System Evolution via Model-driven Design

A. Branson[1], J.-M. Le Goff[2], R. McClatchey[1] and J. Shamdasani[1]
[1]*Centre for Complex Cooperative Systems, University of the West of England, Coldharbour Lane, Bristol, U.K.*
[2]*CERN, 1217 Geneva 23, Switzerland*

Keywords: Description-driven, Model Evolution, Dynamic Requirements, System Reconfiguration.

Abstract: Software engineers frequently face the challenge of systems whose requirements change over time to adapt to organizational reconfigurations or external pressures. Evolving user requirements present a considerable software engineering challenge, all the more so in an environment in which business agility demands shorter development times and responsive prototyping. This paper presents a system called CRISTAL which was developed at CERN in Geneva. CRISTAL is based on the concept of so-called 'description-driven' change management that enables system development which is responsive to changes in user requirements and facilitates dynamic system reconfiguration as required.

SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

The principal motivation in carrying out this research was to address the requirements scientists working on the Large Hadron Collider (LHC) accelerator project at CERN which reached its final phase of construction and testing in 2009. The Compact Muon Solenoid (CMS) (The CMS Collaboration and Chatrchyan, S. et al., 2008) is one of the four main experiments of the LHC; it contains the so-called Electromagnetic Calorimeter (ECAL) detector. Its design and construction were carried out between 1995 and 2008 and form the subject of this paper.

A research project, entitled CRISTAL (Cooperating Repositories and an Information System for Tracking Assembly Lifecycles) (Estrella, F. et al., 2003) was established, using pure object oriented computing technologies where possible, to facilitate the management of the engineering data collected at each stage of construction of CMS. CRISTAL is a distributed product data and workflow management system which uses a database for its repository, a multi-layered architecture for its component abstraction and dynamic object modelling for the design of the objects and components of the system. These techniques are critical in handling the complexity of such a data-intensive system and to provide the flexibility to adapt to the changing production scenarios typical of any research production system. The CRISTAL system has been based on a so-called description-driven approach in which all logic and data structures are described by meta-data, which can be modified and versioned online as the design of the detector changes. A description-driven system (DDS) architecture (Estrella, F. et al., 2003) is an example of a reflective meta-layer architecture. DDSs make use of meta-objects to store domain-specific system descriptions (e.g. items, processes, lifecycles, goals, agents and outcomes) which control and manage the lifecycles of instances or domain objects. As objects, reified system descriptions of DDSs can be organized into libraries conforming with frameworks for modelling of languages, and to their adaptation for specific domains.

The meta-data and the instantiated elements of data are stored in the database; the evolution of the design is tracked by versioning the changes in the meta-data over time. Thus description-driven systems make use of meta-objects to store domain-specific system descriptions that control and manage the lifecycles of domain objects (i.e. instances of the meta-object definitions). The separation of the descriptions from their instances allows them to be specified and managed and to evolve independently and asynchronously. This separation is essential in handling the complexity issues facing many web-based computing applications and facilitates interoperability and reusability with system evolution. Separating descriptions from their instantiation allows new versions of defined objects (and their descriptions) to co-exist with older versions. The reader is directed to previous publications on DDS for further background

(Estrella, F. et al., 2003; Estrella, F. et al., 2001).

At the outset of CRISTAL an approach was followed that enabled both ECAL parts and the activities (both with the metadata of their specifications) that were carried out on the parts to be saved side-by-side in a structured database. In this way as the scientists developed their research ideas we were able to capture each design version and those parts and activities that were processed for that version. The production version of CRISTAL was designed to satisfy a set of requirements covering both the CMS ECAL needs and the needs of commercial Business Process Management (BPM) users. This led to a generalization of the concept of 'parts' to that of 'items' which were intended to be applicable to describe any element of any business process. The intention was to enable developers to define the objects central to the operation of any business and to drive the design process from the standpoint of how those objects ('items') change over time.

CRISTAL is, in essence, an application server that abstracts all of its business objects into workflow driven, version controlled 'Items' which are instantiated from descriptions stored in other Items (see Figure 1) and are managed on-the-fly for target user communities. Items contain:

- Workflows, that comprise of Activities to be executed by Agents (either human users or mechanical/ computational agents via an API), which then generate:
- Events that detail each change of state of an Activity. Completion events generate data, stored as:
- Outcomes which are XML documents from each execution, for which:
- Viewpoints refer to particular versions (e.g. the latest version or, in the case of descriptions, a particular version number).
- Properties are name/value pairs that name and type items, they also denormalize collected data for more efficient querying, and
- Collections that enable items to be linked to each other.

The CMS ECAL was made of thousands of similar parts, all needing characterizing and assembling in an optimal configuration based on sets of detailed measurements. Every component part was registered as an Item in the CRISTAL database, each with its barcode as an identifier, stored as the CRISTAL property 'Name'. Each part had a type, which functioned as the item description, and was linked to the workflow definition that each instance would follow in order to collect its data and mount sub-parts. All collected data and assembly information were stored as outcomes attached to events, therefore the entire history of every interaction with the application was recorded. The result was a set of items representing the top level components of the detector which contained levels of substructure, all with their full production history with all collected and calculated production data attached in the correct context. On creation, an item contains no events, outcomes or viewpoints (the context of the item). These are explicitly generated later during execution of its workflow processes. Initially an item contains properties to identify it, its workflow and any collections of items it may need, with all slots empty. The initial set of properties are created in the process of rendering a property input form by exploiting the property description (itself an outcome stored in the item description), and submitting it as an activity outcome of the description item.

The basic functionality of CRISTAL is best illustrated with an example: using CRISTAL a user can define product types (such as Newcar spark plug) and products (such as a Newcar spark plug with serial number #123), workflows and activities (e.g. to test that the plugs work properly, and mount them into the engine). This allows products that are undergoing workflow activities to be traced and, over time, for new product types (e.g. an improved Newcar spark plug) to be defined which are then instantiated as products (e.g. updated Newcar spark plug #124) which are traced in parallel to pre-existing ones. The application logic is free to allow or deny the inclusion of older product versions in newer ones (e.g. to use up the old stock of spark plugs). Similarly, versions of the workflow activities can co-exist and be run on these products. All versions of items in the production history are simply viewed as items allowing the capture of many versions of items and their coexistence in CRISTAL. Further detail of the the CRISTAL database can be found in (Branson, A. et al., 2012).

2 CRISTAL IN PRACTICE

This section outlines how the CRISTAL system performed in practice at CERN. The CRISTAL software was developed over the period 1997-2000 and was delivered for use at CMS early in 2000, when the data from the characterization and the physical measurement of the 70,000 lead tungstate crystals began. CRISTAL collected initial data against a data model residing in Objectivity but it suffered from numerous performance limitations. After significant redesign CRISTAL V2 was made available to the CMS ECAL

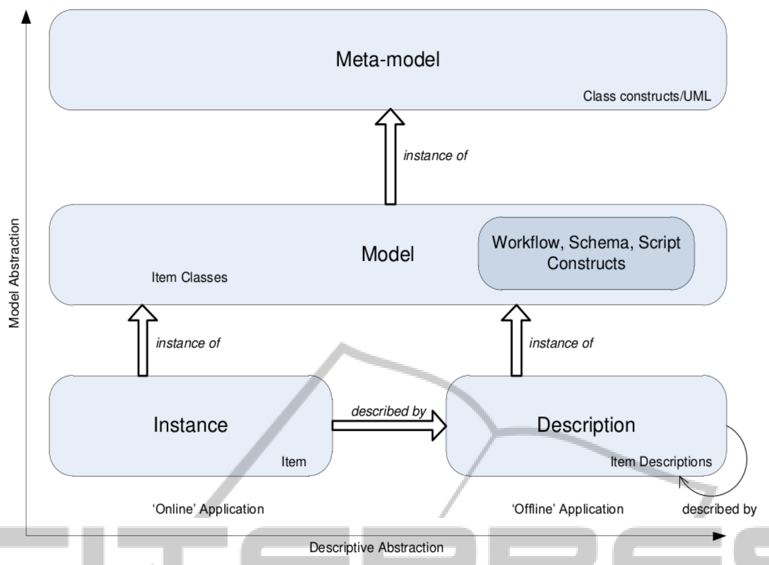

Figure 1: Model vs. description in Cristal 2.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

user community from 2003 and it remained in operation over the following five to six years. Each ECAL crystal generated between 2-3Mbytes of information which was mainly gathered in an automated data acquisition system which characterised the crystals in batches over a period of 8-10 hours for each batch of 30 crystals. Due to the vast numbers of crystals (100,000s) to measure it was important that this data acquisition process was reliable, consistent and that the mean time between failures was very high. Some batches required re-characterisation and some crystals needed to be replaced; the whole data acquisition process took around five years to complete. It was the responsibility of one CRISTAL software engineer (the so-called application maintainer) to ensure as smooth operation as possible of the data acquisition and to provide round-the-clock accessibility to the CRISTAL database. Where changes were required to the descriptions handled by CRISTAL the procedure outlined above was followed.

In the CMS experiment, production models change over time. Detector parts of different model versions must be handled over time and coexist with other parts of different model versions. Separating details of model types from the details of single parts allows the model type versions to be specified and managed independently, asynchronously and explicitly from single parts. In capturing descriptions separate from their instantiations, system evolution can be catered for while production is underway this provides continuity in the production process and design changes can be reflected quickly into production. The approach of reifying a set of simple design patterns as the basis of the description-driven architecture for CRISTAL has provided the capability of catering for the evolution of a rapidly changing research data model. In the six years of operation of CRISTAL it has gathered over 50 GBytes of data and been able to cope with 25 evolutions of the underlying data schema without code or schema recompilations. During the years of near-continuous operation, the descriptions went from beta to production then through years of (relatively few) alterations of the domain logic which necessitated very little change in the actual server software, illustrating the flexibility of the CRISTAL approach.

3 LESSONS LEARNT FROM BUILDING CRISTAL

Probably the main lesson learned from the CRISTAL project in coping with change was to develop a data model that had the capacity to cover multiple types of data (be they products or activities, atomic or composite in nature) and at the same time was elegant in its simplicity. To do this a systemically consistent approach to data modelling was needed: designers needed to think in a way that would facilitate system flexibility, would enable rapid change and would ease the burden of maintenance from the outset of the design process. The approach that was followed in designing CRISTAL was to concentrate on the essential enterprise objects and descriptions (products, activities, user roles, outcomes, events) that could be needed during the lifetime of the system no matter from which viewpoint that data is accessed. Thus the system was allowed to be open in design and flexible in nature and the elegance of its design was not limited from being viewed from one or several application-led standpoints (e.g. BPM, EAI, CRM, PDM,WfMS). Rather we enabled the traceability of enterprise objects over the lifetime of the system as the primary goal of the system and left the application-specific views to be defined as and when they became required. It is this that we see as the main and unique contribution of the CRISTAL approach (a 'description-driven' one) to building flexible and maintainable systems and we believe this makes a significant contribution to how moel-driven enterprise systems can be implemented. These were not simple design skills; designers needed to be able to think conceptually, abstracting the characteristics of everyday objects into 'items' with associated metadata and to be able to represent that complexity in a concrete data model. Great benefits in terms of maintainability and flexibility resulted from being able to treat many different system objects (workflows, events, outcomes, items) in a single standardised manner. The importance of instantiation and description in formulating a generic CRISTAL data model cannot be overemphasised. They are the foundations on which description-driven systems development is based.

Importance was placed on the involvement of users at all stages of the development of CRISTAL. We regard this as one of the prime reasons for the eventual success of the project. Although initially it was hoped that high-end expert users would be able to develop workflows themselves, in practice this was not possible. Instead the users collaborated closely with the designers to establish a much clearer idea of the implications of their requirements, and with a full understanding of the functionality that their workflow must provide. This could then be implemented with verifiable accuracy against what the user originally wanted. Essentially this approach, centred on the identification of items and their descriptions, led to a very intuitive way of representing requirements and absorbing them, as and when they emerged, into the evolving data model. On the negative side the users necessarily did not always know at the outset what their final requirements would be for data and process management, often leading to disruptive changes in design decisions. On the positive side, the users were not locked into a 'static' product: CRISTAL was evolving to cater for their requirements and could be made responsive to their needs.

Control of evolving requirements was a particularly challenging problem. New user requirements needed to be addressed at the application level which, as a consequence, induced requirements at the domain implementation level which in turn passes its own requirements down to the kernel level. The result of this was that there could be a considerable number of potential feature configurations of the CRISTAL kernel needed to meet all possible requirements from the user. Since CRISTAL was conceived as a model-driven system, an attempt was made to follow best software engineering practice in implementing features associated with object orientation (e.g. inheritance, polymorphism, deferral of commitment, etc) to ensure reuse and extensibility. Whenever a new design modification was needed, the approach taken therefore was always to implement as open and as flexible a solution as the design allowed in order not to constrain future extensions. In practice, however, this "second guessing quickly" led to feature creep and spiralling complexity which was at risk of compromising the system development process. To address this situation the approach that we adopted was to make the implementation of new requirements as intuitive as possible with as simple functionality as necessary to cope with the requirements, thereby preserving the elegance of the original (description-driven) design. This led to a closely connected set of system functionalities which was easy to maintain and to dynamically extend as and when needed. In addition this much simpler system has the virtue of being significantly easier for users, developers and administrators new to the system to pick up and start working with. The flexible nature of the DDS approach has recently enabled the use of CRISTAL as a provenance management tool in the neuGRID project (for studies of Alzheimer's disease) (Anjum, A. et al, 2010).

REFERENCES

Anjum, A. et al (2010). Reusable Services from the neuGRID Project for Grid-Based Health Applications. *Studies in Health Technology and Informatics*, 147:283–288.

Branson, A. et al. (2012). Evolving Requirements: Model-Driven Design for Change. *Information Systems*. Under final review.

Estrella, F. et al. (2001). Meta-data Objects as the Basis for System Evolution. In *Proceedings of the Second International Conference on Advances in Web-Age Information Management*, WAIM '01, pages 390–399. Springer-Verlag.

Estrella, F. et al. (2003). Pattern Reification as the Basis for Description-Driven Systems. *Journal of Software and System Modelling*, 2(2):108–119.

The CMS Collaboration and Chatrchyan, S. et al. (2008). The CMS experiment at the CERN LHC. *Journal of Instrumentation*, 3:S08004.