Meta Object Facilities and their Role in Distributed Information Management Systems

N. L. Baker & J-M Le Goff
The Centre for Complex Co-operative Systems Faculty of Computer Studies & Mathematics
University of the West of England Bristol BS16 1QY
Email: Nigel.Baker@csm.uwe.ac.uk
ECP Division, CERN, Geneva, 1211 Switzerland
Email: Jean-Marie.Le.Goff@cern.ch

1 Introduction

The rapid convergence of the communications and information systems industries has motivated the movement towards the decentralisation of computer systems and computer applications. Organisations, people and information are naturally distributed and as such the global market demands and users expect distributed computer systems to be integrated and interoperable. Further it is expected that these systems be adaptable, available and can evolve to meet new demands. The fundamental objective of any distributed application or information system is for the separate components to co-operate and co-ordinate computation in order to do useful work and or achieve some overall common system goal. However as systems integrate and grow so does complexity and the ease with which information can be found and managed. Although some progress has been made using distributed object based technology towards reducing the complexity of systems interaction and towards making systems more interoperable, there is still a lot of issues to be resolved.

This paper opens by discussing what is meant by the term *interoperability*, the motivations for building such systems and the characteristics and issues surrounding self describing reflective information systems. A brief summary of current standardisation work follows together with an analysis of how this might impact on the future design of HEP production management systems. The paper concludes with a section on the experiences gained in applying these ideas to a production management system for an experiment detector.

2 Interoperable systems

A fundamental requirement to make any two distributed systems interoperate is that their software components be able to communicate and exchange data. Remote Procedure Calls and their associated data exchange standards is an example mechanism which allows software components to exchange simple data types. In distributed object based systems object request brokers such as the Object Management Groups (OMG) CORBA[1] fulfil a similar role but provide in addition location and access services. However a stronger aspect of interoperability is that distributed systems and components to be integrated, should have common ways of handling and dealing with such system-wide things as events, security, systems management, transactions and faults. Software components must be able to plug into these common distributed services and facilities. Java Beans [2] and Microsoft's ActiveX [3] are examples of efforts to standardise on the way components handle common events and services particularly with regard to user interface and desktop objects. The OMG's CORBAServices [4] is another example of such a standard but this is directed more towards providing fundamental common services required of any distributed application. The CORBAServices specifies how distributed objects should participate and provide services such as naming, persistent storage, lifecycle, transaction, relationship and query. Most major progress in building interoperable distributed systems has been achieved through making use of object oriented techniques. Encapsulation has been used to partition, manage and hide the complexity of the underlying fundamental services and inheritance the mechanism used by application developers to gain access to them.

A further aspect of interoperability concerns ways of making components and systems self describing. That is we want our systems to be able to retain knowledge about their dynamic structure and 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 explosion.

As an example of a large scale system, consider the engineering data management and production management systems required to support the manufacturing of parts for LHC experiments. A huge quantity of data will accumulate in the product breakdown and assemble breakdown structures. As the HEP experiment production system process evolves so more data and the relationships between different aspects of the data must be permanently recorded. HEP groups, projects and systems will require flexible ways to find, access and share this production data. The actual information required will depend very much on the viewpoint and the role of the user in the organisation. The production system for example must provide support for the "as built " view of the manufacturing process and production data. Future HEP user groups may well require a calibration, maintenance or an experiment systems management viewpoint. Also, over time, new distributed computing systems will need to interoperate with the older production and manufacturing systems perhaps in unforseeable ways. There is no doubt that as the manufacturing process gets underway production schemes and part definitions will change. To cope with this a production management system must be able to support dynamic self reconfiguration. One possible way of achieving this is for the system to make

available to itself a self representation for manipulation. A system which can make modifications to itself by virtue of its own computation is called a reflective system[5]. Computation done with this part of the system does not contribute directly towards the goals of the application domain but to the internal organisation of the system. It also serves the purpose of both self and external systems being able to reason about the system. Therefore in order to inter-operate in an environment of future systems and users and in order to adapt to reconfigurations and versions of itself large scale systems must be self describing. Self describing information is termed meta-information or meta-data.

3 Metadata and metamodels

The concepts of metadata and reflection is not new and use of these ideas have been made in frame based languages ([6], [7]). The Actors language [8] describes its own structures and makes use of reflection. Meta classes, that is a class that is used to describe another class was introduced in Smalltalk-80 [9]. CommonLoops and CLOS also provide greater abstraction power by making use of meta class. Another historical application of the use of metadata is in database management systems where a schema provides a representation of the structure, constraints and use of data within the database.

Because support mechanisms for metadata have to fairly recently been largely adhoc attempting to meet specific application needs, the terminology can be a little confusing. The following is an aid to clarify some of the terms:-

- Meta - this prefix signifies " something that describes".
- Meta Information - information that describes information
- Meta-meta-information - information that describes meta-information

In database terminology the term schema is used to mean a formal description of the structure of some collection of data. Metadata then refers to the reified representation of this descriptive data. However in object oriented systems meta-information is usually represented using meta-objects. In general meta-information requires the descriptive power of class, attributes, containment and relationships The object types used must define a language which has the expressive power to describe a model such as object oriented analysis and design diagrams or an SQL database schema. The language (set of object types) required to adequately model the information in the application domain of interest will in most cases be different. What is required for universal interoperability 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 metamodal. This language will be describing information that describes information, in other words meta-meta-information. In this manner it is possible to have an arbitrary number of metamodel layers. The generally accepted conceptual framework for metamodeling is based on an architecture with four layers. Table 1 illustrates the four layer Metamodeling architecture adopted by the OMG and based on the ISO 11179 standard.

LAYER	DESCRIPTION
meta-metamodel MOF layer	architecture infrastructure defines language for specifying a metamodel
metamodal UML layer	an instance of a meta-metamodel defines language for specifying a model
model user object model	an instance of a metamodel a language to define an information domain
user objects	an instance of a model

Table 1 : The Four Layer Metamodeling Architecture

The meta - metamodeling layer is the layer responsible for defining a general modelling language for specifying meta models. Examples of meta-meta objects in this meta-metamodeling layer are MetaClass, MetaAttribute and MetaOperation. This top layer is the most abstract and must have the capability of modelling any metamodel. At the next layer down a metamodel is an instance of a meta-metamodel. It is the responsibility of this layer to define a language, which is itself defined in terms of the meta-metaobjects of the meta-metamodeling layer above, for specifying models. Examples of objects at this layer are Class, Attribute, Operation, Component.

An example metamodel and associated language familiar to most software engineers is the Unified Modelling Language (UML) [10] which is used in object oriented analysis and design. A model at layer two is an instance of a meta model. The primary responsibility of the model layer is define a language that describes a particular information domain. So example objects for the manufacturing domain would be Part, Measurement, Production Schedule, Composite Part. At the lowest level user objects are an instance of a model and describe a specific information and application domain.

4 OMG metamodeling facilities

The purpose of the OMG Meta Object Facility (MOF)[11] 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 metaobjects defined in the MOF will provide a general modelling language capable of specifying a diverse range of metamodels although the initial focus was on specifying metamodals in the Object Oriented Analysis and Design domain. It has been designed to support:-

- *Generality*: it should be capable of describing a range of metamodels.
- *Extendibility*: it is a core model and is capable of extension by inheritance and composition
- *Reuse*: when developing metadata for a new application it should be possible to reuse metadata from other similar applications
- *Reflection*: it should be capable of being able to represent itself

The usage of the MOF will depend very much on viewpoint. From a systems designers 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. The definition is then used to drive subsequent software design and implementation. 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 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. A repository service called the Repository Common Facility[12] is the architectural component that is used to make this metadata and metamodels available in a run time environments.

Figure 1 shows the positioning of the Repository Common Facility within the OMG Architecture. A repository within the context of distributed object based systems is a framework for developing, integrating, deploying and managing independently developed reusable software components. This framework is usually implemented on one or more database systems and uses database management systems, persistent storage and transaction services. Repositories that currently exist tend to support specific domains such as Software Development or Business Information.

One fundamental repository which is already used in the OMG Architecture is the Interface Repository (IR) which supports the Object request Broker (ORB). The IR contains the interface specifications of all the objects that an ORB recognises. The IR is queryable and updateable at runtime with all the objects being described using CORBA IDL (Interface Definition Language). Proposals for the Common Repository Service has been defined to include the IR and considerably extend repository facilities to satisfy enterprise

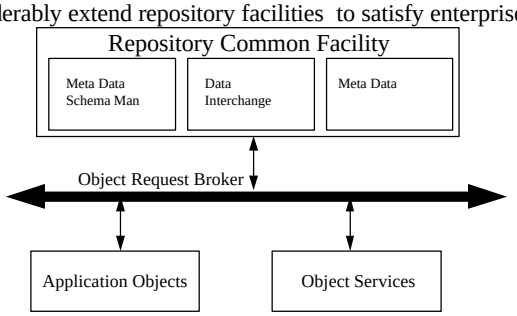

Figure 1. Repository Common Facility within OMA

wide needs. An outline architecture[12] for a Common Repository Facility (CRF) proposed by Unisys to the OMG is shown in figure 2. The Repository Object Model shown as the top layer of figure 2 describes, using the MOF, fundamental repository concepts such as models, classes, relationships and is used to define the CRF itself as well as other repository models

In other words this is the repository meta-metamodel. The layer underneath is the representation of the repository common facility itself and is a composition of key object services and common facilities. This layer is often called the repository meta model. The final set of layers corresponds to the content models, that is they describe the meta models that are contained in the repository. Example models are:- Business models, manufacturing models, workflow models and tool models.

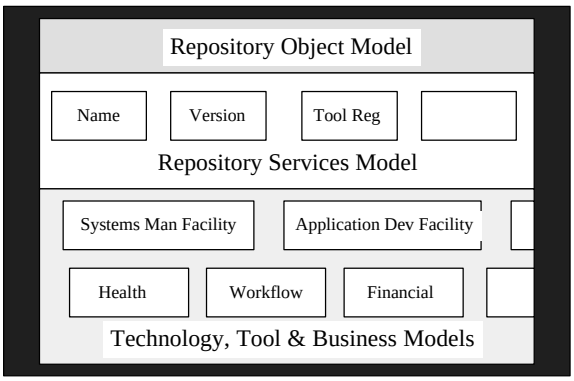

Figure 2. Repository Information Model Architecture

Figure 3 shows an example of a repository architecture and how various types of applications can access repository object instances. A repository object will be an OMG compliant object defined in IDL whose type is defined in a repository meta model or object schema. It will be accessible by repository services such as the Repository Naming service and versioned by the Repository Change Management service. Users must be able to create, manage, extend and evolve the definition of metamodels (object schema's) in order to reflect the requirements of their application or tool. The repository will manage the lifecycle of all repository

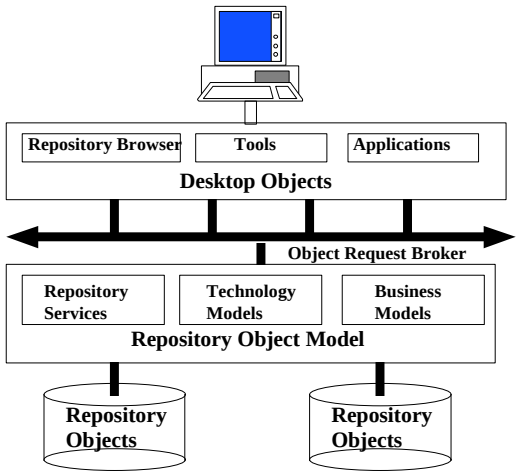

Figure 3. Run Time Distributed Object Repository Architecture

objects and provide a default object factory for defined object types contained in the repository. Other examples of services supported include:- query service, security service, externalisation service, relationship service, event notification service, transaction service.

5 Workflow management

A workflow management system allows managers to co-ordinate and schedule organisational activities in order to optimise the flow of information between system resources. Workflows are collections of human and machine based activities (or tasks) that must be co-ordinated in order to accomplish some business process. Commercial examples of workflow management systems exist which can facilitate the definition, management and execution of workflows whose order of execution is driven by computer representations of the workflow logic..

CRISTAL (Co-operating Repositories and Information System for Tracking Assembly Lifecycles)[13] is a workflow management system that supports production and manufacturing of parts for LHC experiments. The initial phase of the CRISTAL project is concerned with the management of production, testing, assembly and tracking of over 110,000 lead tungstate mono-crystals and their fast electronics to be installed in the CMS Electromagnetic Calorimeter (ECAL) high energy physics experiment. The production and assembly of these parts estimated to be about 1 million in total, is distributed around a number of sites in Europe and Asia.

A Product Data Management (PDM) system holds the descriptions of the Product Breakdown Structure (PBS), the Assembly Breakdown Structure (ABS) and the Work Breakdown Structure(WBS). The main components of CRI-STAL is a distributed workflow enactment service and a workflow application programming interface. The application programming interface is required to define and modify workflows (production schemes) and activities (tasks). The initial

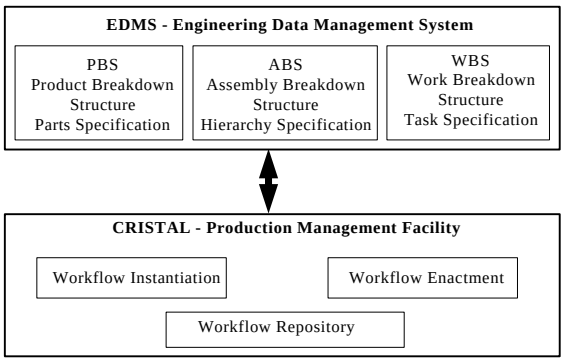

Figure 4. The CRISTAL Environment

workflow definitions are derived from the WBS and ABS parts of the PDM. The workflow enactment service consists of an execution interface and an execution service provided by a so called work engine. The engine is the component that executes the static workflow production descriptions. The monitoring, managing and other run time services associated with the executing workflow instances is effected through the execution interface. The relationship between CRISTAL and PDM is illustrated in figure 4.

6 Object modelling in CRISTAL

Workflows within CRISTAL are distributed objects and therefore there is no need for a centralised work flow engine. Whenever a real physical part or composite part is referenced within a production or assembly centre its corresponding workflow instance (workflow engine) object co-ordinates with the operator through the desktop control panel as to the next possible activities to be performed on the part. So in the object model each part definition object is associated with a workflow definition object. The workflow definition objects are called workflow schema objects in workflow management terminology. Workflow activities (tasks) are objects that describe the activity (human or machine related) that must be performed on the part. These definition objects are meta objects since they are objects that describe.

The workflow meta model of CRISTAL is described using UML, a skeleton outline of which is shown in figure 5. This

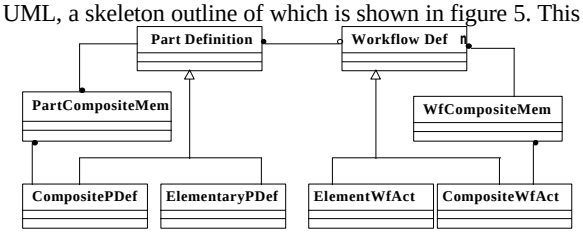

Figure 5. UML Object Model

model describes relationships, types, inheritance, containment and other associations between the metaobjects in the system. However in order to support creation of part and task definitions and to cope with dynamic change a run time representation of this logical model must be available for manipulation. The designers of CRISTAL are making extensive use of the OMG CORBAservices [14] to create a run time workflow model which is stored in a repository. This repository model is an example of one of the Business & Technology Models in the Repository Information Model Architecture shown in figure 3.

The CORBA Relationship Service provides standard interface specifications that describe the behaviour of run time objects that allow modelling of:-

- *Types of Relationship*
- *Relationship Roles*
- *Degree of Relationship*: the number of roles required in a relationship of a given type
- *Cardinality of Relationship*: number of relationships of a given type that can be associated with a given role
- *Attributes*

The implementation of the relationship service allows objects to be related independently of their IDL interfaces which means that relationships can be modified at run time as the workflow metamodel changes. To support model changes requires creation, deletion, move and copy of repository objects. For example creation of the self- model, relationship changes and workflow definitions changes all require support for the creation and destruction of objects. New versions may require move and copy services. All of these object services are specified in the CORBA LifeCycle Service interfaces. To create objects in a distributed system requires Factory

Objects. Although a GenericFactory is provided by Lifecycle Services many factories are required to support creation of different object types. Lifecycle Service object creation and deletion also supports a reference to the Naming Service so that when an object is deleted its associated name is also removed. Other useful CORBAservices that may be used in the future are Query, Event, Transaction and Property services.

Figure 6 illustrates how the CRISTAL architecture relates

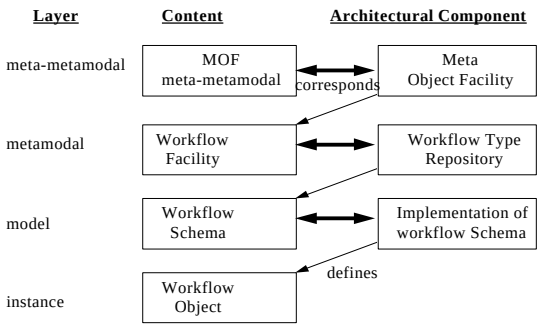

Figure 6. Workflow Facility Metamodal Architecture

to the 4 layer meta data model. The proposal for this type of architecture was first advocated by Schultz[15] at an OMG technical meeting. The MOF is capable of describing all the types used in the CRISTAL workflow metamodel. There are a number of proposals for a standard Workflow Facility currently under consideration by the OMG.

A key role of the CRISTAL production workflow system is to interoperate with PDM systems which describe the parts to be manufactured. Within the Manufacturing Technology Domain of the OMG there is already considerable progress in specifying a PDM metamodel[16] which will be capable of providing a range of PDM object models to support various manufacturing industries. Using this metamodel approach will allow changes or versions of the core metamodel to give the "as built view" object model or "as maintained view" object model or "as planned view" object model and other object models to support the manufacturing business. These object models will be stored in a repository and will interoperate with other business systems like the workflow system of CRISTAL. So that when changes occur in the product design these changes can be propagated to the production workflow system. Similarly changes to manufacturing procedures in the workflow model can be propagated back to the PDM "as built model". There are number of "product data management enablers" proposals under currently under consideration by the OMG most of which are based on Standard for the Exchange of Product model data (STEP). In EDMS (the PDM that CRISTAL must interoperate with) the proposal is to use a commercial product to support all the PDM functions.

Conclusions

Our experiences of using metamodels and metadata at the analysis and design phase in the CRISTAL project have been positive. By modelling the workflow metamodel separately from the workflow enactment runtime model it has allowed the design team to provide consistent solutions to dynamic change and versioning. Further by being able to manipulate and reconfigure the workflow meta-tmodel it will be possible to produce runtime work-flow systems for a range of detectors and scientific systems. The object models are described using UML which itself can be described by the OMG MOF and more importantly is the candidate choice by OMG for describing all business models. Work is currently underway to implement the metamodel using object interfaces defined in the OMG CORBAservices and using an object oriented database to support the construction of the workflow facility repository.

References

[1] Object Management Group. See http://www.omg.org

[2] See http://www.javasoft.com

[3] See http://www.microsoft.com

[4] Object Management Group Publications. CORBA-services: Common Object Services Specification

[5] Maes, P. 1987 "Concepts and Experiments in Computational Reflection". ACM OOPSLA 1987 Proceedings. pp 147- 155

[6] Minsky M. 1974 " A Framework for Representing Knowledge". MIT AI-MEMO 306 Cambridge, Mass.

[7] Roberts R. & Goldstein I 1977 "the FRL Primer" MIT Laboratory for Computer Science. Technical Report 272 Cambridge, Massachusetts

[8] Lieberman H. 1981 " A Preview of ACT1". MIT AI-MEMO 625 Cambridge, Massachusetts

[9] Goldberg A. & Robson D. 1983 "Smalltalk -80: The Language and its Implementation". Addison - Wesley

[10] UML See http://www.rational.com

[11] OMG Meta Object Facility Joint Revised Sub-mission Sept 1977 OMG Document ad/97-08-14

[12] OMG Unisys Response to the OMG Repository Common Facility RFI November 1995

[13] J-M Le Goff et al., "CRISTAL - Co-operating Repositories and an Information System for Tracking Assembly Lifcycles" CERN CMS Note 1996/003

[14] OMG CORBAservices Manual

[15] OMG Publications. Technical document cf/97-05-08

[16] OMG Publications Product Data Management Enablers RFP, Manufacturing Domain Task Force Doc. mfg/96-08-01