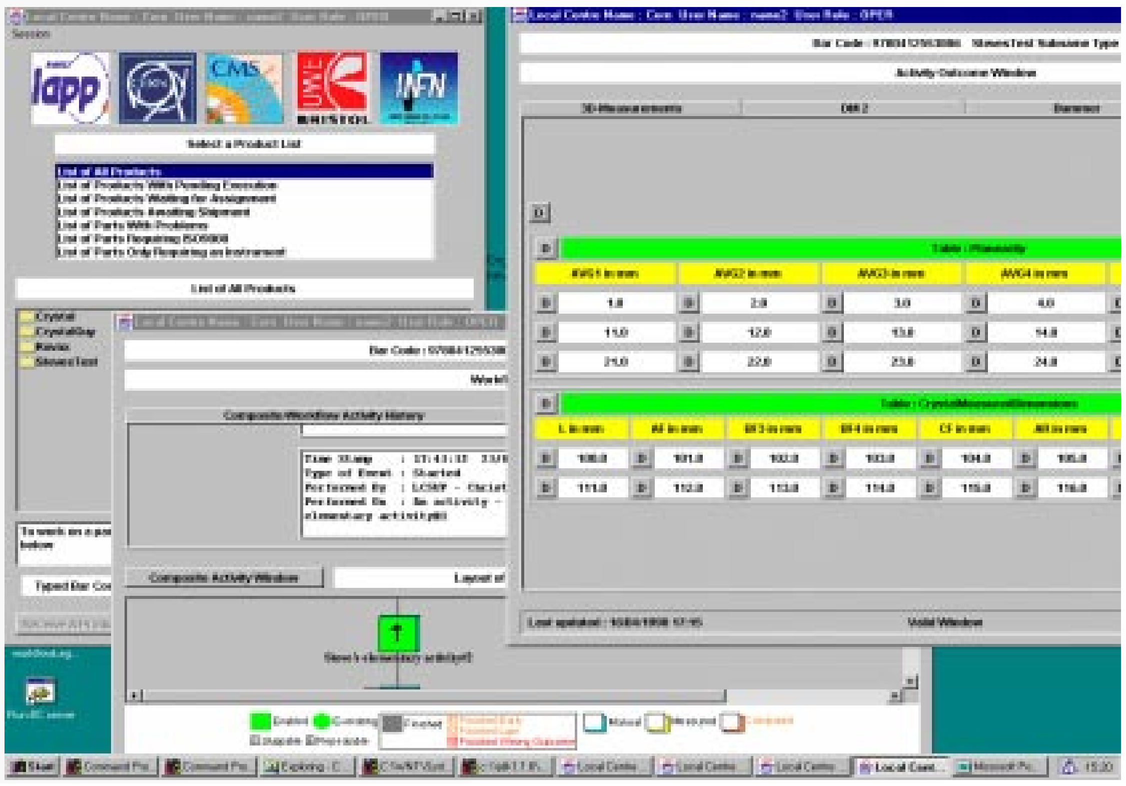


Figure 6-5. A typical screenshot of the Coordinator's Desktop Control Panel (DCP_COOR)

During this period of stabilisation of the prototype a total of 0.5 Gbyte of data was captured in the CRISTAL repository for the pre-production crystals. Around 1 Mbyte of physics data (dimensional measurements, longitudinal and transverse transmission and light yield data) was collected for each pre-production crystal. From one prototype to the next the layout and composition of products (in the PDM structure) and/or workflow activities (in the WfM) structure could evolve. In other words, in a single repository it was possible to collect data from numerous versions of products and these products could follow versions of workflows which evolved with time. This was a severe test of the flexibility of the underlying data model and, in its success, it was a notable verification of the ability of the data model to cater for changing user requirements.

Figure 6-5 and Figure 6-6 show two screenshots of the final CRISTAL prototype. In Figure 6-5 the Desktop Control Panel of the Coordinator user is displayed. In this DCP_COOR window the production details of a Barrel crystal of type 6L is being specified. The interface allows versions of product definitions to be declared in CRISTAL and for any



Select a Product List

List of All Products
List of Products With Pending Execution
List of Products Waiting for Assignment
List of Products Awaiting Shipment
List of Parts With Problems
List of Parts Requiring ISO9000
List of Parts Only Requiring an Instrument

List of All Products

Crystal
CrystalSup
Kevin
SlowesTest

Composite Workflow Activity History

```
Time Stamp      : 17:43:17  23/0
Type of Event   : Started
Performed By    : LCSUP - Christ
Performed On    : An activity -
elementary activity#0
```

Composite Activity Window

Local Centre Name : Cern User Name : name2 User Role : OPER

Bar Code : 9780412955086 SlowesTest Subassy Type

Activity Outcome Window

Table : Planarity

AVG1 in mm	AVG2 in mm	AVG3 in mm	AVG4 in mm
1.0	2.0	3.0	4.0
11.0	12.0	13.0	14.0
21.0	22.0	23.0	24.0

Table : CrystalMeasuredDimensions

L in mm	AF in mm	BF3 in mm	BF4 in mm	CF in mm	AR in mm
100.0	101.0	102.0	103.0	104.0	105.0
111.0	112.0	113.0	114.0	115.0	116.0

Last updated : 10/04/1998 17:45 Valid Window

Figure 6-6. A typical screenshot of the Operator's Desktop Control Panel (DCP_OPER)

version-specific production condition definitions to be recorded e.g StartConditions, EndConditions, ApplicableCentres etc. The DCP_COOR module is also used to specify workflow activity definitions and to supply versions of the Product Graph and Workflow Graph meta-models to Local Centres.Figure 6-6 shows the DCP of the Operator user. In this screenshot, an Operator is browsing (or inputing) a set of data which has resulted from a particular workflow activity being run on a particular product.

From an end-user standpoint the description-driven design of the prototype meant that data could be retrieved from the database and physics analyses could be carried out without the need for extra coding. Further, concurrent analyses of data could be carried out on samples that had been collected using multiple versions of the system. These features demonstrate the adaptability of the CRISTAL data model and the full power of description driven systems.

In the period between 1999 and 2005, each of the 80,000 ECAL crystals must undergo a series of rigorous tests and assembly procedures which can take several days to complete. Each Barrel crystal will be glued to a capsule and the resulting sub-unit assembly will undergo further tests. Over a Mbyte of data will be collected for each sub-unit. Sub-units are assembled into sub-modules, modules and supermodules and data is recorded at each level. In total around 1 Tbyte of data will be collected for the assembly of ECAL. Tests can be repeated on selected crystals, sub-units, modules etc. or can be skipped on a product-by-product basis. New tests and assembly procedures will need to be specified during the lifetime of construction and new data will be collected alongside data gathered following an earlier version of the assembly workflow. Any data will need to be viewed in the context in which it was recorded i.e. the product data will need to be viewed in relation to the version of the workflow process it was following at the time of data taking.

The complexity of this data taking requires the use of an automated data recording and tracking system. It would not be technically feasible for these data volumes to be captured without the use of a system like CRISTAL. In addition, any system specified for ECAL had to be durable and sufficiently flexible to accomodate frequent evolution. The meta-object structure of the CRISTAL data model has been shown to provide the flexibility, reusability, complexity handling, interoperability, version handling and schema evolution as required by the ECAL user community.

7 Conclusions

7.1 Introduction

This chapter concludes the thesis by considering whether and how its research objectives have been realised. It reflects on the use of meta-objects as a means of providing description-driven systems and on the application of the technique in integrating PDM with WfM. It then draws conclusions on the process of extracting patterns from a data model employing meta-objects. The activities of the OMG and, in particular, its Manufacturing Domain Task Force is considered in this chapter and its present work is considered in the light of the work in this thesis. Finally, potential extensions or follow-on research to the present work are considered including the use of software ‘Agents’ and the exploitation of so-called ‘viewpoints’. The role of meta-description in the construction of Virtual Enterprises for manufacturing is also briefly considered in this chapter.

7.2 Meta-Objects and Description-Driven Systems

In this doctoral research, meta-objects have been shown to be appropriate mechanisms for capturing a description of complex product and process models. A meta-object based system, named CRISTAL, has been developed for use in supporting the construction of the CMS detector, presently being assembled at CERN and at other institutes worldwide. This system has demonstrated that it is capable of describing all aspects of detector breakdown structure and production process structure for the assembly of CMS using a meta-object, or description-driven approach. Current PDM and WfM tools provide only a subset of the capability required to support product and process description. This research has demonstrated that through the use of common design artifacts it is possible to integrate PDM and WfM. The final CRISTAL prototype has shown that by generalising a descriptive approach and integrating the product description of PDMs with process description as in WfM, it is possible to deliver the functionality that is required to support the full detector assembly process from design to production. It has also shown that a repository based on meta-structures improves flexibility and reusability and allows systems to handle versioning and schema evolution.

Chapter 3 of this thesis showed how meta-objects allow systems to be designed to cater with partial specification and system evolution over time, thereby allowing design evolution during system implementation. Chapter 4 to Chapter 5 showed how this approach has been deployed in the CRISTAL project. In addition Chapter 5 investigated how existing patterns could be extended as a result of this research to provide description-driven system capabilities.

In the CRISTAL data model, there is an association between a given workflow activity meta-object definition and a named product meta-object definition. The data model has been designed so that each association of a Product Definition to a Workflow Activity Definition is declared for a specific purpose. In detector construction, the association is made to indicate the workflow activity to be instantiated for the assembly of a particular instance of a product of a given product definition. Each association of a Product Definition to a Workflow Activity Definition requires Conditions: in detector construction, the data model captures the definition of the conditions required for each assignment of a workflow activity definition to a product definition.

This technique can, however, be generalised for other applications. For example, the association of a maintenance workflow activity to a specific product will require quite different (maintenance-specific) conditions to be captured than when a construction workflow activity was associated with the product. Similarly, the association of a calibration workflow activity to a product would require calibration-specific conditions to be captured (Le Goff and McClatchey, 1999)and the association of a project management activity to a product would require project management-specific conditions to be captured. In other words, the identified association between the process and product description worlds carries rich semantics. The method of integrating PDM and WfM through the definition of meta-objects and their mutual assignment is thus very powerful. It allows many other links to be made between aspects of the overall data model: the same mechanism can be used to assign agents to workflow activity definitions for the purposes of enactment or the assignment of agents to product definitions for the purposes of resource management. This aspect is investigated in a later section of this chapter, where potential extensions to the research reported in this thesis are considered.

7.3 Designing CRISTAL

The design approach adopted for this research required close and continuous involvement with the user community and ensured that each evolutionary prototype catered for the changing, research requirements of the end-users. Each iteration in prototyping (i.e each loop in the spiral) produced domain knowledge which was fed into the design of the next iteration, for example, via UML Use Cases. The rapid prototyping approach not only assisted in eliciting user requirements but also its iterative development helped in validating the UML object model and its dynamic behaviour, leading to a very robust architecture for CRISTAL. One obvious further advantage of adopting such a rapid application development philosophy to prototype delivery was that following an evolutionary approach to systems development the inherent risk of project failure was minimised.

The approach followed for systems design was strictly dictated by the research-oriented environment in which the prototype was developed: it was driven by the requirement that the resulting application would be highly configurable, flexible, and adaptable. The design philosophy relied on the use of abstraction, disciplined use of object-oriented design and the flexible implementation of the domain-specific business rules. Rather than attempt to fit a prescribed set of patterns emerging from the research of others (Gamma et al., 1995) into the environment in which the prototype was constructed, a philosophy was followed in which an application-specific object model (that of CRISTAL) was developed using ‘traditional’ UML-modeling and that model was latterly *mined* for recurring patterns. This approach has recently been followed elsewhere in the patterns community (Yoder et al., 1998). In the research presented in this thesis it emerged that common patterns could be determined for both product and process models and the resulting patterns were either enrichments of existing patterns or new additions to the family of patterns.

This thesis work has therefore delivered not only a working prototype and a flexible and configurable data model for CRISTAL, it has also provided input to the field of patterns. Updated patterns have been proposed which facilitate the integration of product and process data models: the Graph, Homomorphism and Item Description patterns (enriched versions of that of Blaha & Premerlani, 1998), the Observer pattern (of Gamma et al., 1995) and a new Version pattern, as described in Chapter 5. It is believed that this set of patterns forms the backbone required for the construction of any description-driven system that must cater both for product and process models.

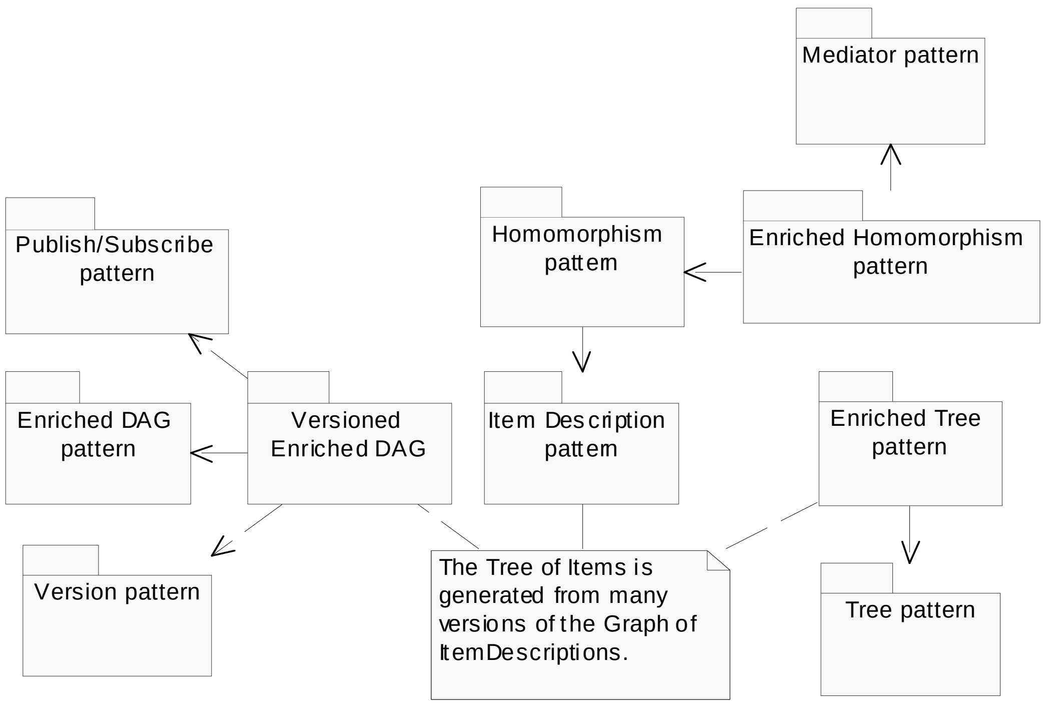


Figure 7-1. CRISTAL pattern summary

7.4 Patterns and Description Driven Systems

In conclusion to the role of patterns in CRISTAL it is informative to reflect on the patterns that have been extracted from the CRISTAL data model and to consider how they are inter-related. The following existing patterns emerged from the data model:

- ItemDescription
- Publish/Subscribe
- Homomorphism
- Graph
- Tree
- Mediator

These patterns were shown to be insufficient to provide the flexibility required in the CRISTAL data model. However, with enrichment of the Graph, Tree and Homomorphism patterns and the addition of a new Version pattern it has been possible to provide the functionality required to integrate PDM with WfM, using a description-driven approach.

Figure 7-1 summarises the inter-relationship between the set of patterns identified for the integration of PDM systems and WFM systems in the CRISTAL data model. It shows a Versioned Graph Pattern (as used in Figure 5-5 on page 68, Figure 5-6 on page 70 and Figure 5-8 on page 73) which has been derived from the Version, Complex Graph and Publish/Subscribe patterns. It also shows an Enriched Homomorphism pattern (as shown in Figure 5-13 on page 78) which has been derived from the ItemDescription and Mediator patterns. Furthermore the diagram brings together the Complex Tree pattern (of Figure 5-10 on page 75 and Figure 5-11 on page 77) with the Versioned Graph pattern from which it is derived. Instantiation of the Complex Tree pattern from the Versioned Graph pattern is performed by the Product Manager which consequently carries out the integration of the PDM and WfM aspects and acts as the CRISTAL execution component.

Blaha and Premerlani (1998) state that “patterns provide a higher level of building blocks for models than the base primitives of class, association and generalization”. This work has shown that this assertion is not only true for models but can be extended to include meta-models.

7.5 PDM and WfM Integration and the OMG

The integration of PDM and WfM has been investigated in the context of a production-specific process in the present work. It has been demonstrated that full integration can be accomplished through the use of a common meta-modeling approach to both PDM- and WfM-specific objects. The familiar BOM (or Bill Of Materials), used in PDM, is not explicitly visible in the model, but is deduced from the execution of a collection of processes (as defined in the meta-layer) where the PDM-WfM integration actually takes place. The CRISTAL model has been designed to be sufficiently abstract to cater for a three-layer architecture of description (see Chapter 3). In doing so, the present system could be applied to other product and process models. For example, the data model could be used to provide document management across the product life cycle - in this case the assignment of a process definition to a product definition would allow the capture of design conditions, as compared to the production conditions captured in the CMS ECAL application.

There are several OMG facilities which are relevant to this integration of PDM with WfM. These include the Workflow Facility (of the Business Object Domain Task Force, see OMG 1998a), the PDM Enablers (of the Manufacturing Domain Task Force, see OMG 1998b) and the Meta-Object Facility (MOF see OMG 1997). Each of these facilities has been proposed since the outset of the research in this thesis and it is interesting to compare the direction followed by the OMG with the philosophy expounded in this thesis.

The purpose of the OMG MOF is to provide a set of CORBA interfaces that can be used to define and manipulate a set of interoperable meta models. The intention is that the meta-meta objects defined in the MOF will provide a general modeling language capable of specifying a diverse range of meta models (Schulze et al. (1998)). The MOF is a key component in the CORBA Architecture as well as the Common Facilities Architecture. Because of the description-driven approach (and the use of patterns) adopted in CRISTAL, any PDM Enabler and/or Workflow facility can be mapped onto the CRISTAL model and this maps exactly onto the Meta-Object Facility of the OMG.

Typically, the MOF will be used for manipulating meta objects to provide integration of tools and applications across the life cycle using industry standard meta-models, such as UML, which itself is being proposed as an OMG standard. A technical goal of importance to the MOF is interoperability with OMG CORBA and the integration with the OMG Business Object Facility and the OMG Object Analysis and Design (OA&D) domain. All these three facilities have strong meta data heritage.

As was stated at the end of Section 7.4, patterns can be used as high level building blocks in the construction of meta-models. This implies that patterns should be included in the MOF, perhaps as a library of reusable modeling artifacts. Designers could then use the patterns as well-established building blocks to facilitate construction of domain-specific meta-models.

Currently the OMG Manufacturing Domain Task Force has established a Product and Process engineering (PPE) working group to ‘develop standardized interfaces for software systems supporting design and analysis of products and design and analysis of the processes and facilities used to make them’. The scope of this effort encompasses engineering throughout the product and process life-cycle including the following kinds of manufacturing systems: CAD/CAM/CAE, PDM, Product Configuration Management, Process modeling and simulation, Product/Process classification, process engineering, plant design and engineering and Product Simulation. This working group has recently stated that the OMG workflow specification and the PDM Enablers are complementary and that the PDM Enablers provide the repository for (multiple versions of) information and the Workflow Facility provides the process to dynamically change the information. This is exactly in line with the conclusions of this thesis which, in addition, provides a detailed data model for this integration.

7.6 Potential Extensions to this Research

7.6.1 Agents and Workflows

Software agents, or cooperating independent and potentially intelligent processes, have been suggested as mechanisms for planning, controlling and optimising production management systems. Agent technologies have been successfully deployed in the design and modification of business processes (Hall & Shahmehri, 1996) and have been proposed as an approach to the management of workflow in business processes (Merz et al., 1996). The Carnot project of Singh et al. (1997) uses Cyc (Douglas B. Lenat (1995)), an ontological approach of sharing information, from which autonomous processes can extract data for decision-making. Up to now, however, agent-based systems have not been used extensively in industrial production management. Agent software could be used both to enact the workflow activities and to interact with the PDM definitions.

Chapter 5 described the data model that has been developed to integrate PDM and WfM aspects for the CRISTAL application. It outlined a set of ‘packages’ which partitioned the data model into aspects such as PBS, WBS, Detector Design & Construction Management and Production Specification. Further packages have been identified in this model including the Agent World package and the Execution Specification package. In the AgentWorld package, object classes such as HumanDefinition, UserCodeDefinition and InstrumentDefinition reside and these ‘Agents’ are defined as the executors or enactors of specific CRISTAL workflows. This package interacts largely with the WBS and Execution Specification package. Agents are Humans (Operators or LocalSupervisors), UserCode supplied by end-users of the CRISTAL system to be invoked automatically by CRISTAL, or Instruments with which the system communicates and which can carry out specific workflow activities. As a result of the execution of these instructions, execution results are gathered of a particular data format definition whose detail is captured in the (separate) DataFormat package.

The ExecutionSpecification package has dependencies on both the WBS and the AgentWorld packages. This is the mechanism through which a defined Agent definition is *enacted* for a specific workflow definition. The assignment of Agents to workflow activities is subject to a set of Agent Conditions, in a manner entirely analogous to that in which Product Definitions are assigned to Workflow Definitions through Production Conditions. The AgentConditions can therefore define the *role* of the agent with respect to a Workflow Definition. Agent Conditions are made up from Outcome and Goal Definitions which together define the operation of the workflow activity by the agent. Each agent-workflow assignment is established to reach a desired goal and following the execution of its activities, the workflow result is stored as an outcome definition. One obvious area where the use of Agent technology can be applied is in resource allocation and management in a workflow system. Currently work is in progress to use software agents to optimise the allocation of resources (e.g. Operators, Machines) to workflow activities in CRISTAL (Murray et al., 1999).

7.6.2 The CRISTAL Repository as a Data Warehouse

Seen from a standpoint of the centralised capture of corporate data from multiple sources and the storage of that data for extraction by multiple users over extended time scales, the CRISTAL database can be viewed as a 'data warehouse' (IEEE, 1995 and Widom, 1995). For the purposes of optimisation and analysis, physicists and engineers must be able to extract data from the general CMS construction data warehouse from a number of viewpoints. The physicist defines his viewpoint in terms of familiar sets of detector components (or 'physics elements') which naturally derive from the tree of physical locations of detector components. This viewpoint definition is captured in the data model for future use. Figure 7-2 shows how a matrix of physics elements can be extracted from the 'as-built' detector construction hierarchy by providing software which traverses the construction tree and extracts the construction data for a selected set of physics elements.

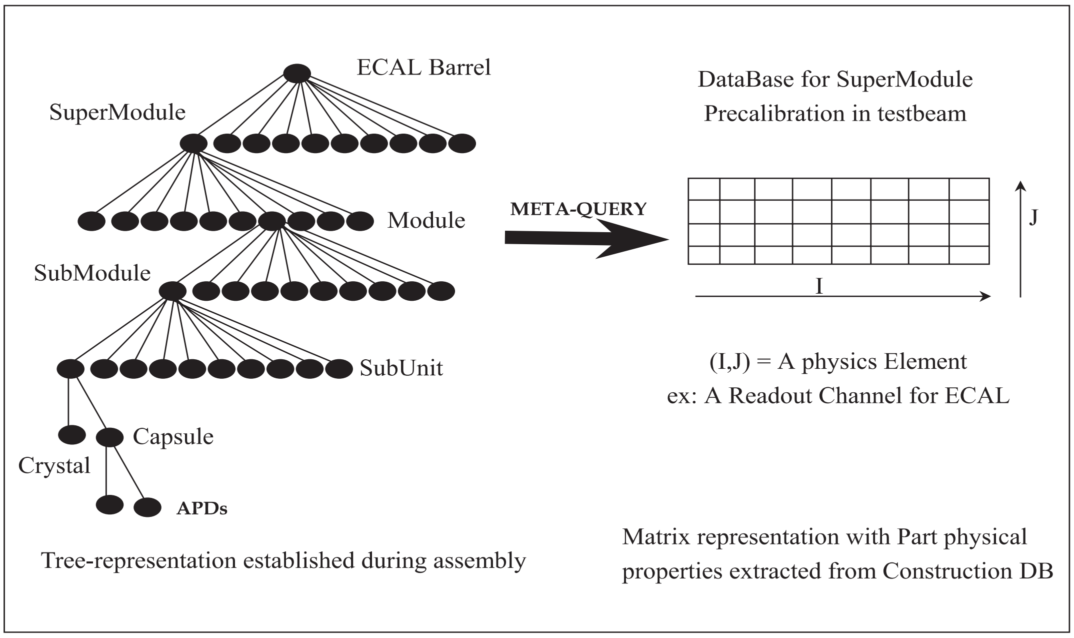

ECAL Barrel
SuperModule
Module
SubModule
SubUnit
Crystal
Capsule
APDs
Tree-representation established during assembly
META-QUERY
DataBase for SuperModule Precalibration in testbeam
J
I
(I,J) = A physics Element
ex: A Readout Channel for ECAL
Matrix representation with Part physical properties extracted from Construction DB

Figure 7-2. The extraction of data from CRISTAL data warehouse into 'viewpoints'

Meta-modeling can assist the extraction of data for physics elements provided such an extraction or *meta-query* mechanism is developed which navigates the meta-model, interprets the structures in the warehouse and presents the data in a form meaningful to the end-user. The meta-query facility comprises a set of software processes (or agents) which are invoked either by a viewpoint-specific application (e.g. calibration) or by a viewpoint non-specific application. The agents either navigate the generalised warehouse meta-model to project out viewpoint-specific data (i.e. ‘looking in’ the meta-model) or they mine the meta-model to correlate effects between viewpoints (i.e. ‘looking out’ from the data model). In the ‘looking in’ (viewpoint-specific) case the agents perform the traversal of the detector description, following selected physics elements in the construction tree and extract the relevant physics data for the application. In the ‘looking out’ case (viewpoint non-specific), the agents are used to determine the effect of a system-wide change on individual viewpoints or sets of viewpoints i.e across viewpoints. A meta-query facility is currently under development for CRISTAL (Estrella et al., 1998).

Most of the time production data is viewed in a manner different to that in which it was collected. In the example of CRISTAL, the CMS detector construction process necessarily leads to a particular construction-oriented representation of construction data. However, for the purposes of optimisation and analysis, physicists and engineers must be able to extract data from the general CMS construction data warehouse from numerous viewpoints. It must be inherently simple for users to extract sets of data from the terabyte-sized warehouse, according to defined criteria, for subsequent analysis. As the CMS production process evolves more data, together with the relationships between different aspects of the data, must be permanently recorded in the CRISTAL data warehouse. Different groups of users will require flexible ways to locate, access and share this production data. The actual information required will depend on the viewpoint and the role of the user in the organisation. User groups may well require a maintenance, a geometry, an alignment or an experiment slow-controls viewpoint. The physicist therefore needs to define a viewpoint in terms of 'physics elements' (or sets of detector components) which are derived from the tree of physical locations of detector components resulting from detector assembly.

The CRISTAL architecture must support autonomous data collection in remote Centres with secure storage in the central data warehouse at CERN. New versions of product and workflow definitions are defined centrally at CERN and are farmed out to each Centre by the warehouse Coordinator. Facilities are therefore required to share configuration data between the independent data acquisition Centres. Remote Centres must also be allowed to continue data gathering even when the network connection to the warehouse is down. Data is duplicated from the Centres to the warehouse when network connections allow. Once written into the warehouse, the data is thereafter mined by read-only processes running centrally.

The database architecture selected for CRISTAL is based on the Objectivity federated database concept in which multiple autonomous databases co-operate. The Central warehouse holds the federation configuration files and any changes in the schema take place centrally and are dispatched to the Centres via a small, shared configuration database (the Configuration DB). Production data is gathered at the Centres and copied remotely to the warehouse. Data mining takes place only at the data warehouse. One example of mining the construction data warehouse is for the extraction of detector calibration data. For physicists calibration data is required for each electronic readout channel of the detector. The structure of readout channels, however, is necessarily different to the assembly structure of the detector. In essence, the calibration system must be able to mine subsets of physics data from the construction database for the calibration of particular components even if these components are specified in a manner which is different to that in the construction database. This is facilitated in CRISTAL through the use of the self-describing nature of the warehouse meta-model.

7.6.3 CRISTAL and the Virtual Enterprise

With increasing complexity of the organisation, data and functions of enterprises, information systems need to be increasingly flexible and extendible, intuitive to interrogate and navigate around and to be interoperable with existing legacy systems. Large scale engineering and scientific projects may, in addition, demand systems which require integration and/or distribution over many separate organisations. The integration of such ‘islands of information’, which ultimately forms the basis of so-called ‘virtual enterprises’ (see van Parunak, 1997) is heavily dependent on the flexibility and accessibility of the data model describing the enterprise’s repository. The model must provide interoperability, extensibility and reusability so that a range of applications can access the enterprise data. Making the repository self-describing and based on meta-object structures ensures that knowledge about the repository structure is available for applications to interrogate and to navigate around for the extraction of application-specific data. In this paper, a large application is described which uses a meta-object based repository to capture product and workflow data in an engineering data warehouse. It shows that adopting a meta-modeling approach to repository design provides support for interoperability and a suitable environment on which to build enterprise-wide data navigation.

The concept of using meta-data to reduce complexity and aid navigability of data resident in a database is well known. Also its use in minimising the effect of schema evolution in object databases has been stated many times elsewhere. In the CRISTAL project meta-data are used for these purposes and, in addition, meta-models are used to provide self-description for data and to provide the mechanisms necessary for developing a meta-query facility to navigate multiple data models. Using meta-queries, data can be extracted from multiple databases and presented in user-defined viewpoints. The CMS meta-model therefore acts as a repository of knowledge against which meta-queries are issued to locate and extract data across multiple databases. Agent processes are used to ‘look in’ the meta-model and extract data from a user-specified viewpoint and to ‘look out’ from the model to correlate effects between viewpoints. The overall effect is to produce an integrated set of cooperating databases accessed through a meta-query facility. Hence ‘islands’ of disparate information (such as maintenance, calibration, alignment) are eliminated. Such an approach could reasonably be applied to organisations developing technologies for ‘virtual enterprises’ (such as in Hardwick et al. (1996) and Gaines et al. (1995)) where collections of autonomous databases could be related via a central enterprise meta-model.

7.7 Closing Statement

Proliferation of systems development on standard meta data services, such as the OMG MOF, together with industry standard meta-models, such as UML, accelerates the market for component software in general and description-driven (or model-driven) component software development in particular, because components meeting specific semantics and requirements can be discovered using the standard meta-object interfaces. Additional work in the areas of standard meta-models for database technologies, component management and tracking, transaction discovery and legacy integration is expected in the future. This thesis work is presented as one step towards the production of standard description-driven component software.

8 Glossary

To aid the readability of this thesis the author includes here a glossary of all the acronyms deployed throughout the body of this thesis's text.

ABS: Assembly Breakdown Structure.

ACCOS: Automatic Crystal Control System.

ADC: Analogue-to-Digital Converter.

ANSI: American National Standards Institute.

APD: Avalanche Photo-Diode.

API: Application Programming Interface.

BOM: Bill Of Materials.

CAD/CAM: Computer Aided Design / Computer Aided Manufacturing.

CAE: Computer Aided Engineering.

CDef: CRISTAL Definitions.

CERN: Conseil Europeen pur la Research Nucleaire. (Now the European Centre for Particle Physics).

CMS: The Compact Muon Solenoid (experiment).

CORBA: Common Object Request Broker Architecture.

CPU: Central Processing Unit.

CRISTAL: Cooperating Repositories and an Information System for Tracking Assembly Lifecycles.

CSCW: Computer Supported Cooperative Work.

DAcq: Data Acquisition.

DBMS: DataBase Management System.

DCP: Desktop Control Panel.

DDL: Data Definition Language.

DDM: Data Duplication Manager.

DESY: Deutsches Elektronen-Synchrotron.

DPS: Detector Production Scheme.

ECAL: Electromagnetic Calorimeter.

EDMS: Engineering Document Management System.

GUI: Graphical User Interfaces.

HCAL: Hadronic Calorimeter.

HEP: High Energy Physics.

HERA: Hadron-Electron Ring Accelerator facility.

IRDS: Information Resource Dictionary System.

IDL: Interface Definition Language.

ISO: International Standards Institute.

ISR: Intersecting Storage Rings.

JDK: Java Development Kit.

LAN: Local Area Network.

LAPP: Laboratoire d’Annecy for Physique des Particules.

LCM: Local Centre Manager.

LEP: Large Electron Positron collider (Phase I).

LEPII: Large Electron Positron collider (Phase II).

LHC: Large Hadron Collider.

MOF: Meta-Object Facility.

MSGC: Micro-Strip Gas Chambers.

OA&D: Object Analysis and Design.

ODBMS: Object DataBase Management System.

ODL: Object Definition Language.

ODMG: Object Database Management Group.

OIF: Object Interchange Format.

OMA: Object Management Architecture.

OMG: The Object Management Group.

OML: Object Manipulation Language.

OMT: Object Modeling Technique.

OQL: Object Query Language.

PBS: Product Breakdown Structure.

PDM: Product Data Management system.

PM: Product Manager.

PPE: Product and Process Enginering

RDBMS: Relational DataBase Management System.

SPS: Super Proton Synchrotron.

SQL: Structured Query Language.

UML: Unified Modeling Language.

WBS: Work Breakdown Structure.

WfMC: The Workflow Management Coalition.

WfM: Workflow Management systems.

References

[1] Alexander, C., S Ishikawa and M. Silverstein (1977) with M. Jacobson, I. Fiksdahl-King and S. Angel *A Pattern Language*. Oxford Press, New York.

[2] Alonso, G., D. Agrawal, A. El Abbadi, M. Kamath, R. Guenthor and C. Mohan (1996) *Advanced Transaction Models in Workflow Contexts.* In Proc. of the 12th IEEE International Conference on Data Engineering, New Orleans, USA.

[3] ANSI (1988) *American National Standard X3.138-1988: Information Resource Dictionary System,* American National Standardization Institute, New York.

[4] Auffray, E. et al., (1998) *Certifying Procedure for Lead Tungstate Crystal Parameters during the Mass production for CMS ECAL*. IEEE Transactions on Nuclear Science, in press.

[5] Bachy, G and A. Hameri (1995) *What Should be Implemented at the Early Stages of a Large-Scale Project*. CERN MT/95-02 (DI) LHC Note 315.

[6] Baker, N. and J-M. Le Goff (1997) *Meta-Object Facilities and their Role in Distributed Information Management Systems.* In Proc. of the EPS ICALEPCS'97 International Conference Beijing, China. Published by Science Press.

[7] Baker, N. et al. (1998) *An Object Model for Product and Workflow Data Management*. In Proc. of the 9th ACM International Workshop and Conference on Database and Expert Systems Applications (DEXA'98) pp 731-738.

[8] Baumer, D. et al. (1997) *Framework Development for Large Systems*. Communications of the ACM Vol 40 (10) pp 52-59.

[9] Bazan A. et al. (1998) *The Use of Production Management Techniques in the Construction of Large-Scale Physics Detectors*. IEEE Transactions on Nuclear Science, in press.

[10] Blaha, M. and W. Premerlani. (1998) *Object-Oriented Modelling and Design for Database Applications*. Prentice Hall publishers.

[11] Boehm, B. (1988) *A Spiral Model of Software Development and Enhancement.* IEEE Computer Vol. 21 (5) pp 61-72.

[12] Bussler, C. (1997) *Metadata in Workflow Management Systems*. In Proc. of the 2nd IEEE MetaData Conference

[13] Byrne, B. (1996) *IRDS Systems and Support for Present and Future CASE Technology*. In Proc. of the CAiSE'96 International Conference, Heraklion, Crete.

[14] Cadim (1998) CADIM: A Product Data Managent system produced by Eigner & Partner. See http://www.ep-ag.com.

[15] Cattell, R. (ed.) (1994) *The Object Database Standard: ODMG-93*. Morgan Kaufmann publishers.

[16] CIMdata (1998). *Product Data Management: The Definition. An Introduction to Concepts, Benefits and Terminology.* CIMdata white paper, available from http://www.CIMdata.com

[17] CMS (1995). *The CMS Technical Proposal.* Available from ftp://cmsdoc.cern.ch/TPref/TP.html. Also CERN/LHCC 94-38.

[18] Coad, P. (1992) *Object-Oriented Patterns*. Communications of the ACM Vol 35 (9) pp 152-159.

[19] Coad, P., D. North and M. Mayfield (1995) *Object Models: Strategies, Patterns and Applications.* Englewood Cliffs, New Jersey: Yourdon Press, 1995.

[20] Crawley, S. et al. (1997a) *Meta-Information Management*. In Proc. of the International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS), Canterbury, UK.

[21] Crawley, S. et al. (1997b) *Meta-Meta is Better-Better !* In Proc. of the IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS'97).

[22] Davis, A.M., E.H. Bersoff, and E.R. Comer, (1988) *A Strategy for Comparing Alternative Software Development Life Cycle Models.* IEEE Transactions on Software Engineering, Vol. 14, No.10, Opp. 1453-1461.

[23] Devos, M. and M. Tilman. (1998) *Some Reflections on Building a Reflective Framework*. In Proc. of the MetaData Pattern Mining Workshop at the ACM OOPSLA'98 International Conference.

[24] Douglas B. Lenat (1995) *CYC: A Large-Scale Investment in Knowledge Infrastructure.* Communication of the ACM, Vol 38(11) pp. 32-38

[25] ECAL (1997). The CMS Electromagnetic Calorimeter Project - Technical Design Report. Available from http://cmsdoc.cern.ch/ftp/TDR/ECAL/ecal.html. Also CERN/LHCC 97-33.

[26] Ellis, C. et al. (1991) *Groupware Some Issues and Experiences*. Communications of the ACM, Vol 43(1) pp. 39-58.

[27] Ellis, C. and G. Rozenberg. (1995) *Dynamic Change Within Workflow Systems*. In Proc. of the International Conference on Organizational Computing Systems (COCS) pp 10-22.

[28] Estrella, F., Z. Kovacs, J-M Le Goff, R. McClatchey and I. Willers (1998). *The Design of an Engineering Data Warehouse Based on Meta-Object Structures.* To be published in Lecture Notes in Computer Science, Springer-Verlag publishers.

[29] Feiler, P. (1991) *Configuration Management Models in Commercial Environments,* Technical Report, CMU/SEI-91-TR-7, ESD-91-TR-7. Available from http://www.sei.cmu/~case/scm/abstract/abscm_models_TR07_91.html.

[30] Foote, B. and J. Yoder (1998) *MetaData and Active Object-Models*. In Proc. of the International Conference on Pattern Languages of Programs, Monticello, USA.

[31] Fowler M. and K. Scott (1997) *UML Distilled - Applying the Standard Object Modelling Language*. Addison-Wesley Longman publishers.

[32] Gamma, E., R. Helm, R. Johnson and J. Vlissides (1995) *Design Patterns - Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman publishers.

[33] Gaines, B., D. Norrie and A. Lapsley (1995) *An Intelligent Information System Supporting the Virtual Manufacturing Enterprise*. In Proc. of the IEEE International Conference on Systems, Man & Cybernetics.

[34] Georgakopoulos, D., M. Hornick and A. Sheth (1995) *An Overview of Workflow Management: from Process Modelling to Infrastructure for Automation,* Journal of Distributed and Parallel Database Systems Vol 3 (2), pp 119-153.

[35] Gordon, V.S., and J.M. Bieman (1991) *Rapid Prototyping and Software Quality: Lessons From Industry.* Technical Report No. CS-91113, Department of Computer Science, Colorado State University.

[36] Hales, K. and M. Lavery (1991) *Workflow Management Software: the Business Opportunity.* Ovum Ltd., London, UK.

[37] Hall, T. and N. Shahmehri (1996) *An Intelligent Multi-Agent Architecture for Support of Process Reuse in a Workflow Management System.* In Proc. of the 1st International Conference on the Practical Application of Intelligent Agents and Multi-Agent technology, pp 331-343.

[38] Hameri, A. (1996) *EDMS - Concepts, Motivations and Basic Requirements*. Proc. of the CERN School of Computing 1996, CERN 96-08.

[39] Hameri, A. and J. Nihtila (1998) - *Product Data Management - Exploratory Study on State-Of-The-Art in One-Of-A-Kind Industry.* Journal of Computers In Industry Vol. 35. (3) pp 196-206.

[40] Hardwick, M., et al. (1996) *Sharing Manufacturing Information in Virtual Enterprises*. Communications of the ACM Vol 39 (2) pp 46-54.

[41] Haugen, R. (1998) *Inventory by Process Stage: A Useful Pattern for Integrating Process and Product*. In Proc. of the OOPSLA'98 Mid-Year Workshop on Implementing Lifecycle Process and Product Models.

[42] HCAL (1997). The CMS Hadronic Calorimeter Project - Technical Design Report. Available from http://cmsdoc.cern.ch/ftp/TDR/HCAL/hcal.html. Also CERN/LHCC 97-31.

[43] Hsu, M. (ed.) (1995) *Special issue on Workflow Systems*. Bulletin of the IEEE Technical Committee on Data Engineering, Washington, USA.

[44] IEEE (1995). *IEEE Data Engineering Bulletin*, Special Issue on Materialised Views and Data Warehousing, 18 (2), June 1995.

[45] IEEE (1996, 1997) Selected papers from the 1st and 2nd IEEE MetaData Conference, Silver Spring, Maryland, 1996 and 1997

[46] ISO (1998) The International Standards Organisation. See http://www.iso.org.

[47] Jablonski, S. and C. Bussler (1996) *Workflow Management: Modelling Concepts, Architecture and Implementation*. Thomson Computer Press.

[48] Jacobson, I. (1994) Object-Oriented Software Engineering - A Use Case Driven Approach. Addison-Wesley publishers Ltd.

[49] Java (1998) see http://www.javasoft.com

[50] Johnson, R., and B. Foote (1988) *Designing Reusable Classes*. Journal of Object-Oriented Programming Vol 1 (3) pp 22-35.

[51] Johnson, R., and J. Oakes (1999) The User-Defined Product Framework. Work in progress.

[52] Kerherve, B. and A. Gerbe. (1997) *Models for MetaData of MetaModels for Data?* In Proc. of the 2nd IEEE MetaData Conference.

[53] Kim, W. (1995) *Modern Database Systems*. ACM Press / Addison-Wesley Longman publishers.

[54] Kovacs, Z., J-M. Le Goff and R. McClatchey (1998) *Support for Product Data from Design to Production*. Computer Integrated Manufacturing Systems Vol 11 (4) pp 285-290.

[55] Laddaga, R. and J. Veitch (1997) Guest Editors: *Dynamic Object Technology*. Communications of the ACM Vol. 40 (5) pp 37-69.

[56] Le Goff, J-M. and R. McClatchey (1999) *The CMS ECAL Detector Database System.* Accepted for presentation at AIHENP'99, the 6th International Workshop on Artifical Intelligence in High Energy and Nuclear Physics, Crete, Greece. April 1999.

[57] Lebeau, M., P. Lecoq and J-P. Vialle (1995) *A Distributed Control and Data base System for the Production of High Quality Crystals*. CERN report CMS TN/95-024.

[58] Lee, C., R. Sause and K. Hong (1998) *Overview of Entity-Based Integrated Design Product and Process Models*. In Advances in Engineering Software, Vol. 29 (10) pp 809-823.

[59] LHC (1993) *The Large Hadron Collider accelerator project*. CERN AC/93-03.

[60] Lowell, J. (1992) *Rapid Evolutionary Development : Requirements, Prototyping and Software Creation*. John Wiley publishers.

[61] Maes, P. (1987) *Concepts and Experiments in Computational Reflection*. In Proc. of the ACM OOPSLA'87 International Conference pp 147-155.

[62] Maes, P. and D. Nardi (eds.) (1988) *Meta-Architectures and Reflection*. North-Holland Publishers.

[63] R. Malveau & T. J. Mowbray (1997) *Corba Design Patterns*. Published by John Wiley & Sons, 1997 ISBN 0-471-15882-8

[64] Manolescu, D-A. and R. Johnson (1998). *A Proposal for a Common Infrastructure for Process and Product Models*. In Proc. of the OOPSLA'98 Mid-year Workshop on Applied Object Technology for Implementing Lifecycle Process and Product Models, July 1998, Denver, Colorado.

[65] Matrix (1998). MATRIX: A Product Data Management system from ADRA. See http://www.adra.com

[66] McClatchey, R. et al. (1997) *A Distributed Workflow and Product Data Management Application for the Construction of Large Scale Scientific Apparatus*. NATO ASI Series F: Computer & Systems Sciences Vol 164 pp 18-34.

[67] Merz, M., D. Moldt, K. Muller and W. Lamersdorf (1996) *Inter-Organisational Workflow management with Mobile Agents in COSM*. In Proc. of the 1st International Con-

ference on the Practical Application of Intelligent Agents and Multi-Agent technology, pp 405-420.

[68] Mohan, C., G. Alonso, R.Guenthor and M. Kamath (1995). *Exotica: a Research Perspective of Workflow Management Systems.* Data Engineering Bulletin Vol. 18 No. 1, 1995.

[69] Murray, S. et al. (1999) *A Software Tool for Optimising the Production of a Large Crystal Calorimeter.* Accepted for presentation at AIHENP'99, the 6th International Workshop on Artifical Intelligence in High Energy and Nuclear Physics, Crete, Greece. April 1999.

[70] Objectivity (1998) *Objectivity: an Object Oriented database* product from Objectivity Inc. See http://www.objy.com.

[71] OMG (1992a) *The Common Object Request Broker: Architecture & Specifications*, OMG publications.

[72] OMG (1992b) *The Object Management Architecture guide V2.1*, OMG publications

[73] OMG (1993) *CORBAservices: Common Object Facilities Specification.* OMG publications.

[74] OMG (1994) *CORBAservices: Common Object Services Specification.* OMG publications.

[75] OMG (1997) *Meta-Object Facility RFP-5,* TC document cf/96-05-02, Evaluation Report TC document cf/97-04-02 and ad/97-08-14.

[76] OMG (1998a) *Workflow Management Facility, Second Submission*, Business Object Management document bom/98-0607. OMG publications.

[77] OMG (1998b) *PDM Enablers, Proposal*, Manufacturing Domain Task Force document mfg/98-02-02. OMG publications.

[78] Oracle (1998). Oracle a relational database management system. See http://www.oracle./com

[79] Orbix (1998). A commercial implementation of the OMG CORBA standard. See http://www.iona.com.

[80] Papazoglou, M. and G. Schlageter (1998). *Cooperative Information Systems: Trends and Directions*. Academic Press Limited, London, UK.

[81] Parunak, H. Van Dyke (1997) *Technologies for Virtual Enterprises*. Forthcoming in the Agility journal, available from http://www.erim.org/~van/papers.htm#VEandEC

[82] PDM (1998). The Product Data Management Information Centre. See http://www.pdmic.com.

[83] Peigneux, J-P. et al., (1997) *Studies and Proposals for an Automatic Crystal Control System.* CERN CMS NOTE 1997/036.

[84] Peters, J. and T. Ozsu (1994) *Reflection in a Uniform Behavioural Object Model*. Lecture Notes in Computer Science Vol 823, Springer-Verlag publishers.

[85] Philpotts, M. (1996) *An Introduction to the Concepts, Benefits and terminology of Product Data Management*, Industrial Management & Data Systems, Vol 4 pp 11-17

[86] Pikosz, P. and J. Malmqvist (1996) *Possibilities and Limitations when Using PDM Systems to Support the Product Development Process*, In Proc. of NordDesign'96, Helsinki, Finland.

[87] Pikosz, P. and J. Malmqvist (1997) *Strategies for Introducing PDM Systems in Engineering Companies.* Proc of the 4th ISPE International Conference on Concurrent Engineering.

[88] Pressman, R. (1992) *Software Engineering: a Practitioner's Approach*. McGraw-Hill publishers.

[89] Ramanathan, J. (1996) *Process Improvement and Data Management*, IIE Solutions, 1996 Vol 28 (12) pp 24-27.

[90] Riehle, D. and T. Gross (1998) *Role Model Based Framework Design and Implementation*. In Proc. of the ACM OOPSLA'98 International Conference, pp 117-133.

[91] Roberts, D. and R. Johnson (1998) *Evolve Frameworks into Domain-Specific Languages: Pattern Languages of Program Design 3*. Addison-Wesley Longman publishers.

[92] Rumbaugh, J. et al. (1991) *Object Oriented Modelling and Design*. Prentice Hall publishers.

[93] Schall, T. (1996) *Workflow Management Systems for Process Organisations*, Lecture Notes in Computer Science Vol 1096 Springer-Verlag publishers.

[94] Schulze, W., K. Meyer-Wegener and M. Bohm (1996) *Services of Workflow Objects and Workflow Meta-Objects in OMG-Compliant Environments*. In Proc. of the ACM OOPSLA'96 Workshop on Business Object Design and Implementation, San Jose, USA.

[95] Schulze, W. (1997) *Fitting the Workflow Management Facility into the Object Management Architecture*. In Proc. of the Third Workshop on Business Object Design and Implementation, OOPSLA'97.

[96] Schulze, W., C. Bussler and K. Meyer-Wegener. (1998) *Standardising on Workflow Management - The OMG Workflow Management Facility.* ACM SIGGROUP Bulletin Vol 19 (3).

[97] Sheth, A., D. Georgakopoulos, S. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden and A. Wolf (1996). *Report from the NSF workshop on workflow and process automation in information systems*. SIGMOD Record vol. 25 No. 4, pp 55-67.

[98] Sheth, A. and K. Kochut (1997). Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems. NATO ASI Series F: Computer & Systems Sciences Vol 164 pp 35-60.

[99] Singh, M., P. Cannata, M. Huhns, N. Jacobs, T. Ksiezyk, K. Ong, A. Sheth, C. Tomlinson & D. Woelk (1997). *The Carnot Heterogeneous Database Project: Implemented Applications*. Distributed and Parallel Databases Vol. 5 No. 2 pp 207-225.

[100] Stumpf, A., R. Ganeshan, S. Chin and L. Liu (1996) *Object-Oriented Model for Integrating Construction Product and Process Information*. Journal of Computing in Civil Engineering Vol, 10 (3) pp 204-212.

[101] TRACKER (1998). The CMS Tracker Project - Technical Design Report. Available from http://cmsdoc.cern.ch/ftp/TDR/TRACKER/tracker.html. Also CERN/LHCC 98-6.

[102] Wainer, J. et al. (1996) *Scientific Workflow Systems,* In Proc. of the NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions, Athens, Georgia, USA.

[103] Weske, M., G. Vossen and C. Medeiros (1996). *Scientific Workflow Management: WASA Architecture and Applications*. Fachbericht Angewandte Mathematik und Informatik 03/96-I, University of Muenster. http://wwwmath.uni-muenster.de/dbis/Weske/Common/wasa.html

[104] WfMC (1996) *Workflow Management Coalition: Glossary & Terminology*. Document No WFMC-TC-1011. See http://www.aiai.ed.ac.uk:80/WfMC

[105] Widom, J. (1995) *Research Problems in Data Warehousing*. Proceedings of the Fourth International Conference on Information and Knowledge Management (CIKM '95), pages 25-30, Baltimore, Maryland, November 1995. Invited paper.

[106] Wodtke, D., J. Weissenfels, G. Weikum and A. Kotz-Dittrich (1996). *The Mentor Project: Steps Towards Enterprise-wide Workflow Management*. In Proc. of the 12th IEEE International Conference on Data Engineering, New Orleans, USA.

[107] Yoder, J., B. Foote, D. Riehle and M. Tilman (1998) *Metadata and Active Data Models*. In Proc. of the MetaData Pattern Mining Workshop at the ACM OOPSLA'98 International Conference.

Appendix A - CRISTAL Data Dictionary

This appendix gives more detailed description of the classes introduced in Chapter 5 in order to aid understanding of that chapter. It contains simplified description of classes with associated attributes, which are relevant to the study of this thesis.

A.1 CRISTAL Meta-Model Package

This package contains classes to manage and store the specification of the detector production. The behaviour of each of the classes in this package are, in principle, very similar as their sole purpose is to manage meta-data. They differ mostly in their attributes (i.e. the type of data they manage).

A.1.1 Production Scheme Management Package

This package contains classes for the overall management of definitions.

A.1.1.1 DetectorProductionScheme (DPS)

This manages the list of consistent/applicable releases of the detector production specification (i.e. it implements the ReleaseManager of the Version pattern). It is the ‘root’ to find any CristalDefinition object, and has public methods to manipulate them as a whole (e.g. get the list of all FieldDefinition in a given DPS release).

Attributes:

- mIdentifier : Integer
- mName : String
- mListOfProperties : List<DetectorProdSchemeProperty>

A.1.1.2 DetectorProdSchemeProperty

This maintains the set of CristalDefinitions which are included in a single DPS release (i.e. it implements the ReleaseManagerProperty of the Version pattern).

Attributes:

- mVersionId : Integer
- mListOfDefnitions : List<CristalDefinition>

A.1.1.3 CristalDefinition (CrDef)

This is an abstract class for all definition classes (i.e. it implements the VersionedObject of the Version pattern and the ItemDescription of the Item Description pattern). It defines the unversioned-data which are common to all of the definitions and has public methods to manipulate it. Also it can retrieve the property that is valid for a given DPS release (see mMapOfProperties attribute).

Attributes:

- mIdentifier : Integer
- mName : String
- mSubName : String
- mType : String
- mMapOfProperties : Map<Integer , CristalDefinitionProperty>.

A.1.1.4 CristalDefinitionProperty

This is an abstract class for all definition properties. It defines the versioned-data which are common to all definition properties (i.e. it implements the VersionedObjectProperty of the Version pattern).

Attributes:

- mVersionId : Integer
- mDocumentation : String

A.1.2 Product Graph Meta-Model Package

This package contains classes used to build the product description graph or the so-called Product Breakdown Structure (PBS) (i.e. it is an implementation of the Enriched Directed Acyclic Graph (DAG) pattern shown in Figure 3-10 on page 42).

A.1.2.1 ProductDefinition (ProdDef)

Derived from CristalDefinition. This is an abstact class which defines the unversioned-data that are common to all definitions in the product description graph (i.e. it implements the ItemDescription of the Enriched DAG pattern).

Attributes:

- mIcon : BinaryFile

A.1.2.2 ElementaryProductDefinition (EProdDef)

Derived from ProductDefinition. This defines the unversioned-data for atomic elements in the product description graph (i.e. it implements the ElementaryItemDescription of the Enriched DAG pattern).

A.1.2.3 CompositeProductDefinition (CProdDef)

Derived from ProductDefinition. This defines the unversioned-data for composite elements in the product description graph (i.e. it implements the CompositeItemDescription of the Enriched DAG pattern).

A.1.2.4 ProductDefinitionProperty

Derived from CristalDefinitionProperty. This is an abstract class which defines the versioned-data that are common to all definitions in the product description graph.

Attributes:

- mWorkflowDefinitionId : Interger

A.1.2.5 ElementaryProductDefProperty

Derived from ProductDefinitionProperty. This defines the versioned-data for atomic elements in the product description graph.

A.1.2.6 CompositeProductDefProperty

Derived from ProductDefinitionProperty. This defines the versioned-data for composite elements in the product description graph.

Attributes:

- mCompositionLayout : ProductCompositionLayoutDef

A.1.2.7 ProductCompositionLayoutDef

This defines data to store the description of ProductDef composition with the view of the layout (in a VRML file). It maintains a map between ProductDefCompositeMembers and identifiers of volumes (VolumeId) in the VMRL file.

Attributes:

- mMapOfCompositeMembers : Map<VolumeId , ProductDefCompositeMember>
- mLayoutView : VRMLfile

A.1.2.8 ProductDefCompositeMember

This identifies one constituent of the composition description (i.e. it implements the CompositeMember of the Enriched DAG pattern).

Attributes:

- mIdentifier : Integer
- mTemporary : Boolean (tempoary member of the composition for tooling reason)

A.1.3 Workflow Graph Meta-Model Package

This package contains classes used to build the workflow description graph or the so-called Work Breakdown Structure (WBS) (i.e. it is an implementation of the Enriched Directed Acyclic Graph (DAG) pattern shown in Figure 3-10 on page 42).

A.1.3.1 ActivityDefinition (ActDef)

Derived from CristalDefinition. This is an abstact class which defines the unversioned-data that are common to all definitions in the workflow description graph (i.e. it implements the ItemDescription of the Enriched DAG pattern).

A.1.3.2 ElementaryActivityDefinition (EActDef)

Derived from ActivityDefinition. This defines the unversioned-data for atomic elements in the workflow description graph (i.e. it implements the ElementaryItemDescription of the Enriched DAG pattern).

A.1.3.3 CompositeActivityDefinition (CActDef)

Derived from ActivityDefinition. This defines the unversioned-data for composite elements in the workflow description graph (i.e. it implements the CompositeItemDescription of the Enriched DAG pattern).

A.1.3.4 ActivityDefinitionProperty

Derived from CristalDefinitionProperty. This is an abstract class which defines the versioned-data that are common to all definitions in the workflow description graph.

Attributes:

- mRepeatable : Boolean
- mMandatory : Boolean
- mExecutorAgentDefIdentifier : Integer
- mActivityOutcomeDefIdentifier : Integer (references one DataFormatDefinition)
- mMinimumDuration : Float (in minutes)
- mMaximumDuration : Float (in minutes)

A.1.3.5 ElementaryActivityDefProperty

Derived from ActivityDefinitionProperty. This defines the versioned-data for atomic elements in the workflow description graph.

A.1.3.6 CompositeActivityDefProperty

Derived from ActivityDefinitionProperty. This defines the versioned-data for composite elements in the workflow description graph. It is possible to define three different layouts/flows for one CActDef: process flow, data flow and transactional dependency (how this is achieved is not discussed further here)

Attributes:

mListOfLayouts : List<ProductCompositionLayoutDef>

A.1.3.7 ActivityCompositionLayoutDef

This abstract class is used to store the description of different layouts of a CActDef. It maintains a map between ActivityDefCompositeMembers and their identifiers in the layout.

Attributes:

- mMapOfCompositeMembers : Map<LayoutId , ActivityDefCompositeMember>

A.1.3.8 ActivityDefCompositeMember

This identifies one constituent of the composition description (i.e. it implements the CompositeMember of the Enriched DAG pattern).

Attributes:

- mIdentifier : Integer

A.1.4 Production Conditions Package

This package contains classes used to build the dependency between the product description graph and the workflow description graph.

A.1.4.1 ProductionCondition

This 'association class' defines the unversioned-data which are required to describe the dependency between product and workflow models (i.e. it implements the ConditionDescription of the Homomorphism pattern and the VersionedObject of the Version pattern).

A.1.4.2 ProductionCondProperty

This defines the versioned-data which are required to describe the dependency between product and workflow models (i.e. it implements the VersionedObjectProperty of the Version pattern).

Attributes:

- mConditions : List<WorkflowDefMember>

A.1.4.3 WorkflowDefMember

This maps one ActDef instance (attribute mWfActivityDefInstance) in the scope of the composition of the workflow definition (i.e. the CActDef that is assigned with the ProdDef, see Section A.1.2.1) with the conditions required for its execution.

Attributes:

- mWfActivityDefInstance : List<Integer>
- mStartConditions : List<StartCondition>
- mEndCondition : EndCondition
- mApplicableCenters : List<ApplicableCenter>
- mCommand : Command

A.1.4.4 StartCondition

This class stores the single ProdDef instance required to start the activity. It can only be used for CProdDefs (i.e. when the ProductionCondition is built for a CProdDef).

Attributes:

- mProductDefMemberPath : ProductDefMemberPath

A.1.4.5 ProductDefMemberPath

This identifies one ProductDefinition instance in the scope of the CompositeProductDefinition.

Attributes:

- mProductDefCompositeMemberIds : List<Integer>

A.1.4.6 EndCondition

This defines the nominal values that need to be checked against the outcome of an ActivityDefinition instance. It is a map between FieldDefinition instances (i.e. FieldDefs in the scope of the DataFormatDef associated with the ActivityDefinition, see Section A.1.3.4) with the applicable set of values.

Attributes:

- mNominalValues : Map<FieldDefInstance , NominalValues>

A.1.4.7 ApplicableCenter

Defines one centre where an activity can be executed.

Attributes:

- mCentreId : Integer

A.1.4.8 Command

This defines an instruction for an instrument so that it can execute the ActDef instance. This is a computer readable format of the ActDef description/documentation including values which are specific to the current ActDef instance.

A.1.5 DataFormat Meta-Model Package

This package contains classes used to build the data description graph. In other words this package describes the data structures used to collect data (i.e. it is an implementation of the Enriched Directed Acyclic Graph (DAG) pattern shown in Figure 3-10 on page 42).

A.1.5.1 DataFormatDefinition

Derived from CristalDefinition. This is an abstract class which defines the unversioned-data that are common to all definitions in the data description graph (i.e. it implements the Item-Description of the Enriched DAG pattern).

A.1.5.2 FieldDefinition

Derived from DataFormatDefinition. This defines the unversioned-data for atomic elements (e.g. C typedef) in the data description graph (i.e. it implements the ElementaryItemDescription of the Enriched DAG pattern).

A.1.5.3 DataRecordDefinition

Derived from DataFormatDefinition. This defines the unversioned-data for composite/nested (e.g. C structure) elements in the data description graph (i.e. it implements the CompositeItemDescription of the Enriched DAG pattern).

A.1.5.4 DataFormatDefProperty

Derived from CristalDefinitionProperty. This is an abstract class which defines the versioned-data that are common to all definitions in the data description graph.

A.1.5.5 FieldDefProperty

Derived from DataFormatDefProperty. This defines the versioned-data for atomic elements in the data description graph.

Attributes:

- mUnit : String
- mType : Enum{integer, float, string, binaryFile, timeStamp, arrayOfIntegers, arrayOfFloats};

A.1.5.6 DataRecordDefProperty

Derived from DataFormatDefProperty. This defines the versioned-data for composite elements in the data description graph.

Attributes:

- mCompositionLayout : DataRecordCompositionLayoutDef

A.1.5.7 DataRecordCompositionLayoutDef

This defines data to store the description of a DataRecordDef composition with a representation of it in XML format. It maintains a map between DataFormatDefCompositeMember and identifiers of the representation in the XML file.

Attributes:

mMapOfCompositeMembers : Map<PresentationId , DataFormatDefCompositeMember>

A.1.5.8 DataFormatDefCompositeMember

This identifies one constituent of the composition description (i.e. it implements the CompositeMember of the Enriched DAG pattern).

Attributes:

- mIdentifier : Integer

A.2 CRISTAL Model Package

This package defines classes that are used to instantiate the detector production specifications, described in Section A.1. These classes have complex functionalities as their purpose is to instantiate the definitions and manage the data generated during the production.

A.2.1 Product Tree Model Package

This package defines classes used to build the product tree (i.e. it is an implementation of the Complex Tree pattern shown in Figure 3-3 on page 35).

A.2.1.1 Product

This is an abstract class for all elements in the product tree (i.e. it implements the Node from the Complex Tree pattern). Its events are: register, reject, assignToComposition, deAssignFromComposition, ship and receive. It can determine the next possible Product actions (e.g. ship, receive, assign, deassign) in its scope. It also stores the identifier of CompProduct (barcode) to which the Product was assigned.

Attributes:

- mIdentifier : String (e.g. barcode)
- mDefinitionIdentifier : Integer
- mParentProductId : String
- mEvents : List <ProductEvent>

A.2.1.2 ElementaryProduct

Derived from Product. This defines an atomic element in the product tree (i.e. it implements the Leaf from the Complex Tree pattern).

A.2.1.3 CompositeProduct

Derived from Product. This defines a composite element in the product tree (i.e. it implements the Branch from the Complex Tree pattern). It maintains the composition (e.g allocate, deAllocate members) and it can determine the next possible CompositeProduct actions (e.g. allocateMemberProduct, deAllocateMemberProduct) in its scope.

Attribute:

- mComposoitionLayout : CompositeProductlayout

A.2.1.4 CompositeProductLayout

This stores the actual state of the composition. It uses the VRML file to store the corresponding definition.

Attributes:

- mActualComposition : List<CompositeProductMember>

A.2.1.5 CompositeProductMember

This references a single Product (e.g. by barcode in mMemberProductId) which was assigned to the composition.

Attributes:

- mSlotNumber : Integer (see Section A.1.2.8)
- mMemberProductId : String

A.2.2 Workflow Tree Model Package

This package defines classes used to build the workflow tree (i.e. it is an implementation of the Complex Tree pattern shown in Figure 3-3 on page 35).

A.2.2.1 Activity

This is an abstract class for all elements in the workflow tree (i.e. it implements the Node from the Complex Tree pattern). Its events are: start, finish, ignore, skip and repeat.

Attributes:

- mIdentifier : String

- mDefinitionIdentifier : Integer
- mEvents : List <ActivityEvent>

A.2.2.2 ElementaryActivity

Derived from Activity. This defines an atomic element in the workflow tree (i.e. it implements the Leaf from the Complex Tree pattern).

A.2.2.3 CompositeActivity

Derived from Activity. This defines a composite element in the product tree (i.e. it implements the Branch from the Complex Tree pattern). It manages activities in its scope and keeps track of their execution. It delegates activity management to composite activities and it can determine the next possible activitie(s) in its scope.

A.2.2.4 CompositeActivityLayout

This stores the actual composition layout which is generated from the different versions of CADef.

Attributes:

- mActualLayout : List<CompositeActivityMember>

A.2.2.5 CompositeActivityMember

This references one Activity (e.g. by its identifier in mMemberActivityId) in the layout of the CompositeActivity.

Attributes:

- mMemberActivityId : String

A.2.3 Production Management Package

A.2.3.1 ProductManager

The ProductManager implements the Mediator from the Enriched Mediator pattern shown in Figure 5-13 on page 78. It evaluates Start and End Conditions and processes requests from Agents of the form ‘What-Should-Be-Done-Next’ by consulting the Product and Workflow objects (which yield lists of next Activities and Product actions), prioritising the results and giving the list back to the requester. It also handles requests from Agents for activity executions by evaluating StartConditions and by delegating the execution to the Activity and for activity terminations by evaluating EndConditions and by delegating the termination to the Activity.

Attributes:

- mProduct : Product
- mActivities : List <Activity>