Meta Data Interchange Patterns

Abstract. In the model-based approach to meta data interchange, instances of formal domain models are exchanged between heterogeneous software tools. Achieving reliable model interchange requires an agreed-upon modeling language and interchange format. It also requires that software tools understand the *intent* of each interchange event. However, describing the intent of model interchange is generally beyond the scope of what most modeling languages are capable of expressing, so it must be accomplished via some other means. This paper presents *meta data interchange patterns* as an effective mechanism for specifying model interchange semantics. The rationale for using pattern techniques to solve meta data integration problems is discussed in depth, as are practical methods for defining and publishing meta data interchange patterns. The current effort within the Object Management Group to formalize the specification of meta data interchange patterns within the context of Model-Driven Architecture is also presented, along with an overview of applying these methods to the creation of a pattern-based, meta data repository.

Introduction

Meta data[1] sharing is a key prerequisite to integrating heterogeneous software tools, products, and applications. Sharing meta data effectively requires both an agreed-upon language for describing meta data, and a standard interchange format or interface for representing shared models. In recent times, the use of formal modeling languages, such as the Unified Modeling Language (UML) [15] and Common Warehouse Metamodel (CWM) [10, 18, 19, 21], and their mappings to various implementation technologies, such as XML Metadata Interchange (XMI) [11] and Java™ Metadata Interface (JMI) [5], have been proposed as solutions for model interchange.

Effective model interchange, however, also requires that producers and consumers understand the *intent* (or *meaning*, or *intended effect*) of each interchange event, and that shared models are formulated in a manner consistent with that intent. This issue first emerged (at least within the author's realm of experience) during initial validation and testing of CWM, when it was discovered that successful interchange between CWM-enabled software tools required a considerable amount of up-front agreement between participants on the overall structure and content of exchanged XMI files. It was apparent that this had nothing to do with the effectiveness of CWM itself, but rather, that there were certain *general forms* of models that were more effective than others when it came to expressing the meaning of specific kinds of interchange scenarios. Participants found that they could meaningfully exchange any models consistent with a small number of

1 By "meta data", we mean a broad category of formal models, including models of information structures, business processes, and application systems. Throughout this paper, the terms "meta data" and "model" are used interchangeably.

agreed-upon, general forms, even though individual models differed from one another considerably in their details.

It was subsequently reasoned that such model variation could perhaps best be described in terms of standard patterns of model composition. Such a pattern-based approach would allow for the exchange of crisply defined, predictable structures that, nonetheless, exhibited a wide range of useful variability. This culminated in the concept of a *meta data interchange pattern*, and initial formulations of the theory and practice of such patterns have been described in a text book on CWM [19], and formalized in ongoing work within the Object Management Group (OMG) [12]. These early formulations of meta data interchange patterns leveraged some of the well-known practices of the software design patterns community [3, 4], but also introduced new pattern theories, methods, and artifacts.

Purpose of this Paper

This paper provides a comprehensive introduction to the theory and practice of meta data interchange patterns. The reader is shown, through a simple case study, how meta data interchange patterns can be used to achieve precise, model-based interoperability between heterogeneous software tools. Much of the underlying theory of meta data interchange patterns is also presented, as part of the explication of the case study.

Techniques and artifacts for specifying meta data interchange patterns are also presented, with emphasis on the automated publishing and management of patterns based on the use of pattern repositories and software-consumable pattern specifications. This includes an overview of the OMG's recent work in this area [12].

It should be noted that, although this treatment of meta data interchange patterns is strongly grounded in CWM, the use of patterns in model-based interoperability is certainly not bound to CWM, nor any other formal language or metamodel. The pattern theory described in this paper has broader application within the domain of model-driven systems, including the OMG's Model-Driven Architecture (MDA) initiative [8, 14]. Furthermore, the pattern-oriented modeling practices described here easily transfer to other modeling paradigms, such as entity-relationship and business process modeling.

The next section provides an overview of CWM and describes how CWM is used to interchange shared meta data. This is followed by a detailed investigation of why model-based interoperability in heterogeneous environments is difficult to achieve, and how pattern application can help to overcome much of this difficulty. Meta data interchange patterns are then given formal definition, and the benefits of using patterns in the construction of interoperable tools are briefly enumerated. Pattern specification techniques, including those supporting the automated publishing and management of

online pattern catalogs and pattern-based meta data repositories, are described next. Finally, a summary of results is provided, along with directions for future research.

An Overview of CWM

CWM [10, 18, 19, 21] is a generic domain model of data warehousing and business intelligence concepts that serves as a common model of meta data (a *metamodel*) for the various software tools, products, and applications comprising a data warehouse, information supply chain, corporate information factory, or other similar approaches to realizing business intelligence solutions. Instances of CWM (*models*) represent shared meta data, and may be interchanged between tools and repositories via XML documents that conform to the OMG's XMI standard [11].

CWM is defined in UML notation, and effectively extends UML by defining new modeling classes representing data warehousing and business intelligence domain concepts. Thus, CWM is a platform-independent modeling language for describing data warehouse schemas, in the same manner that an entity-relationship (ER) language is used to design relational databases, or UML is used to design object-oriented software systems. CWM is an instance of the OMG's Meta Object Facility (MOF), which is just another formal language used to describe object-oriented modeling languages [13] (UML is also an instance of the MOF). Fig. 1 summarizes the layered architecture of CWM, which is comprised of a number of sub-metamodels (or *packages*) representing different aspects of a typical data warehouse: Relational database, online analytical processing (OLAP), data transformations, data warehouse management activities, and so on.

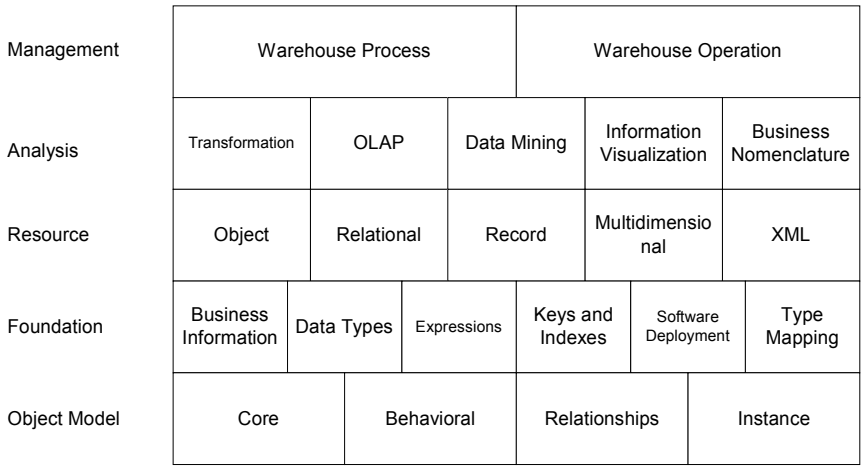

Fig. 1: CWM layered architecture

A data warehouse architect can design a complete data warehouse by piecing together those CWM modeling elements that are relevant to various modeling sub-problems. For example, to design the extract, transformation, and load (ETL) process that builds the data warehouse, the architect might use CWM's Relational and Record packages to specify the raw data sources and operational data store (ODS), the Relational and OLAP packages to design the dimensional analysis store, and the Transformation package to design source-to-target data mappings and conversions. The complete model of the ETL process is a combination of modeling elements from each of these packages[2]. Figure 2 illustrates a CWM instance representing a high-level model of an ETL process.

The data warehouse architect might construct her model using a CWM-aware visual modeling tool, or perhaps a graphical modeling tool with a more user-friendly symbolic notation that is, nonetheless, back-ended by a CWM-aware export utility or model repository. The CWM instance is exported in the form of an XMI document that is subsequently consumed by a CWM-aware ETL tool. The ETL tool might use the model to initialize its internal meta data, or, alternatively, traverse the model in an interpretive manner, as part of the process of managing its operations[3]. But regardless of how the model is actually used by the consumer, the key point here is that the ETL process has been defined in terms of a technology-independent representation that is external to any particular software process or tool. The ETL process model is readily usable by any CWM-aware tool requiring it. Fig. 3 below illustrates a detailed model of this mapping.

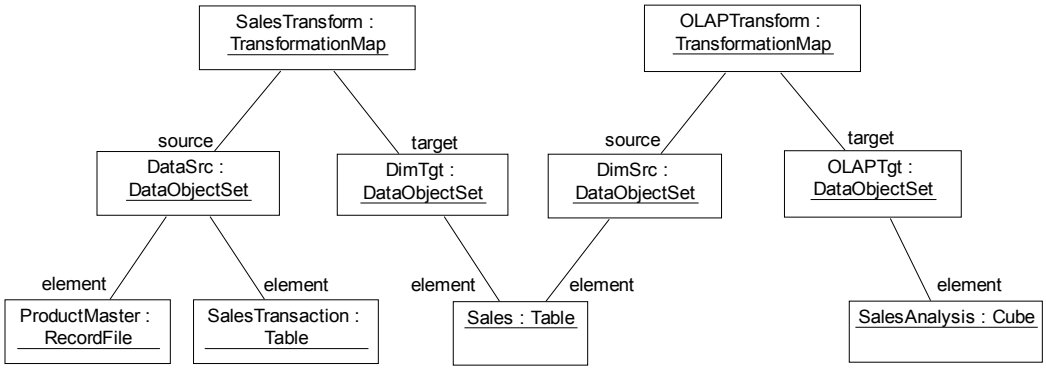

Fig. 2: High-Level Model of an ETL Transformation Mapping

2 Any CWM model can be thought of in terms of a trajectory through some subset of the packages comprising the layered architecture of Figure 1.

3 A software tool that uses MOF directly as its implementation model is capable of understanding languages like CWM and UML, without necessarily having to be coded to do so. A software tool designed to interpret shared models directly (whether MOF-based or not) is fundamentally capable of totally dynamic operation and evolution over time. [16, 17].

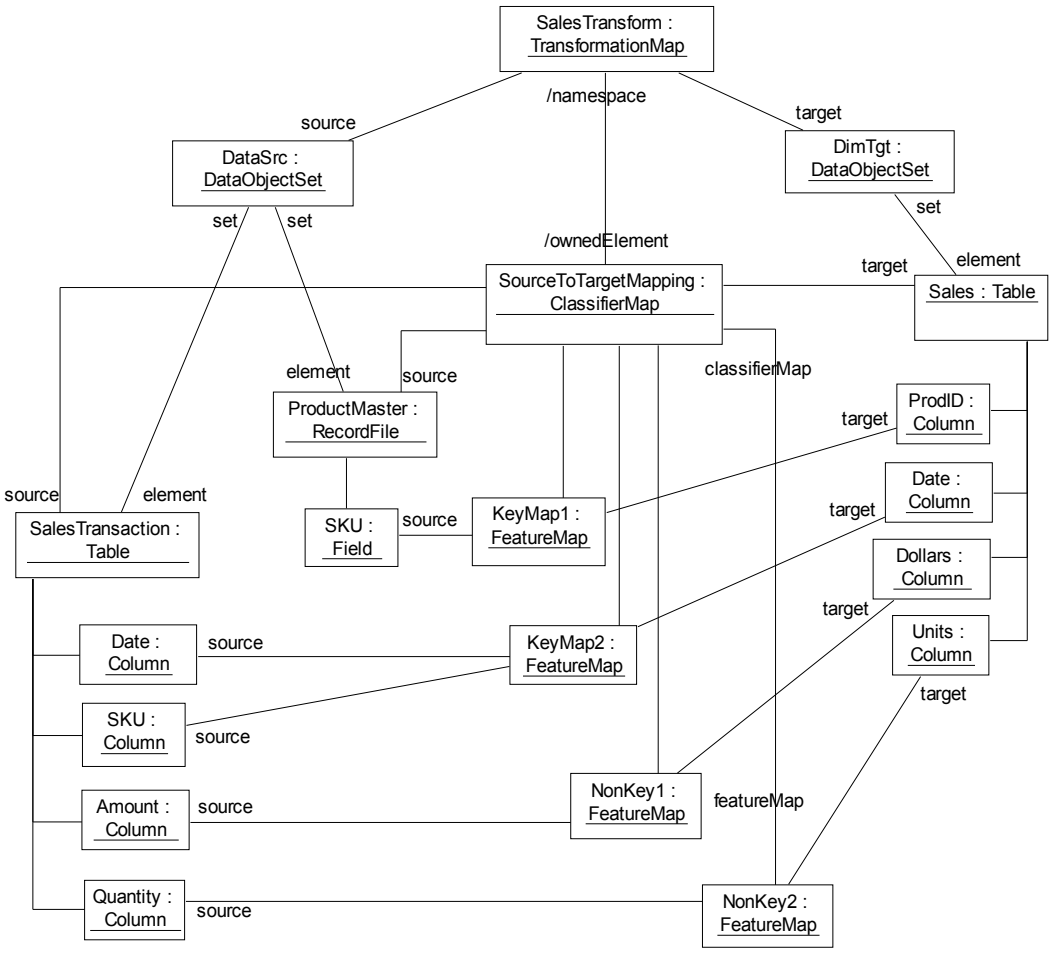

Fig. 3: Detailed Source-Target ETL Transformation Mapping using CWM

The Fundamental Problem of Model-Based Interoperability

Although the ETL tool example of the preceding section would suggest that a meta data interchange strategy like CWM is straightforward and effective, there is a fundamental difficulty associated with model interchange that is not addressed by a common metamodel and interchange format. In any collaborative computing environment comprised of multiple, heterogeneous software tools, there are potentially many meta data producers and consumers, all of varying capabilities, and with very diverse meta data requirements. The fact that a commonly agreed upon language is being used to describe meta data does not guarantee that some particular model will necessarily satisfy the requirements of its intended consumers. Nor is there any assurance that a given model

will actually make sense in terms of what some particular model interchange event is attempting to accomplish.

For example, if a CWM-enabled ETL tool were supplied with a model that consisted of definitions of data resources, but no accompanying transformation mappings, would the ETL tool find such a model useful? Note that there is no inherently right or wrong answer to that question. Much of this depends on the capabilities of the ETL tool itself, and not so much the fact that the tool understands CWM. The tool might reject such a model altogether. On the other hand, the tool might conclude that it was simply dealing with an under-specified model, and that the transformation definitions were forthcoming, in which case, it might go ahead and incorporate the data resource definitions into its internal model.

As a formal language for describing data warehousing and business intelligence meta data, CWM has no means of expressing how its models should be used, nor what makes sense or does not make sense for any particular model interchange event. Clearly, such concerns are beyond the scope of any descriptive formalism No language definition is capable of describing how, or in what ways, it might be used to achieve some result. This is simply outside the realm of possibilities. To do so requires going outside the language. So how do we go about overcoming these difficulties within the context of CWM and model-based interoperability? We can best illustrate the solution by delving into yet another, slightly more detailed, real-world example.

Consider an analysis-oriented consumer, such as an OLAP tool or reporting package, that expects to process meta data describing a relational database organized as a simple *star schema* [6]. The relational model [2] defines no concept of star schema. Rather, this is an *organizational pattern*, or *pattern of usage*, of relational database structures that provides a best-practice solution to a specific problem (i.e., dimensional analysis).

As a consequence, the CWM Relational package also has no inherent notion of any particular strategy for table organization, but rather provides the means whereby a modeler can specify any conceivable form of relational database organization. So, from the viewpoint of an analysis tool that works against relational star schemas, a general description of the general structural form (or simply, *general form*) of any valid star schema model (as an instance of the CWM Relational metamodel) needs to be established. It turns out that such a description can be formulated by answering three simple questions regarding model content:

1. What portions of the CWM metamodel might an imported model represent? In other words, precisely which classes and associations of the CWM metamodel are expected to have corresponding instances in the imported model?

2. What boundaries on the instances of CWM metamodel classes might an imported model be constrained by? In other words, what limits on either the cardinalities or value ranges of instances might reasonably be assumed?

3. At what CWM metamodel element instance should inspection of an imported model begin? In other words, what is the *outer most*, or *root level*, model element at which traversal of a particular model might begin?

In the particular case of an interchange scenario in which our analysis tool is expecting a star schema model, each of the preceding questions might be answered as follows:

1. The model will contain instances of the CWM Relational metamodel classes of Catalog, Schema, Table, Column, Primary Key, Foreign Key, and all concrete, inherited, or derived associations relating these classes.

2. The model will contain precisely one instance of Catalog, one instance of Schema, and at least three instances of Table (representing a *fact table*, and at least two *dimensional lookup tables*, respectively). Furthermore, precisely one Table instance – the one representing the fact table – has a composite primary key whose component columns each have a many-to-one relationship with the primary key column of precisely one of the dimensional-lookup Tables. Each such relationship is realized via an instance of the Foreign Key class.

3. Inspection of the model should begin at the single instance of the Catalog class.

Fig. 4 shows an instance of the CWM Relational metamodel that conforms to the star schema description just formulated. This model consists of a single fact table called *Sales*, and two dimensional look-up tables named *Product* and *Time*, respectively. Two non-key columns of the Sales table, *Dollars* and *Units*, representing dollar amount and quantity sold, respectively, define the measures of each unique sales event.

Assuming that all meta data producers and consumers agree on the preceding description as a general characterization of all possible star schema models, there is no ambiguity regarding which instances of the CWM Relational metamodel represent star schemas, and which represent non-star schema database organizations. The fact that the total number of star schema configurations is potentially unlimited does not prevent us from stating a simple, finite description of them. And this description clearly involves the specification of a *pattern*.

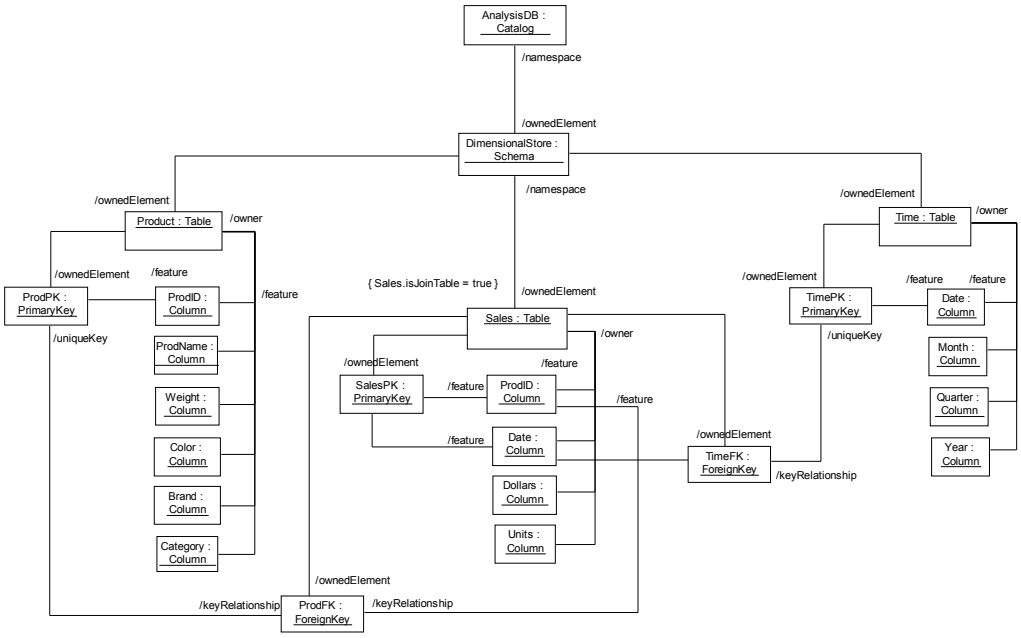

Fig. 4: CWM Model of a Relational Star Schema Database

One particular advantage of having this unambiguous description of a star schema is that it provides any meta data consumer expecting to import star schema models with a completely logical strategy for searching the content of any XMI document:

1. If the XMI content is not a valid rendering of CWM – that is, it was not successfully parsed against the a DTD or XML Schema definition – the XMI document is rejected without any further investigation.

2. Otherwise, the consumer searches for a single instance of Catalog, as the root element of the model. If the XMI document contains no instance of Catalog, or contains more than one instance of Catalog, the consumer concludes that the content does not conform to the expected pattern, and may opt to reject the XMI document.

3. If a single instance of Catalog is indeed discovered, the consumer continues by inspecting the Catalog for a single, contained instance of Schema, which in turn should contain three or more instances of Table, and so on. Ultimately, the consumer can identify the single fact table by inspecting the key relationships between the various tables in the model. If the consumer discovers precisely one Table instance whose composite primary key columns have foreign key relationships to the primary keys of each of the other Tables, then it knows that this is the fact table, and that the model represents a relational star schema.

4. Otherwise, the consumer rejects the model as not representing a valid star schema. That is, the model does not conform to the accepted notion of how a star schema model should be structured.

Of course, the ability to realize such a pattern-based, general description of a star schema model in software and associate it with a particular model interchange event requires considerable formalization of these notions. These formalisms are developed through out the next two sections.

Introducing Meta Data Interchange Patterns

It should be clear from the preceding example that achieving meta data integration using the model-based approach requires not just an agreed-upon formal model, but also the more general notion of a *pattern* imposed on model content. Formal modeling languages like UML and CWM are highly flexible and expressive, and are intended for general-purpose problem solving within broad subject areas. Such formal languages could not achieve true generality, however, if they also attempted to accommodate all conceivable *idiomatic usages* of those languages that are often used to solve problems in very specific domains. Such idiomatic solutions are often expressed in terms of *patterns*, and they represent standard ways of using a more general language to solve recurring problems peculiar to some particular domain (that is, a sub-domain of the broader subject area of the formal language). The dimensional star schema is an example of just such an idiomatic solution to a highly domain-specific problem.

It is the position of this paper that a pattern-based approach to meta data interchange is a highly effective solution to the more general problem of ensuring reliable, model-based integration between software tools. This is due in large part to the fact that the pattern-based approach provides end users of modeling languages (who are the true problem domain experts) with the means to express precisely structured models that still allow for considerable variability in model content[4]. End-users who adopt the pattern-based approach to meta data interoperability can also leverage the pattern codification practices already established by the software design patterns community [4]. This topic is addressed in subsequent sections of this paper.

Definitions of Meta Data Interchange Pattern Concepts

Let's now formally define the fundamental concepts of meta data interchange patterns. We will do so by re-casting the questions regarding model content that were informally posed in the previous section in terms of more formal requirements for successful, model-based interchange and interoperability. Generally, an effective model-based

4 This characteristic is referred to as *variation-oriented design* [3].

interoperability solution based on the use of patterns must account for each of the following:

- A context for interchange
- Boundaries on solution extents
- A starting point for content discovery

We'll now consider each of these general requirements in greater detail, and provide a formal, technical definitions of each.

A Context for Interchange

Context refers to some precisely specified subset of all possible instances of a metamodel that a given interchange event may consist of. Instances occurring within a prescribed context are acceptable to a consumer. Instances occurring outside of that specified context are not acceptable to the consumer. A consumer requiring a particular context has announced, in effect, that it knows how to interpret the overall *intent* of any interchange event based on that context. Context can be viewed as an expression of intent.

Conversely, intent can be viewed as something that implies a particular context.
As described earlier in this paper, the specification of context is necessarily outside the scope of the metamodel itself, in much the same manner that the context of a conversation between two humans is external to the content of a spoken exchange. If two persons establish a context for their conversation, then most sentences uttered by the speaker will make sense to the listener. On the other hand, if context has not been established (for example, as in the case of a third person who happens to walk in on a conversation already in progress), it is highly likely that most statements will not make sense to the listener, even though they are perfectly valid statements from a grammatical point of view.

In the exchange of sentences of a formal language or metamodel, such as CWM, a context needs to have been established between producer and consumer (that is, as part of the manifest semantics of an interchange event) if the interchanged content is to make sense to the consumer. A human designer might have a reasonable understanding of the intent of an interchange event and, from this, infer its related context. But any subjective notion of context needs a precise definition if it is to be intelligible to software. It turns out that any attempt to explicate one's subjective notion of context invariably points to some subset of all potential instances of the metamodel, and this subset in turn points invariably to some projection of the language model itself. Identifying this projection is the first step in specifying a computationally precise notion of context:

Definition. A *Projection* is some portion (for example, sub-graph or cut-set) of the metamodel that is relevant to a particular model interchange event.

The specification of this sub-graph formalizes the structure of the more subjective notion of a context for interchange. The projection of the CWM Relational metamodel that is relevant to the star schema example described earlier in Fig. 4 is shown below in Fig. 5:

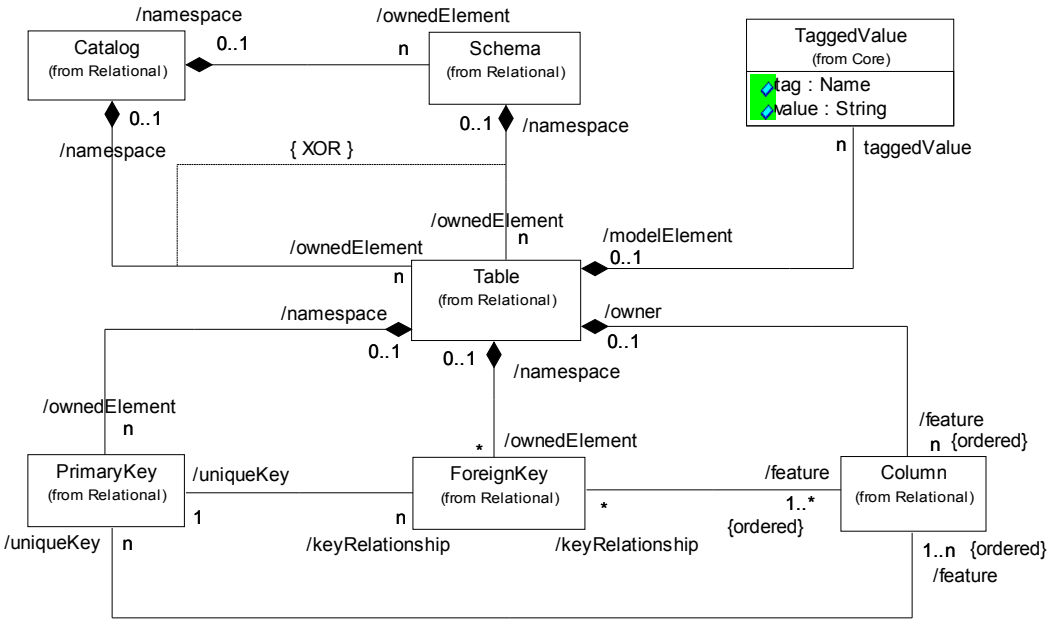

Fig. 5: Projection of CWM Metamodel for Relational Star Schema Model

Boundaries on Solution Extents

An instance of a metamodel potentially consists of an unlimited number of valid arrangements of instances of metamodel elements. This is largely a consequence of the fact that:

- Many association-ends in the definition of the metamodel have unbounded cardinalities.
- Most formal languages and metamodels specify no boundaries on the number of instances of any given class that a model might contain.
- Certain class attributes are of data types with no prescribed boundaries on their value ranges (for example, null-terminated strings).

Of course, the reason for allowing unlimited model extents is clear: There is usually no logical reason for restricting the number of instances of a class or values of an attribute. Modeling languages generally avoid constraining the logical design process by not committing to arbitrary limitations on model extents. The problem with this, however, is that allowing an unlimited multiplicity of instances may lead to the potential for unbounded model complexity, as well as ambiguous interpretations in situations where having only a single, unique instance of some model element would be meaningful. For example, it is obvious that the Relational metamodel projection in Figure 4 contains everything necessary for representing the topology of any relational star schema.

However, it should also be obvious that the extent of this projection consists of a countably infinite number of instances. Every conceivable relational database organization (in terms of catalogs, schemas, tables, columns, and keys) is represented by this projection. And, of course, this includes every conceivable star schema design, as well. In fact, the total collection of all possible star schemas is also a countably infinite set, and yet it constitutes a vanishingly small subset of the total extent of the Relational projection.

Therefore, both producer and consumer clearly need to agree on the imposition of certain limits on the extent of the projection. This also must be part of the definition of any interchange event that the producer and consumer participate in. This leads us to our second definition:

Definition. A *Restriction* is a set of constraints that are imposed on the extent of the metamodel projection for a particular model interchange event.

Restrictions have the effect of limiting the number of metamodel classes, the subclasses that may be substituted for some base class, or the ranges of certain class attribute values, in such a manner that interchange semantics are made more precise, and model complexity is contained.

Quite often, only a very small number of constraints need to be defined. In the star schema example, the only necessary constraints were those related to the multiplicities of Catalog, Schema, and the Tables participating in the role of fact table (precisely one, in the case of our highly simplified star schema example). Note that there is no meaningful upper limit on the number of dimension tables (although at least two dimension tables are required). Similarly, there is no need to constrain the multiplicities of the key relationships. This is handled indirectly by the constraint identifying the single fact table. And there is certainly no reason to restrict things like Table names. The star schema consumer isn't concerned with specific element names, but only with overall model structure.

Thus, there is a distinction between restrictions that clearly influence structure, and those that constrain content in a manner that's relatively independent of overall structure. For example, if we introduced a restriction that required the name of the fact table to always be assigned the value "Sales", we would certainly be constraining content without changing structure. We refer to restrictions that affect content only as *binding parameters* (or simply *parameters*), since they represent values that can readily be varied without affecting the overall semantics of the model, and yet their assigned values are a necessary part of *binding* the pattern definition to a particular *pattern realization*.

A Starting Point for Content Discovery

The final piece of information that must be specified as part of the interchange event definition is precisely where the consumer should start reading the model. The reason for this is that most models based on languages like UML and CWM are not necessarily structured as directed graphs with definite root nodes. A metamodel instance may possibly have several, potential top-level elements, all of which could be valid starting points for traversing the model. This gives us our third definition:

Definition. An *Anchor Element* is any metamodel element of the projection whose instances are to be used as starting points in the inspection or traversal of a model.
In the star schema example, Catalog is designated as the anchor element. Since the restrictions require that there be precisely one instance of Catalog, any star schema model has precisely one anchor point, and its identity is readily discerned in terms of element class.

Given the three previous definitions, we can now provide a formal definition of a meta data interchange pattern:

Definition. A *Meta Data Interchange Pattern* is an identified projection of a metamodel, optionally with one or more restrictions on instances of that projection, and optionally with one or more specified anchor elements.

Benefits of Meta Data Interchange Patterns

A number of significant benefits are realized by the application of patterns to model-based interoperability. The following briefly summarizes those benefits:

- The intended semantics (intent, semantic context, etc.) of any defined interchange event can be precisely specified and published (made known) to the environment, in terms of one or more meta data interchange pattern definitions.

- Shared meta data can be described in terms that domain experts are familiar with. For example, one can talk about publishing "a star schema model" as opposed to

"an instance of the relational metamodel". Meta data patterns allow higher levels of description to be attached to interoperable units of meta data.

- Pattern-aware consumers have a precise strategy for processing the content of an imported model. Models can be infinitely variable in content and yet completely unambiguous in structure. A consumer can readily search any model in a completely intentional manner, with no ambiguity of outcome (i.e., model content either obviously makes sense, in which case it is used, or makes no sense, and is rejected by the consumer). This simplifies implementation logic, lowers implementation costs, and increases the overall reliability of model interchange, specifically in collaborative, heterogeneous, software environments.

- Pattern-aware producers have precise guidelines by which to construct models for export. This has the same implications for the producer as it does for the consumer. It simplifies implementation logic, lowers implementation costs, and increases the overall reliability of model interchange.

Note that producers and consumers in pattern-oriented environments that disregard established interchange patterns obviously run the risk of not always being able to understand one another. Furthermore, producers that do not follow prescribed patterns when constructing models may publish meta data that is not readily interchangeable with any pattern-aware tools.

Specifying Meta Data Interchange Patterns

Now that we've formally defined the concept of a meta data interchange pattern, let's examine how one might go about specifying and publishing pattern definitions. Meta data interchange patterns can be specified in terms of either *human-readable* or *software-consumable (*i.e., *machine-readable) specifications:*

- Human-readable specifications enumerate all metamodel classes and associations comprising the projection as literals, based on the use of logical class and association names within the namespace of the metamodel. The restriction is specified as a collection of constraints on the projection, in either a natural language, such as English, or some formal notation, such as the Object Constraint Language (OCL) [23]. Anchor elements are also designated by the logical names of the metamodel classes whose instances serve as anchor elements.

- Software-consumable specifications consist of formally defined models that reference (either directly or indirectly) instances of the metamodel classes and associations forming the projection and those instances designated as anchor elements. The specification may also include a collection of constraints against

the projection (that is, the restrictions), expressed in some formal, directly executable language, such as OCL.

An important aspect of the software-consumable specification, of course, is that it can be directly consumed and understood by software tools. For example, a pattern-aware tool could discover and download some meta data interchange pattern of interest from a Web-based pattern catalog or pattern repository and use it as the basis for dynamically engaging in pattern-based interchange with other tools. The design of a software-consumable pattern specification is treated subsequently in this paper.

A Specification Template for Meta Data Interchange Patterns

Human-readable meta data interchange patterns are specified in terms of a standard specification, or *pattern template*. The template greatly simplifies the process of publishing patterns, because it provides a uniform way of expressing pattern content. This makes it easy for pattern users to search for published patterns that they can either use directly or leverage in the creation of new patterns. These are principles that are well established within the software design patterns community [3, 4].

The following paragraphs describe the entries of a possible pattern template for constructing human-readable meta interchange pattern specifications. This template satisfies a number of requirements for such specifications, including support for inclusion in Web-based pattern catalogs, and appropriate pattern "characterization" to facilitate the effective browsing of catalogs by users. This template definition is generally consistent with previously proposed templates in [12, 19, and 20], but is simpler and also includes entry names that are better aligned with more conventional terminology used by the software design patterns community.

Name. Name of the pattern.

Intent. States the purpose of the pattern. Should be as terse as possible, to facilitate the rapid browsing of a catalog for patterns that solve specific problems.

Version. Version of the pattern.

URI. Uniform resource identifier pointing to the master copy of the pattern specification. Facilitates locating and downloading the specification via the Web.

Contributor. Person or organization that contributed this pattern to the pattern community or catalog. Should include an email address or home page URL.

Problem. Description of the meta data interchange problem that this pattern is intended to solve (this entry is sometimes referred to as "context" in conventional design pattern templates).

Forces. Describes in some detail the overall requirements and constraints placed on the solution. Summarizes the dialectics influencing the overall shape of the solution.

Solution. Provides an overview of the meta data interchange pattern being proposed as a solution to the stated problem.

Classification. Specifies the *structural classification* of the pattern. Structural classification describes the level of composition that this pattern usually occupies in the layering of composite patterns. The "classification" key word enables users to search a catalog for patterns falling into specific structural classifications. The following classifications are defined:

> ***Micro pattern*** – A fine-grain meta data pattern that is often re-used in the composition of larger patterns (such as *domain patterns*, which are described below). Micro pattern is somewhat analogous to the concept of "micro-pattern" described in early pattern theory [1, 9].
>
> ***Domain pattern*** – A medium-grain pattern that is generally used to solve some domain-level integration problem (e.g., a specific sub-domain of some broader subject area). Domain patterns are the basis for defining interchange context. The projections of domain patterns often span multiple sub-packages of a metamodel. For this reason, a domain pattern is roughly analogous to the concept of a "framework" in traditional software design pattern theory [4].
>
> ***Macro pattern*** – A course-grain pattern that serves as a means of organizing and labeling model content for interchange. A macro pattern generally specifies some container in which the model content is stored, and some means of identifying the content as a realization of some other pattern (usually a domain pattern).

Category. Specifies the *usage category* of the pattern. Usage category designates, in general, how the pattern is usually employed in solving some meta data integration problem. Like "classification", the "category" key word facilitates pattern catalog search. The following categories are defined:

> ***Interchange*** – Any pattern that generally facilitates interchange of shared meta data. Most macro patterns fall into this category.
>
> ***Mapping*** – Any pattern that establishes mappings or correspondences between model elements.

Typing – Any pattern used to further classify model elements as members of some semantic domain.

Extension – Any pattern that extends the semantics or structure of a model.

Interpretation – Any pattern that aids in interpreting the semantics of a model.

Generation – Any pattern that facilitates the automated generation (or partial generation) of an implementation of a model.

Structural – Any pattern that facilitates the construction of more complex meta data structures from simpler ones.

***Projection*.** Enumerates the metamodel classes and associations comprising the projection.

***Restriction(s)*.** Specifies any constraints on classes and associations of the projection.
***Anchor Element(s)*.** Enumerates any metamodel classes whose instances are to serve as anchor elements.

***Parameters*.** Defines the parameter values used in binding this pattern to some pattern realization.

***Related Patterns*.** Lists any other meta data interchange patterns that this pattern is closely related to, collaborates with, or is composed from. Includes any URLs pointing to these pattern descriptions.

***Example*.** An example of this pattern being used to solve a meta data interchange problem. Usually consists of instance diagrams and equivalent XMI fragments, if applicable.

A Brief Word on Methodology

The theory developed in this paper is focused on the definition and specification of meta data interchange patterns, but it does not address any particular methodology for actually discovering or formulating patterns. This is because we are currently more concerned with developing a precise definition of the modeling artifacts, rather than any process that might yield these artifacts (and, as always, we are more concerned with logical expression than with physical implementation).

A simple pattern development methodology is outlined in [19]. However, any methodology or process that is effective in discovering and codifying patterns may be used. What is more important is that producers and consumers use mutually agreed-upon

patterns as a foundation for meta data integration. And what is most important is that the domain experts and modelers themselves be the primary authors of domain-specific interchange patterns. Establishing public-domain specification artifacts and actively managed, Web-based pattern catalogs greatly facilitates this objective.

Examples of Meta Data Interchange Patterns

We'll now briefly describe three examples of useful meta data interchange patterns. These are: *Star Schema*, *Surrogate Key*, and *Unit Of Interchange*.

Star Schema. This is a domain pattern based on the relational star schema pattern developed informally in this paper. It represents a highly simplified description of a star schema that could, nonetheless, be leveraged in solving real-world data warehousing and business intelligence problems. For example, one could easily envision using it as the basis for interchanging meta data based on the dimensional data warehousing models described by Kimball in [6]. A specification of this pattern, based on the template defined in the preceding section, is provided in the Appendix of this paper.

Surrogate Key. This is a micro pattern that models the use of surrogate keys in the joining of several information structures (usually relational tables). A surrogate key is a unique identifier that has been automatically generated and carries no specific semantic meaning [7]. The Surrogate Key pattern provides a standard way for structuring meta data when modeling surrogate keys. For example, an instance of the Star Schema pattern might be comprised of instances of the Surrogate Key pattern. A specification of this pattern may be found in [20].

Unit Of Interchange. This is a macro pattern that defines both a container of a pattern instance, and a label (or tag) on that container. The label identifies a particular domain pattern that the container's content conforms to. Unit Of Interchange provides a basic framework for interchanging or storing instances of domain-specific patterns. A specification of this pattern may be found in [19, 20]. The derivation of this pattern is presented at length in [19].

A Software Consumable Pattern Specification: The CWM MIP Model

A software-consumable pattern specification would provide a number of benefits:

- Pattern definitions can readily be interchanged between software tools, stored in pattern repositories, and made available for automated download from Web-based pattern catalogs.

- A pattern definition can be interchanged along with its pattern instances. This establishes a direct and machine-understandable semantic link between pattern description and pattern instance that is not possible with human-readable pattern specifications.

- The development of a new generation of visual modeling tools for pattern development is greatly facilitated by having a standard mechanism for the import, export, and general interchange of pattern definitions.

- The development of a new generation of highly dynamic, pattern-driven software tools is likewise greatly facilitated by having a standard mechanism for the import, export, and general interchange of pattern definitions.

- A formal pattern model allows for greater symmetry between import-export and request-response styles of meta data interchange. Note that a pattern definition is equivalent to a *query* for one or more pattern instances that match (or realize) the pattern definition. This means that there is very little difference between import-export and request-response -style interactions, as far as representation of content is concerned. Both interaction styles are fully supported by the same exchange format.

The OMG recently developed just such a software-consumable pattern specification. Known as the *CWM Metadata Interchange Patterns Specification* [12], this specification consists of a formal, MOF-compliant metamodel that is capable of describing meta data interchange patterns, as well as linking those pattern definitions to actual meta data.

Figure 6 illustrates below the high-level classes of the CWM Metadata Interchange Pattern (CWM MIP) metamodel:

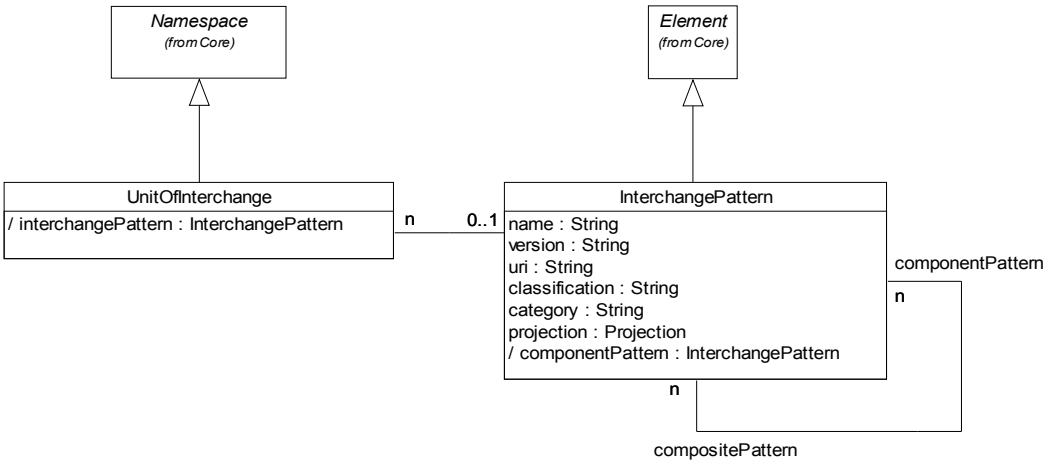

Fig. 6: CWM MIP Metamodel

The *InterchangePattern* class is essentially a model of a meta data interchange pattern. Its attributes consist of *name*, *version*, *URI* (the purpose of which is to provide a link to a human-readable specification of the same pattern), *classification*, *category*, and *projection*. A reflexive association on the class facilitates composing patterns out of other patterns.

UnitOfInterchange provides a container for storing meta data described by Interchange-Pattern. Since UnitOfInterchange inherits from CWM's *Namespace* class, an instance of UnitOfInterchange can contain instances of any CWM classes that are not otherwise exclusively owned by other namespaces. Normally, this would be one of the outer-lying, anchor elements of some pattern instance.

Figure 7 below illustrates the type model of CWM MIP, which defines various classes used to define the projection attribute of InterchangePattern. Two concrete subclasses of the abstract Projection base class are defined: *SemanticContext*, which models what is essentially the same information provided by the "projection", "restriction", and "anchor element" entries of the human-readable pattern specification template, and *GraphSubset*, an expression whose evaluation, provides a more coarsely grained projection in terms of instances of a physical sub-graph of the CWM metamodel.

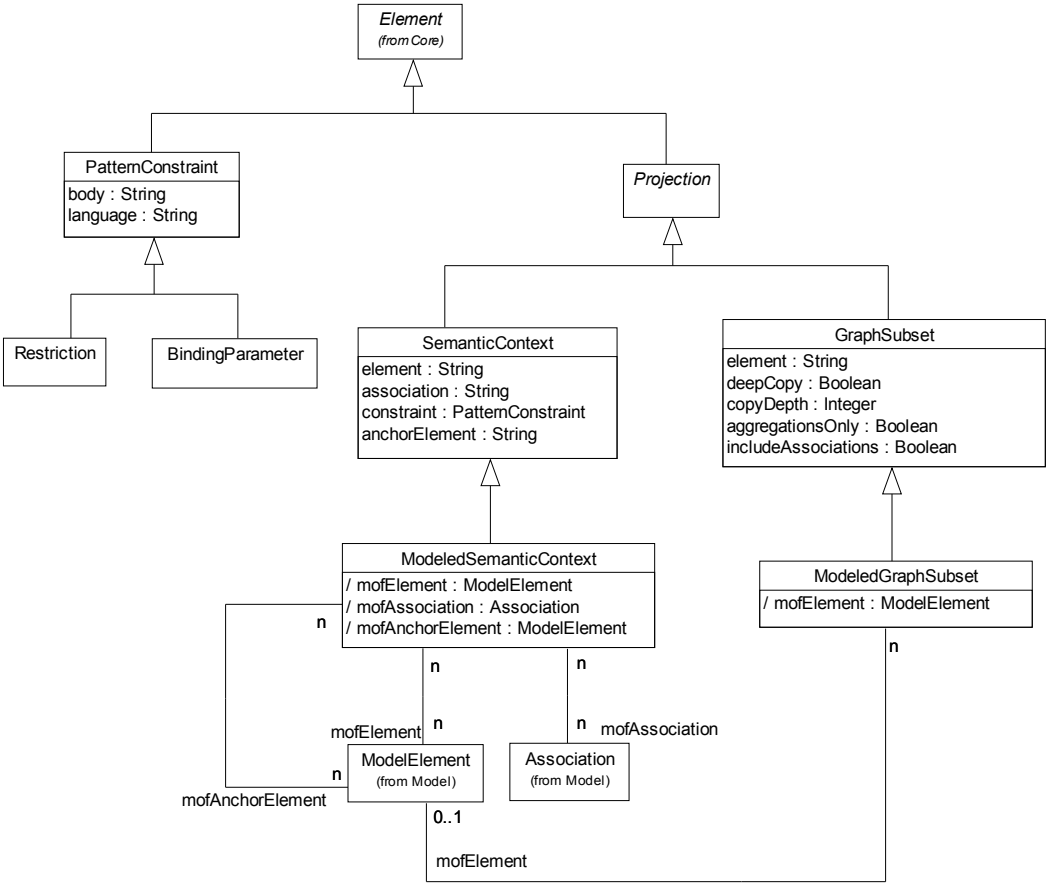

Fig. 7: CWM MIP Type Model

The attributes of SemanticContext identify the classes and associations comprising the metamodel projection via arrays of logical names. This enables software tools to implement CWM MIP without having to implement the MOF. MOF instances (that is, metamodel elements) are simply enumerated according to established naming standards, rather than being explicitly modeled. On the other hand, MOF-based tools that wish to provide explicitly modeled pattern definitions may do so via the equivalent *ModeledSemanticContext* and *ModeledGraphSubset* classes.

Finally, PatternConstraint is a simple class for modeling the constraints associated with both restrictions on the projection and binding parameters. This class allows constraints to be expressed in any textual language. The only requirement of CWM MIP in this regard is that, if OCL is specified, the body must contain an OCL expression that is both syntactically valid and relevant to the modeling context of the projection (that is, the expression must be at the same level of abstraction as the metamodel and must reference

some class of the projection as its context object). The following three scenarios describe typical usage of the CWM MIP metamodel:

- A pattern authoring tool instantiates the InterchangePattern class and defines some pattern by assigning relevant values to its attributes. This object could then be stored in a pattern repository, or serialized in an XMI file and exported to the environment. Note that there is no need for the pattern definition to have any associated meta data; pattern definitions can be created and interchanged between tools independently of any meta data they might describe.

- A pattern-aware consumer imports a pattern definition of interest from a pattern repository or XMI file. The consumer then uses the pattern definition to expand its own meta data import capabilities. In other words, it has just "learned" a new pattern, and is now capable of consuming and making intelligent use of meta data conforming to that pattern. This is an example of a dynamic, pattern-driven tool.

- A pattern-enabled producer decides to export several application-level models that all conform to the same pattern, together with a definition of that pattern. The producer first creates an instance of InterchangePattern and initializes it appropriately. The producer then creates the application models, along with an instance of the UnitOfInterchange class for each model. The producer adds the upper-most element of each model to its respective UnitOfInterchange instance, and then connects the UnitOfInterchange instances to the single Interchange-Pattern instance, via the shared association between the two classes. The producer then serializes this entire structure to a single XMI file and exports the file to the environment. Note that the exported XMI file contains four top-level elements: The three distinct units of interchange, and the single, common pattern definition.

Generating and Using a Pattern-Based Repository

As a proof-of-concept of the notion of a pattern-oriented, meta data repository, we combined the CWM MIP metamodel with an MDA repository tool [5, 22] to automatically generate, initialize, and then populate, an online repository of both pattern definitions and related meta data. The following paragraphs describe the steps.

We first launched our repository tool, the Complex Information Manager, Standard Edition (CIM SE), from Unisys Corporation [22]. CIM is an open-source, MOF-aware, repository service that also serves as the JMI (JSR-40) Reference Implementation [5]. Within CIM, we created a *facility* (CIM's concept of a persistent information service) named "Pattern Based Meta Data". Then, within the facility, we created two repositories, one for storing patterns, called "CWM Patterns", the other for storing the meta data itself, called "CWM Models". Next, we imported XMI renderings of both the CWM MIP and

CWM metamodels into the “CWM Patterns” and “CWM Models” repositories, respectively, thereby initializing each repository with a metamodel definition.

The screen shot in Fig. 8 shows the CIM administrative console following importation of the two metamodels. Note that the “CWM Models” repository lists multiple, *top-level models*, each corresponding to one of the CWM packages (not all are visible in the screen shot). By comparison, the “CWM Patterns” repository shows the single top-level model of the CWM MIP (which is a much smaller metamodel than CWM). Also shown in the display are the top-level models of the MOF Model, which are essentially built into CIM. The next step was to generate *servers* (that is, Java interfaces and corresponding Java implementation classes) for managing each repository. CIM automatically generates the Java code, based on the metamodels, and according to JMI’s MOF-to-Java mapping rules [5]. Once the servers were generated, it was possible to begin using the repositories.

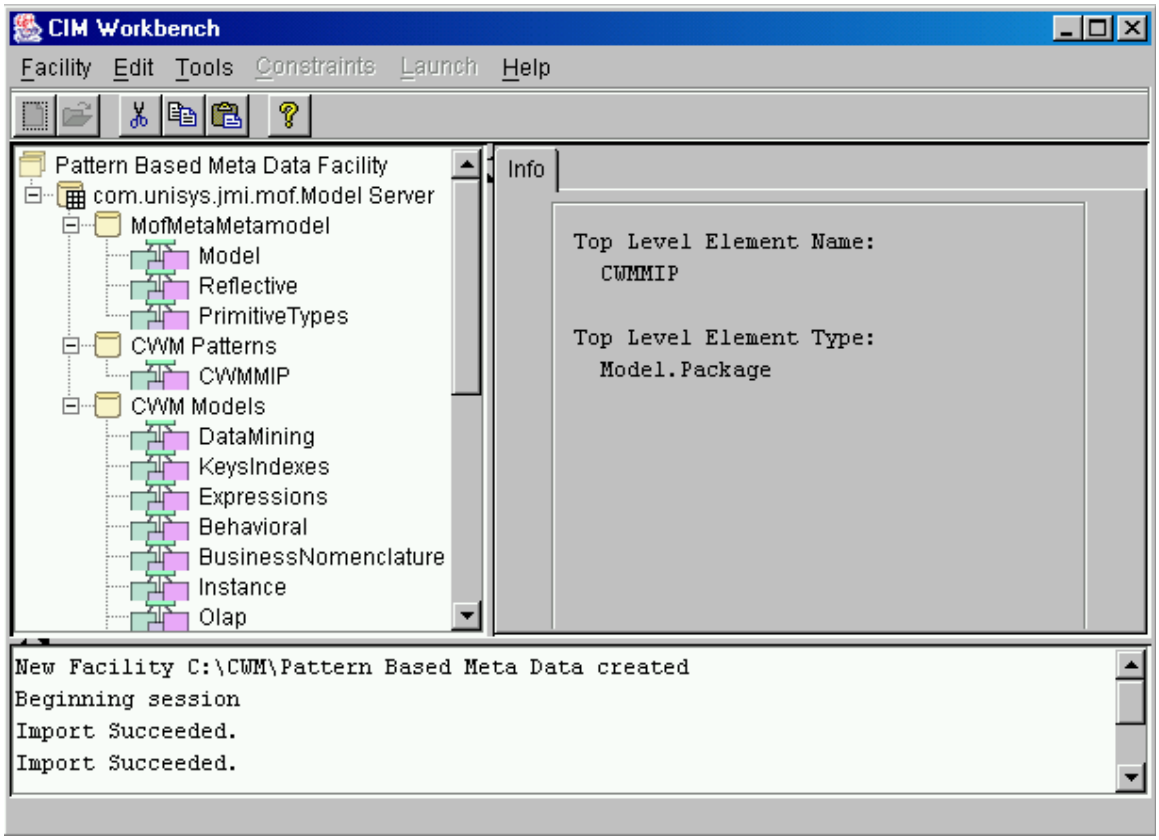

Fig.8: Generating A MOF-Based Repository of both Models and Patterns

A simple Java client program was then coded. The client connected to the CIM facility, created the relational star schema model shown in Figure 3, as an instance of the CWM metamodel, and then created the Star Schema pattern defined in the Appendix of this paper, as an instance of the CWM MIP metamodel. The model and pattern were then kinked together via the association relating the UnitOfInterchange and

InterchangePatterns classes. Both instances were then serialized to a single XMI file for export from the facility.

The following Java code fragment illustrates the logic of the Java client. The code creates and manages instances of the various metamodel elements via the JMI-prescribed Java interfaces that were automatically generated by CIM and exposed by the CIM servers. For brevity, we've omitted most of the actual code. The fragment focuses mainly on pattern creation logic. The star schema model itself resides within a single instance of the Catalog interface, called *catalog*. The fragment does not include the code that created the model:

[SAMPLE CODE SEQUENCE GOES HERE]

Conclusions and Directions for Future Research

This paper provided a comprehensive treatment of the application of a new class of pattern, *meta data interchange patterns*, to the problem domain of model-based interoperability. In highly collaborative and heterogeneous computing environments, model interchange between software tools requires not only a common language and interchange format, but some means of precisely describing expected variations in model structure and content. Doing so ensures that interchanged models are consistent with their intended purpose, while also simplifying the construction of model import and export logic. This paper has shown, through the use of a real-world example, how meta data interchange patterns can accomplish this.

Standard artifacts for the specification and publication of meta data interchange patterns, inspired by similar means used in the software design patterns community, were also proposed. In particular, an industry-standard, software-consumable specification model for meta data interchange patterns developed by the OMG, the CWM MIP metamodel, was introduced, and a case study of using this model in the creation of a pattern-oriented, meta data repository was presented.

Areas of future research for meta data interchange patterns include:

- Alignment of the CWM MIP with the UML 2.0, MOF 2.0, and MOF 2.0 Query, View, and Transformation (QVT) efforts. In particular, it is hoped that the CWM MIP metamodel will be able to use the UML 2.0 Infrastructure as its base metamodel. This will have the effect of allowing the CWM MIP metamodel to be generally applicable across all of MDA, not just CWM.

- Integration of the CWM MIP metamodel with several of the various open-source Java IDEs that are incorporating MDA modeling capabilities, including the

Eclipse IDE and Eclipse Modeling Framework (http://www.eclipse.org/emf/), as well as Sun's NetBeans IDE (http://www.netbeans.org).

- Ongoing collection, codification, and publishing of useful meta data interchange patterns, including expressions of these patterns in terms of CWM MIP instances serialized as XMI documents.

- Ongoing development and experimentation with pattern-based and pattern-driven tools. A specific goal in this regard is attempting to leverage meta data interchange patterns in the development of totally dynamic and adaptive model-driven architectures, of the nature described in [17].

Acknowledgements

Special thanks go to OMG colleagues Dan Chang, Sridhar Iyengar, David Mellor, Pete Rivett, and Doug Tolbert, for their various contributions the development of the CWM MIP, and for participation in the CWM MIP Finalization Task Force.

References

1. Alexander, Christopher, S. Ishikawa, and M. Silverstein. *A Pattern Language*. Oxford University Press, 1977.
2. Codd, E.F. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM*. Vol. 13. No. 6. pp. 377-387. Association for Computing Machinery, Inc., 1970. Internet: http://www.acm.org/classics/nov95
3. Gamma, Erich, Richard Helm, Ralph Johnson, John Vlissides. "Design Patterns: Abstraction and Reuse of Object-Oriented Designs." *Proceedings, ECOOP '93*. Heidelberg: Springer-Verlag, 1993.
4. Gamma, Erich, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison Wesley Longman, Inc., 1995.
5. Java Community Process. Java™ Metadata Interface Specification Version 1.0 (JSR-40). Sun Microsystems, 2002. Internet: http://jcp.org/.
6. Kimball, Ralph. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. New York: John Wiley & Sons, Inc., 1996.
7. Kimball, Ralph, and Richard Merz. *The Data Webhouse Toolkit: Building the Web-Enabled Data Warehouse*. New York: John Wiley & Sons, Inc., 2000.
8. Kleppe, Anneke, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture – Practice and Promise*. Reading, MA: Addison Wesley Longman, Inc., 2003.
9. Lea, Doug. "Christopher Alexander: An Introduction for Object-Oriented Designers." SUNY Oswego, 1997. Internet: http://gee.cs.oswego.edu/dl/ca/ca/ca.html
10. OMG. *Common Warehouse Metamodel Specification, Version 1.1*. Needham, MA: Object Management Group, 2003. Internet: http://www.omg.org/technology/documents/formal/cwm.htm.
11. OMG. CORBA, XML and XMI Resource Page. Needham, MA: Object Management Group, 2003. Internet: http://www.omg.org/technology/xml/index.htm
12. OMG. *CWM Metadata Interchange Patterns Specification, Version 1.0.* Needham, MA: Object Management Group, 2004. Internet: http://www.omg.org/technology/documents/formal/cwm_mip.htm.
13. OMG. Data Warehousing, CWM and MOF Resource Page. Needham, MA: Object Management Group, 2003. Internet: http://www.omg.org/technology/cwm

14. OMG. Model-Driven Architecture Home Page. Needham, MA: Object Management Group, 2003. Internet: http://www.omg.org/mda
15. OMG. Unified Modeling Language Resource Page. Needham, MA: Object Management Group, 2003. Internet: http://www.omg.org/technology/uml/index.htm
16. Poole, John. "The Common Warehouse Metamodel as a Foundation for Active Object Models in the Data Warehousing Environment." ECOOP 2000 Workshop on Meta data and Active Object Models, 2000. Internet: http://www.adaptiveobjectmodel.com/ECOOP2000 or http://www.cwmforum.org/paperpresent.htm
17. Poole, John. "Model-Driven Architecture: Vision, Standards, and Emerging Technologies." ECOOP 2001 Workshop on Metamodeling and Adaptive Object Models, 2001. Internet: http://www.adaptiveobjectmodel.com/ECOOP2001 or http://www.omg.org/mda/
18. Poole, John, Dan Chang, Douglas Tolbert, and David Mellor. *Common Warehouse Metamodel: An Introduction to the Standard for Data Warehouse Integration*. New York: John Wiley & Sons, Inc., 2002. Internet: http://www.wiley.com/compbooks/poole
19. Poole, John, Dan Chang, Douglas Tolbert, and David Mellor. *Common Warehouse Metamodel Developer's Guide*. New York: John Wiley & Sons, Inc., 2003. Internet: http://www.wiley.com/compbooks/poole
20. Poole, John. CWM Meta data Interchange Patterns Catalog. New York, John Wiley & Sons, Inc., 2003 Internet: http://www.wiley.com/compbooks/poole/CWM_Guide/patterns.html
21. Tolbert, Doug. "CWM: A Model-Based Architecture for Data Warehouse Interchange." Workshop on Evaluating Software Architectural Solutions 2000. University of California at Irvine, 2000. Internet: http://www.cwmforum.org/paperpresent.html
22. Unisys Corporation. Complex Information Manager, Standard Edition. Unisys Corporation, 2003. Internet: http://java.sun.com/products/jmi or http://ecommunity.unisys.com
23. Warmer, Jos, and Anneke Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Reading, MA: Addison Wesley Longman, Inc., 1999.

About the Author

John Poole is a Distinguished Software Engineer at Hyperion Solutions Corporation. He is one of several co-developers of the Common Warehouse Metamodel (CWM) within the Object Management Group. He currently leads the Java OLAP Interface technology effort within the Java Community Process, as well as the CWM Metadata Interchange Patterns initiative within the OMG. John actively lectures and writes on metadata integration and modeling issues relevant to the business performance management, data warehousing, business intelligence and OLAP domains. He is the lead author and editor of two books on the Common Warehouse Metamodel, recently published by John Wiley & Sons.

