Design Patterns for Description-Driven Systems

N. Baker[3], A. Bazan[1], G. Chevenier[2], Z. Kovacs[3], T Le Flour[1], J-M Le Goff[4], R. McClatchey[3] & S Murray[1]

[1]LAPP, IN2P3, Annecy-le-Vieux, France
[2]HEP Group, ETHZ, Zurich, Switzerland
[3]Centre for Complex Cooperative Systems, Univ. West of England, Bristol BS16 1QY, UK
[4]EP Division, CERN, 1211 Geneva 23, Switzerland

Abstract

In data modelling, product information has most often been handled separately from process information. The integration of product and process models in a unified data model could provide the means by which information could be shared across an experiment throughout the system lifecycle from design through to maintenance. This paper relates description-driven systems to multi-layer architectures and reveals where existing design patterns facilitate the integration of product and process models. The CRISTAL system is being used to store the physics data gathered during HEP detector construction and to track the progress of CMS detector assembly. Data stored in the CRISTAL data warehouse will provide the detector geometry and facilitate calibration as well as providing a CMS detector knowledge base for physics reconstruction and analysis programmes.

Keywords: Design Patterns, description-driven systems, data modelling, UML

1 Description-Driven Systems and Multi-Layer Architectures

'Description-driven systems' can be defined as systems in which the description of a domain-specific configuration is captured in a computer-readable form. This description can be interpreted by applications to achieve domain-specific goals. In a description-driven system descriptions are separated from instances and managed independently - descriptions can be specified and can evolve asynchronously from particular instantiations of those descriptions. As a consequence a description-driven system requires computer-readable models both for descriptions and for instances. These models are loosely coupled and coupling only takes place when instances are created or when a description, corresponding to existing instantiations, is modified. The coupling is loose since the lifecycle of each instantiation is independent from the lifecycle of its corresponding description.

The semantics required to adequately model application-specific information will, in most cases, be different. For example, the semantics for describing Product Data Management (PDM) systems will be very different from those describing WfM systems. To facilitate integration between meta-models a universal type language capable of describing all meta-information is required. 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*. The accepted conceptual framework for meta-modeling is based on an architecture with four layers.

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

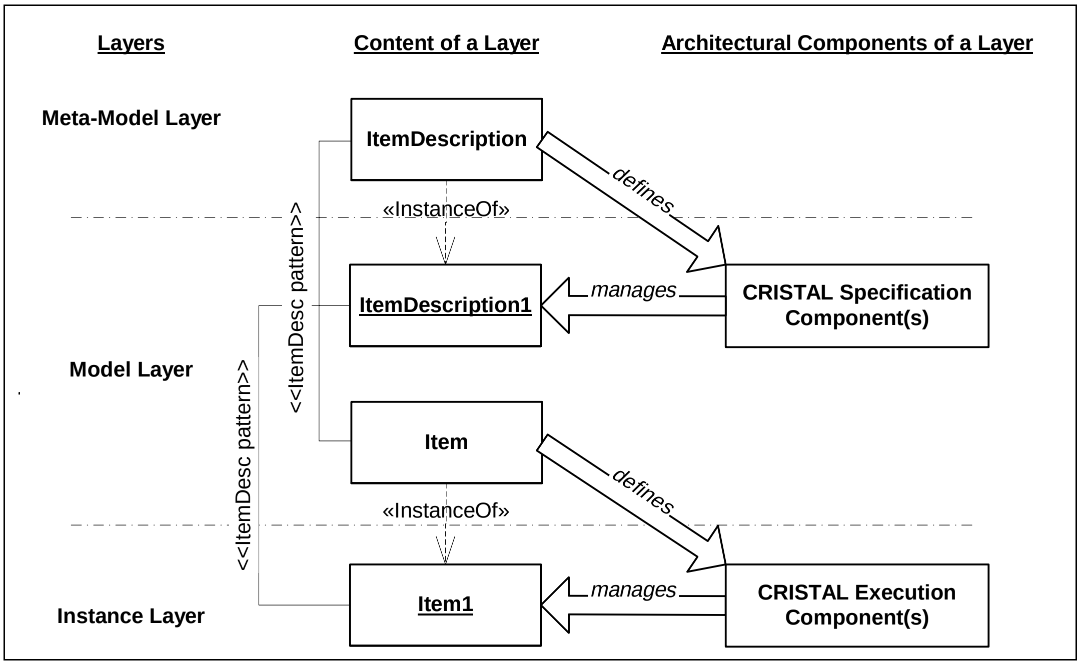

Figure 1: The CRISTAL three-layer architecture.

terms of the meta-meta types of the meta-meta modeling layer above. Examples of objects at this level from manufacturing 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. Example objects for the manufacturing domain would be product, production schedule, composite product. At the lowest level user objects are an instance of a model and describe a specific information and application domain.

A recent thesis [1] studied the integration of product data and workflow management through a common description-driven data model for the CRISTAL project. In building the data model a set of design patterns [2] were identified including the item description pattern, an enriched directed acyclic graph pattern, a publish/subscribe pattern, a version pattern, an enriched homomorphism pattern and use of a mediator pattern for retrieval of information from the integrated product and process data model. It was speculated that these design patterns are part of the essential elements of any description-driven system. Figure 1 shows how the item description pattern allows the dependency between a description and its instantiation to be handled. In the figure the item description pattern has been employed to provide the semantics that relate the Item class (of the model layer) with the ItemDescription class of the meta-model layer. 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 forms the basis of a description-driven system is a direct consequence of the use of the ItemDescription pattern. Later in this paper these patterns are used to build an ontology which facilitates not only product and process integration, but the development of an enterprise model which spans multiple domains.

Description-driven systems features can be realised through the adoption of a multi-layered architecture. Description-driven systems are flexible and provide many powerful features including reusability, complexity handling, versioning, system evolution and interoperability The study reported in this paper investigates how product and process information can be handled through the use of a common meta-model in a so-called description-driven system. The description-driven approach is outlined and its role in integrating product and process models for a data warehouse example is identified. The resulting meta-model is general in form and can be used to produce materialised views (so-called ‘viewpoints’) onto the data warehouse.

Essentially the meta-model can be used as the basis of an ontology describing how products and processes are inter-related, as it is predicated on a set of basic concepts and their relationships. A prototype has been developed which facilitates this study of a common ontology for product and process modeling. The objective of this prototype is to integrate a Product Data Management model with a Workflow Management model in the context of the CRISTAL (Cooperating Repositories and Information System for Tracking Assembly Lifecycles) project.

2 The CRISTAL Project

The design of the CRISTAL prototype was dictated by the requirements for adaptability over extended timescales, for system 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 assembly of CMS is being carried out by groups, distributed geographically over several continents, with responsibilities for individual sub-detectors. Each group needs to be only loosely coupled to others for final detector integration and must preserve their autonomy during the assembly process. Each CRISTAL system is set up to manage the accumulation of potentially Terabytes of physical characteristic data into a detector data warehouse during the construction of a particular CMS sub-detector. The construction follows a specific production plan and each detector is assembled and tested in a step-wise fashion. A distributed object-oriented database is used to hold both the engineering data and the definitions of the detector components and of the tasks which are performed on the components.

An approach has been taken in the CRISTAL design, which promotes self-description and data independence. A multi-layer architecture has been developed to cater for the CRISTAL (meta-)model which facilitates data integration. This allows separation of definitions from instantiations through the use of so-called ‘meta-objects’, promotes object re-use, reduces complexity and facilitates self-description.

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 2 shows the meta-object concept. 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. 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 and the flexibility required to cope with ad-hoc activity specification.

In designing the CRISTAL meta-model, UML has been followed; the result being a detailed model, presented elsewhere [3]. This model describes relationships, types, inheritance, containment and other associations between the meta objects in the system. The meta-objects in the model are definitions, for example, part definitions or activity definitions and the definitions are either elementary or composite in nature. The CRISTAL model is rich in semantics and, consequently, could be applied to general aggregation-based data management systems.

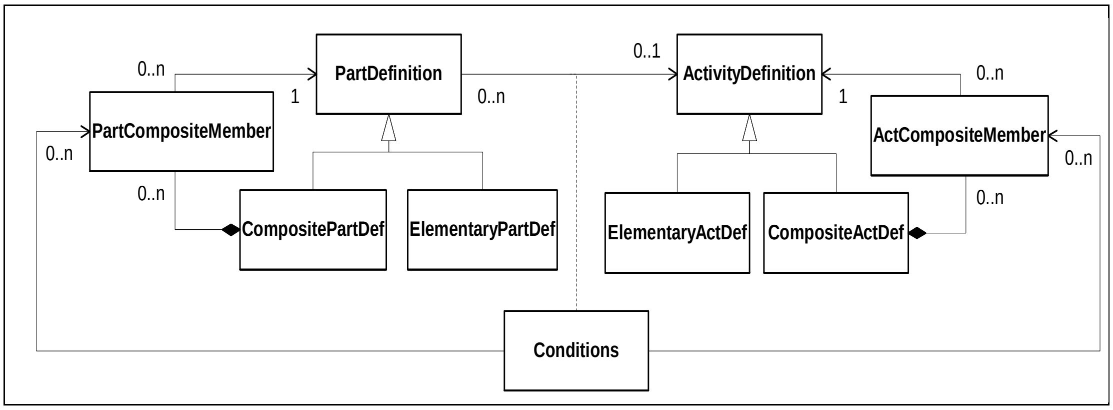

Figure 2: Subset of the CRISTAL meta-model.

Figure 2 shows that there is an association between a given activity meta-object definition and a named part meta-object definition and that this association carries semantics. The left side of Figure 2 captures the product aspects of the meta-model and the right side the process aspects. The CRISTAL data model has been designed so that each assignment of a Part Definition to an Activity Definition is declared for a specific purpose. In detector construction, the assignment is made to indicate the activity to be instantiated for the assembly of a particular instance of a part of a given part definition. Each assignment has associated with it some conditions: in detector construction, the data model captures the definition of the conditions required for each assignment of an activity definition to a part definition. This technique can be generalised for other applications. For example, the association of a maintenance activity to a part will require quite different conditions to be captured than when the detector was constructed. Also, the association of a calibration activity to a part would require calibration-specific conditions to be captured. In other words, the identified association between the process and part description worlds carries rich semantic.

3 Patterns for Description-Driven Systems

In implementing the prototype ontology, it was decided to utilise the UML meta-model. Using it to build a model of object-orientation as the meta-meta-model of the system, it is possible to integrate different application domains and to construct a global meta-meta-data repository. This repository is a collection of all the domain meta-models and it should experience monotonic growth as the only way of change, since meta-models undergo careful analysis and design phases. This incorporation of a meta-meta-model into the system allows use of the meta-model as the abstract medium of communication for the agents, and as the execution of the aforementioned meta-query facility.

In building an ontology, meta-models of different application domains have been used to enable communication between software agents. However, having access to meta-models alone at the highest level of abstraction is insufficient for applications accessing the ontology. In addition the model of the data (i.e. its schema) must be available for applications to query. Then the meta-model describes an abstraction of all the objects of the system whereas the model specifies how instances of the objects, specific to a domain, are specified and together they provide the knowledge required by agents about the system and how it can be accessed. Consequently, not only are both meta- model and model layers required by agents in an ontology, but these two layers must be tightly coupled.

Earlier research has shown that there are certain distinguished design patterns that recur in many different application domains (e.g. those of PDM and WfM discussed earlier) and that should also

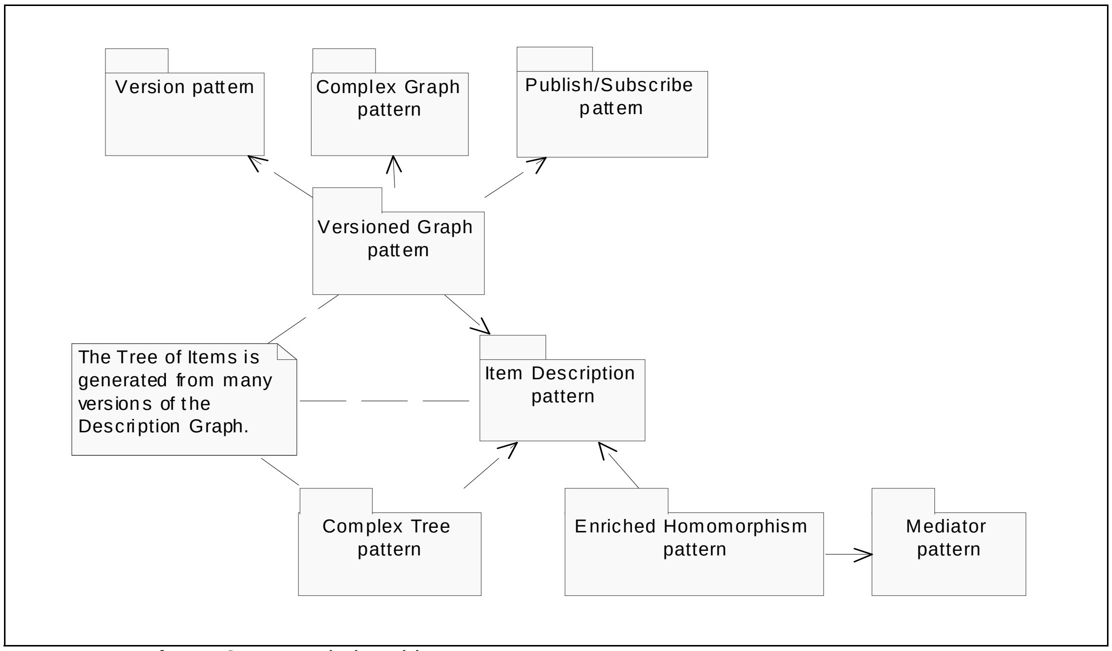
Version pattern
Complex Graph pattern
Publish/Subscribe pattern
Versioned Graph pattern
The Tree of Items is generated from many versions of the Description Graph.
Item Description pattern
Complex Tree pattern
Enriched Homomorphism pattern
Mediator pattern

Figure 3: Description-driven system pattern summary.

be formally specified in the ontology. It is easy to model these patterns in an ontological formalism since they are architectural patterns, identified by their structure and relationships. In Figure 1 the ItemDescription pattern was identified as being central to the construction of a description-driven system. It is through use of the ItemDescription pattern that tight coupling is achieved between the meta-model and model layers of the ontology.

Another important element of description-driven system is the homomorphism pattern which describes how two ItemDescription patterns are related. As a consequence of using the ItemDescription pattern semantics in the homomorphism pattern, and the fact that semantics (conditions) have been added to the association between item descriptions, there will necessarily be semantics attached to the association of one item class to another. However, the constraints that result from the use of the homomorphism pattern cannot be modeled with UML but need to be part of the ontology. One way of representing these constraints is through the expressive power of propositional logic. As an example, consider an acyclic graph pattern appearing as part of the meta-model of a specific domain. That acyclic graph pattern will translate into a tree pattern at the model layer via the ItemDescription pattern. Using UML it is not possible to express the fact that a node in an instance of the graph pattern (at the meta-model layer) cannot recursively appear. As a consequence it is also not possible to preclude this in the instantiated tree at the model layer. In this case, on top of modeling these patterns in an ontological formalism, it is necessary to cater for the constraints between meta-model and model layers through a mechanism such as propositional logic.

One aspect that is required by description-driven systems and which must appear in an ontology, which has been abstracted from a description-driven system, is that of versioning. As stated earlier, versioning between multiple layers in a description-driven system must be asynchronous. However, versioning itself is not part of object-oriented languages and therefore propositional logic must be used to supplement UML in order to handle the constraints that emerge from the use of a versioning pattern [2]. Since the ontology has been abstracted from a description-driven system, it must cater for the set of design patterns that underpin multi-layer systems, including homomorphism, versioning, complex graph, complex tree patterns etc. Figure 3 shows a summary of the design patterns that emerge from a study of description-driven systems. The dependencies between patterns are shown as arrows - for example the versioned graph pattern uses the version, complex graph and publish/subscribe pattern - and there are constraints between these patterns (e.g. between the versioned graph, complex tree and item description patterns). These constraints must be represented in and satisfied by the ontological representation, which, as stated above requires additional semantics to that provided by object-orientation.

4 Conclusions

In the CRISTAL project 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 object models are described using UML which itself can be described by the OMG Meta Object Facility [4] and is the candidate choice by OMG for describing all business models. The overall effect is to produce an integrated set of cooperating databases accessed through a meta-query facility.

Work in the area of design patterns [2] is directly relevant to the ideas expounded in this paper. Foote and Yoder [5] 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. Design patterns are thus needed in object-oriented design to describe meta-schemae such as CRISTAL meta-objects. Similar conclusions are being drawn by Riehle & Gross [6] 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.

In conclusion, This paper has proposed that an ontology, which is rich enough to support multi-domain access from multi-purpose agents, can be abstracted from a description-driven system design. While UML provides sufficient expressive power to model software, certain restrictions, such as its inability to express formally constraints or functions, are serious impediments for the modeling of sharable knowledge. UML as a language is therefore insufficient to provide sufficient semantics for agents to be able to access knowledge in the ontology and mechanisms such as propositional logic must be used alongside UML. Together UML and propositional logic provide the expressive power required for agents to exploit the ontology. Such an approach could reasonably be applied to organisations developing technologies for 'virtual enterprises' (e.g. [7]) where collections of autonomous databases could be related via a central enterprise meta-model.

5 References

1 Kovacs, Z., "The Integration of Product Data with Workflow Management Systems through a Common Data Model". PhD thesis, University of the West of England, 1999.

2 Gamma, E., Helm, R., Johnson, R., and Vlissides, J., "Design Patterns - Elements of Reusable Object-Oriented Software". Addison-Wesley Longman Publishers, 1995.

3 Baker, N. et al., "An Object Model for Product and Workflow Data Management". *Proc. Workshop at the 9th Int. Conference on Database & Expert System Applications*. Vienna, Austria August 1998.

4 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 .

5 Foote, B., and Yoder, J., "Metadata and Active Object-Models". *Proc. of the Int. Conference on Pattern Languages Of Programs,* Monticello, Illinois, USA, August 1998.

6 Riehle, D., and Gross, T.,"Role Model Based Framework Design and Integration". *Proc of the 1998 Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'98),* pp.117-133. ACM Press, 1998.

7 Hardwick, M., Spooner, D., Rando, T., and Morris, K., "Sharing Manufacturing Information in Virtual Enterprises", *Communications of the ACM* **39**(2):46-54, 1996.