Handling Evolving Data Through the Use of a Description Driven Systems Architecture

F. Estrella[1], Z. Kovacs[1], J-M. Le Goff[2], R. McClatchey[1], and M. Zsenei[3]

[1]Centre for Complex Cooperative Systems, UWE, Frenchay, Bristol BS16 1QY UK
{Florida.Estrella, Zsolt.Kovacs, Richard.McClatchey}@cern.ch
[2]CERN, Geneva, Switzerland
Jean-Marie.Le.Goff@cern.ch
[3]KFKI-RMKI, Budapest 114, H-1525 Hungary
Marton.Zsenei@cern.ch

Abstract. Traditionally product data and their evolving definitions, have been handled separately from process data and their evolving definitions. There is little or no overlap between these two views of systems even though product and process data are inextricably linked over the complete software lifecycle from design to production. The integration of product and process models in an unified data model provides the means by which data could be shared across an enterprise throughout the lifecycle, even while that data continues to evolve. In integrating these domains, an object oriented approach to data modelling has been adopted by the CRISTAL (Cooperating Repositories and an Information System for Tracking Assembly Lifecycles) project. 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 object persistence, flexibility, reusability, schema evolution and versioning of data elements. This paper describes the model that has been developed in CRISTAL and how descriptive meta-objects in that model have their persistence handled. It concludes that adopting a description-driven approach to modelling, aligned with a use of suitable object persistence, can lead to an integration of product and process models which is sufficiently flexible to cope with evolving data definitions.

Keywords: Description-Driven systems, Modelling change, schema evolution, versioning

1. Introduction

This study investigates how evolving data can be handled through the use of description-driven systems. Here 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 is interpreted by applications in order to achieve the domain-specific goals. In a description-driven system definitions 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 loosely coupled and coupling only takes place when instances are created or when a definition,

corresponding to existing instantiations, is modified. The coupling is loose since the lifecycle of each instantiation is independent from the lifecycle of its corresponding definition.

Description-driven systems (sometimes referred to as meta-systems) are acknowledged to be flexible and to provide many powerful features including (see [1], [2], [3]): Reusability, Complexity handling, Version handling, System evolution and Interoperability. This paper introduces the concept of description-driven systems in the context of a specific application (the CRISTAL project), discusses other related work in handling schema and data evolution, 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 how these structures enable the handling of evolving data.

2. Related Work

This section places the current work in the context of other schema evolution work. A comprehensive overview of the state of the art in schema evolution research is discussed in [4]. Two approaches in implementing the schema change are presented. The internal schema evolution approach uses schema update primitives in changing the schema. The external schema evolution approach gives the schema designers the flexibility of manually changing the schema using a schema-dump text and importing the change onto the system later.

A schema change affects other parts of the schema, the object instances in the underlying databases, and the application programs using these databases. As far as object instances are concerned, there are two strategies in carrying out the changes. Immediate conversion strategy immediately converts all object instances. Deferred conversion strategy takes note of the affected instances and the way they have to be changed, and conversion is done when the instance is accessed. None of the strategies developed for handling schema evolution complexity is suitable for all application fields due to different requirements concerning – permanent database availability, real-time requirements and space limitations. More so, existing support for schema evolution in current OODB systems (O2, GemStone, Itasca, ObjectStore) is limited to a pre-defined set of schema evolution operations with fixed semantics [5]. CRISTAL adheres to a modified deferred conversion strategy. Changes to the production schema are made available upon request of the latest production release.

Several novel ideas have been put forward in handling schema evolution. Graph theory is proposed as a solution in detecting and incorporating schema change without affecting underlying databases and applications [6] and some of these ideas have been folded into the current work. An integrated schema evolution and view support system is presented in [7]. In CRISTAL there are two types of schemas – the user's personal view and the global schema. A user's change in the schema is applied to the user's personal view schema instead of the global schema. The persistent data is shared by multiple views of the schema.

To cope with growing complexity of schema update operations in most current applications, high level primitives are being provided to allow more complex and advanced schema changes [8]. Along the same line, the SERF framework [5] succeeds in giving users the flexibility to define the semantics of their choice and the extensibility of defining new complex transformations. SERF claims to provide the first extensible schema evolution framework. Current systems supporting schema evolution concentrate on changes local to individual types. The Type Evolution Software System (TESS) [8] allows for both local and compound changes involving multiple types. The old and the new schema are compared, a transformation function is produced and updates the underlying databases. The need for evolution taxonomy involving relationships is very much evident in OO systems. A set of basic evolution primitives for uni-directional and bi-directional relationships is presented in [9]. No work has been done on schema evolution of an object model with relationships prior to this research.

Another approach in handling schema updates is schema versioning. Schema versioning approach takes a copy of the schema, modifies the copy, thus creating a new schema version. The CRISTAL schema versioning follows this principle. In CRISTAL, schema changes are realised through dynamic production re-configuration. The process and product definitions, as well as the semantics linking a process definition to a product definition, are versionable. Different versions of the production co-exist and are made available to the rest of the CRISTAL users.

An integration of the schema versioning approach and the more common adaptational approach is given in [10]. An adaptational approach, also called direct schema evolution approach, always updates the schema and the database in place using schema update primitives and conversion functions. The work in [10] is the first time when both approaches have been integrated into a general schema evolution model.

3. The CRISTAL Project

A prototype has been developed which allows the study of data evolution in product and process modelling. The objective of this prototype is to integrate a Product Data Management (PDM) model with a Workflow Management (WfM) model in the context of the CRISTAL (Cooperating Repositories and Information System for Tracking Assembly Lifecycles) project currently being undertaken at CERN, the European Centre for Particle Physics in Geneva, Switzerland. See [11, 12, 13, 14].

The design of the CRISTAL prototype was dictated by the requirements for adaptability over extended timescales, for schema evolution, for interoperability and for complexity handling and reusability. In adopting a description-driven design approach to address these requirements, 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.

The CRISTAL meta-model is comprised of so-called 'meta-objects' each of which is defined for a class of significance in the data model: e.g part definitions for parts,

activity definitions for activities, and executor definitions for executors (e.g instruments, automatically-launched code etc.).

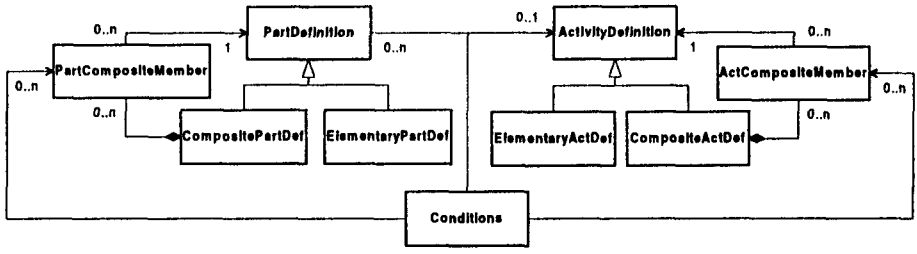

Figure 1. Subset of the CRISTAL meta-model.

Figure 1 shows a subset of the CRISTAL meta-model. In the model information is stored at specification time for types of parts or part definitions and at assembly time for individual instantiations of part definitions. At the design stage of the project information is stored against the definition object and only when the project progresses is information stored on an individual part basis. This meta-object approach reduces system complexity by promoting object reuse and translating complex hierarchies of object instances into (directed acyclic) graphs of object definitions. Meta-objects allow the capture of knowledge (about the object) alongside the object themselves, enriching the model and facilitating self-description and data independence. It is believed that the use of meta-objects provides the flexibility needed to cope with their evolution over the extended timescales of CRISTAL production. Discussion on how the use of meta-objects provides the flexibility needed to cope with evolving data in CRISTAL through the so-called Production Scheme Definition Handler (PSDH) is detailed in a separate section of this paper.

Figure 2 shows the software architecture for the definition of a production scheme in the CRISTAL Central System. It is composed of a Desktop Control Panel (DCP) for the Coordinator, a GUI for creating and updating production specifications and the PSDH for handling persistency and versioning of the definition objects in the Configuration Database.

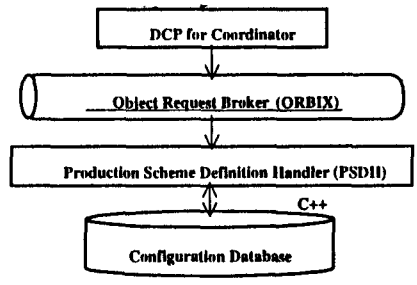

Figure 2. Software architecture of definition handler in Central System.

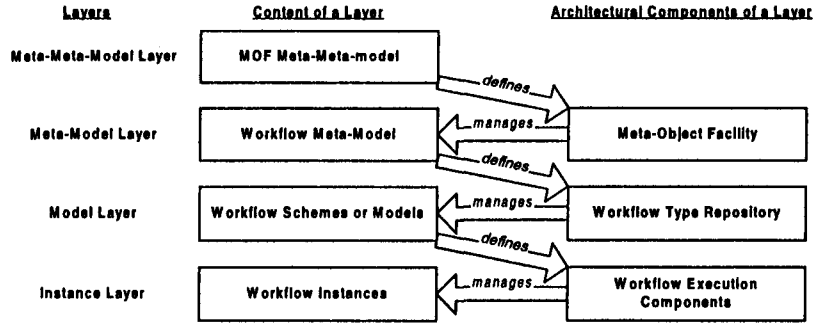

Figure 3. A 4-layer meta-modeling architecture.

4. The Layered Architecture of Description Driven Systems

The concept of models which describe other models has come to be known as '*meta-models*' and is gaining wide acceptance in the world of object-oriented design. One example of a system which uses a multi-layer architecture is that of a WfM system [15]. 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 describes the workflow instances and corresponds to the next layer of abstraction - the model layer. The information about a model is generally described as meta-data. In order for the workflow scheme itself to be built, a further model is required to capture/hold the semantics 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 [16].

The semantics required to adequately model the information in the application domain of interest will in most cases be different. 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. [17]). 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 [18]. Figure 3 illustrates the four layer meta-modeling architecture adopted by the OMG and based on the ISO 11179 standard.

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. 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 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.

5. Features of Description Driven Systems

It is the basic tenet of this study that the desirable features of description-driven systems can be realised through the adoption of a flexible multi-layered system architecture. This section examines each required 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 can be reused for multiple applications.
- *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, to alleviate data management. Scalability can be eased, and a potential explosion in the number of products to be managed can be 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.
- *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.
- *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 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.

6. Repository-Based Handling of Description

The CRISTAL system is composed of a Central System whose purpose is to control the overall production process and several Local Centres at which production (assembly and testing) is carried out. The overall system Coordinator at the Central System defines the design or configuration of the different detector components, called the Detector Production Scheme (DPS). The Coordinator is responsible for supplying the production scheme to the different Local Centres. The local Operator at each Local Centre applies the production scheme to the local production line.

The design of the detector changes over time and many versions of the production exists. The DPS object stores the production scheme version number, and the list of definition objects used in that version. DPS is also versionable, thus creating a linear production scheme version geneology, storing the history of the different definitions and the history of the different production schemes. Changes to the definitions are allowed. This is done centrally and made available to the rest of the system via the next supply of the configuration by the Coordinator. After each supply, the Coordinator has the option to start with an empty production scheme or from the last supplied production scheme. The supply automatically creates a new version of the DPS object, and attaches it to the end of the version geneology.

Each definition object is divided into two parts – definition attributes (not versioned) and definition properties (versioned). New definition properties are created whenever the definition object is updated for the first time in a new production scheme. The version table, which tracks which production scheme version a property has been created for, is part of the attributes. Figure 4 shows the organisation of the different versions of persistent CRISTAL objects.

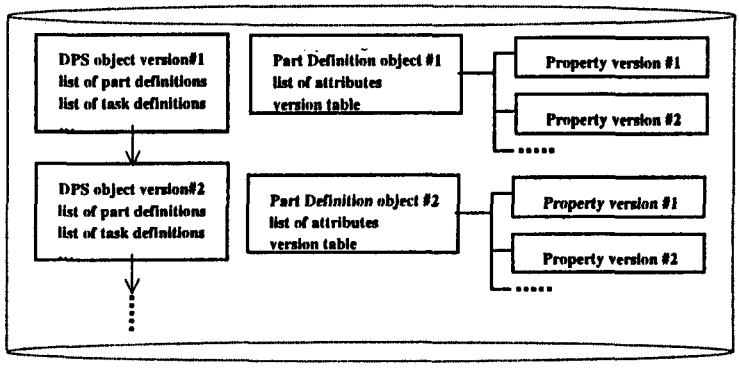

Figure 4. Organisation of persistent CRISTAL definition objects.

The persistent definition objects are stored using Objectivity. The PSDH is responsible for handling the production scheme versions and the definition versions.

The PSDH is implemented using C++ and is a CORBA service layered on top of the DB, providing the different Coordinator functionalities. Each definition object is assigned a unique identification number (OID) generated by the PSDH. To access a particular definition object, the OID is given, plus the production scheme version of the property. The version table is scanned to locate the correct property, if it exists, and the full definition is then loaded into the user interface (DCP) of the Coordinator.

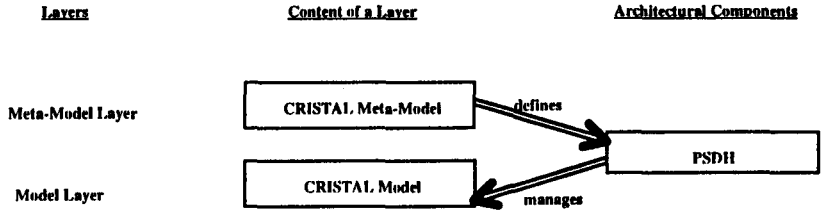

Figure 5. Subset of CRISTAL Configuration Architecture

The PSDH and its relation to the different layers of the CRISTAL architecture is shown in Figure 5. The CRISTAL meta-model is an abstraction of the CRISTAL model. It captures and holds the semantics for the generation of the CRISTAL model. It describes the elements for overall production scheme management and embodies the adopted versioning policies in CRISTAL. The CRISTAL model is composed of the definition classes and the associations between these classes. The PSDH provides the mechanisms for production specification management. It describes and manages the complete lifecycle of all CRISTAL definitions used to specify the design of a sub-detector.

The PSDH allows for the co-existence of many versions of the production scheme. It caters for dynamic design upgrade, that is, the creation of new detector components or changes in existing detector specification, without the need for the production line to stop so that a new scheme can be uploaded. Likewise, the history of the versions of the configuration is stored in the database, allowing users to access historical data. Consequently, evolving design knowledge is managed transparently without the need for the production line to be flushed.

7. Conclusions

It is apparent that the description-driven approach, advocated in this paper, reduces system complexity by promoting object reuse 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-WfMS

integration. The CRISTAL system has demonstrated that evolving data and persistence can be handled using a description driven system approach.

The experience of using meta models and meta objects at the analysis and design phase in the CRISTAL project has been very positive. Designing the meta model separately from the runtime model has allowed the design team to provide consistent solutions to dynamic change and versioning. The object models are described using UML [19] which itself can be described by the OMG Meta Object Facility [20] and is the candidate choice by OMG for describing all business models.

In distributed object-based systems, object request brokers, such as the Object Management Group's CORBA [21] provide for the exchange of simple data types and, in addition, provide location and access services. The CORBA standard is meant to standardise how systems interoperate. OMG's CORBA Services specify how distributed objects should participate and provide services such as naming, persistent storage, lifecycle, transaction, relationship and query. The CORBA Services standard is an example of how self-describing software components can interact to provide interoperable systems.

The current phase of CRISTAL research aims to adopt an open architectural approach, based on a meta-model and an extraction facility to produce an adaptable system capable of interoperating with future systems and of supporting views onto an engineering data warehouse. The meta-model approach to design reduces system complexity, provides model flexibility and can integrate multiple, potentially heterogeneous, databases into the enterprise-wide data warehouse. A first prototype for CRISTAL based on CORBA, Java and Objectivity technologies has been deployed in the autumn of 1998 [22]. The second phase of research will culminate in the delivery of a production system in 1999 supporting queries onto the meta-model and the definition, capture and extraction of data according to user-defined viewpoints.

Recently a considerable amount of interest has been generated in meta-models and meta-object description languages. Work has been completed within the OMG on the Meta Object Facility which is expected to manage all meta-models relevant to the OMG Architecture. 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 modelling 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). The next phase of CRISTAL research intends to help verify this.

8. Acknowledgments

The authors take this opportunity to acknowledge the support of their home institutes, In particular, the support of P. Lecoq, J-L. Faure and M. Pimia is greatly appreciated. Richard McClatchey is supported by the Royal Academy of Engineering and Marton Zsenei by the OMFB, Budapest. N. Baker, A. Bazan, T. Le Flour, S.

Lieunard, L. Varga, S. Murray, G. Organtini and G. Chevenier are thanked for their assistance in developing the CRISTAL prototype.

References

1 IEEE Meta-Data Conferences 1996, 1997 & 1999. Silver Spring, Maryland. See http://www.computer.org/conferen/meta96/paper_list.html, //computer.org/conferen/proceed/meta97/ and //computer.org/conferen/proceed/meta/1999/
2 S. Crawley et al., "Meta-Information Management". Proc of the Int Conf on Formal methods for Open Object-based Distributed Systems (FMOODS), Canterbury, UK. July 1997
3 N. Baker & J-M Le Goff., "Meta Object Facilities and their Role in Distributed Information Management Systems". Proc of the EPS ICALEPCS97 conference, Beijing, November 1997. Published by Science Press, 1997.
4 S. Lautemann, "An Introduction to Schema Versioning in OODBMS". Proc of the 7th Intl Conf on Database and Expert Systems Applications (DEXA), Zurich, Switzerland. September 1996.
5 K. Claypool et. al., "SERF: Schema Evolution through an Extensible, Re-usable and Flexible Framework". Technical Report, Department of Computer Science, Worcester Polytechnic Institute, 1998.
6 S. Ram et al., "Dynamic Schema Evolution in Large Heterogenous Database Environments". Advanced Database Research Group report. University of Arizona, 1997.
7 Y. Ra et. al., "A Transparent Schema Evolution System Based on Object Oriented View Technology". IEEE Trans on Data and Knowledge Engineering. September 1997.
8 B. Lerner, "A Model for Compound Type Changes Encountered in Schema Evolution". Technical Report, Computer Science Department, University of Massachusetts. June 1996.
9 K. Claypool et. al., "Extending Schema Evolution to Handle Object Models with Relationships". Computer Science Technical Report Series, Worcester Polytechnic Institute, March 1999.
10 F. Ferrandina et. al., "An Integrated Approach to Schema Evolution for Object Databases". Proc of the 3rd Intl Conf on Object Oriented Information Systems (OOIS). December 1996.
11 CMS Technical Proposal. The CMS Collaboration, January 1995. Available from ftp://cmsdoc.cern.ch/TPref/TP.html
12 R. McClatchey et al., "The Integration of Product and Workflow Management Systems in a Large Scale Engineering Database Application". Proc of the 2nd IEEE IDEAS Symposium. Cardiff, UK July 1998.
13 N. Baker et al., "An Object Model for product and Workflow Data Management". Workshop proc. At the 9th Int. Conference on Database & Expert System Applications. Vienna, Austria August 1998.
14 F. Estrella et al., "The Design of an Engineering Data Warehouse Based on Meta-Object Structures". Lecture Notes in Computer Science Vol 1552 pp 145-156 Springer Verlag 1999.
15 W. Schulze., "Fitting the Workflow Management Facility into the Object Management Architecture". Proc of the Business Object Workshop at OOPSLA'97.
16 W. Schulze, C. Bussler & K. Meyer-Wegener., "Standardising on Workflow Management - The OMG Workflow Management Facility". ACM SIGGROUP Bulletin Vol 19 (3) April 1998.
17 S. Crawley et al., "Meta-Meta is Better-Better!". In proc. of the IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS'97). 1997
18 B. Byrne., "IRDS Systems and Support for Present and Future CASE Technology". In proc. of the CaiSE'96 International Conference, Heraklion, Crete.

19 M. Fowler & K. Scott: "UML Distilled - Applying the Standard Object Modelling Language", Addison-Wesley Longman Publishers, 1997.
20 Object Management Group Publications, Common Facilities RFP-5 Meta-Object Facility TC Doc cf/96-02-01 R2, Evaluation Report TC Doc cf/97-04-02 & TC Doc ad/97-08-14
21 OMG, "The Common Object Request Broker: Architecture & Specifications". OMG publications, 1992.
22 A. Bazan et al., "The Use of Production Management Techniques in the Construction of Large Scale Physics Detectors". IEEE Transactions in Nuclear Science, in press, 1999.