Hyperion®

Meta Data Interchange Patterns: The OMG's New Model for Precise Meta Data Interchange

Presented by:
John Poole, Distinguished Software Engineer
Hyperion Solutions
john_poole@hyperion.com

Dr. Jon Siegel, VP Technology Transfer
Object Management Group
siegel@omg.org

What are Meta Data Interchange Patterns?

Hyperion

- A means of achieving precise meta data interchange between dissimilar software components, tools, and applications
- An application of “pattern techniques” to the problem of meta data sharing and interchange
- A method of describing useful model forms when using generic modeling languages, such as OMG’s UML and CWM
- A formal pattern model that ultimately enables the development of pattern-driven / pattern-aware software tools (OMG’s CWM MIP)

Presentation Overview

- Introduction to OMG by Dr. Jon Siegel
- Overview of Meta Data Driven Interoperability
- Precision and reliability issues for Meta Data Driven Interoperability
- Introduction to Meta Data Interchange Patterns (MIPs) and how they resolve the precision and reliability issues
- Economics of MIP approach (enhanced ROI)
- Overview of OMG CWM MIP Specification
- Case study: Star Schema database pattern

Presentation Overview (Continued)

- Survey of Hyperion’s ongoing work in this area (Hyperion Developer Network)
- Future directions
- Conclusions & Wrap-Up
- Q&A
- Main presentation: About one hour and fifteen minutes
- Q&A: Remaining fifteen minutes

Poll: Are you familiar with OMG standards for modeling and integration?

- *[PlaceWare Yes/No Poll. Use **PlaceWare** > **Edit Slide Properties...** to edit.]*
- Yes
- No

Poll: Do you use or plan to use software products based on standards?

- *[PlaceWare Multiple Choice Poll. Use* ***PlaceWare*** *>* ***Edit Slide Properties...*** *to edit.]*
- UML
- CWM
- MOF
- XMI
- MDA
- Other

Dr. Jon Siegel, Object Management Group

Meta Data Interchange and Interoperability

- Meta data sharing is key to integrating dissimilar software components, products, tools, and applications
- Meta data sharing requires a common definition of meta data; often approached in terms of *formal modeling* and *formal modeling languages* (e.g., ER, UML, XML Schema)
- Meta data sharing also requires a common interface or interchange format (e.g., Java or XML documents)

Meta Data Interchange and Interoperability

- In recent times, standards-based approaches to meta data interchange have been developed by industry leaders working together in various consortia
- Object Management Group standards:
  - Unified Modeling Language (UML) [12]
  - Meta Object Facility (MOF) [10]
  - XML Metadata Interchange (XMI) [8]
  - Common Warehouse Metamodel (CWM) [7, 13, 14, 15, 16, 17]
  - Model-Driven Architecture (MDA) [6, 11]
- Open standards and consortia-based approaches ensure equitable vendor representation and relative vendor-independence of delivered solutions

Common Warehouse Metamodel (CWM)

- Extends the UML to include Data Warehousing and Business Intelligence concepts
- Provides a formal modeling language for describing meta data in the DW/BI domain
- Indirectly (by virtue of MOF and XMI) provides an XM-based interchange format
- MDA: Implies potential mappings of CWM to other representations or implementation models (e.g., Java and J2EE APIs, C#, .NET)
- For example: Java™ Metadata Interface (JMI) [4]

Common Warehouse Metamodel (CWM)

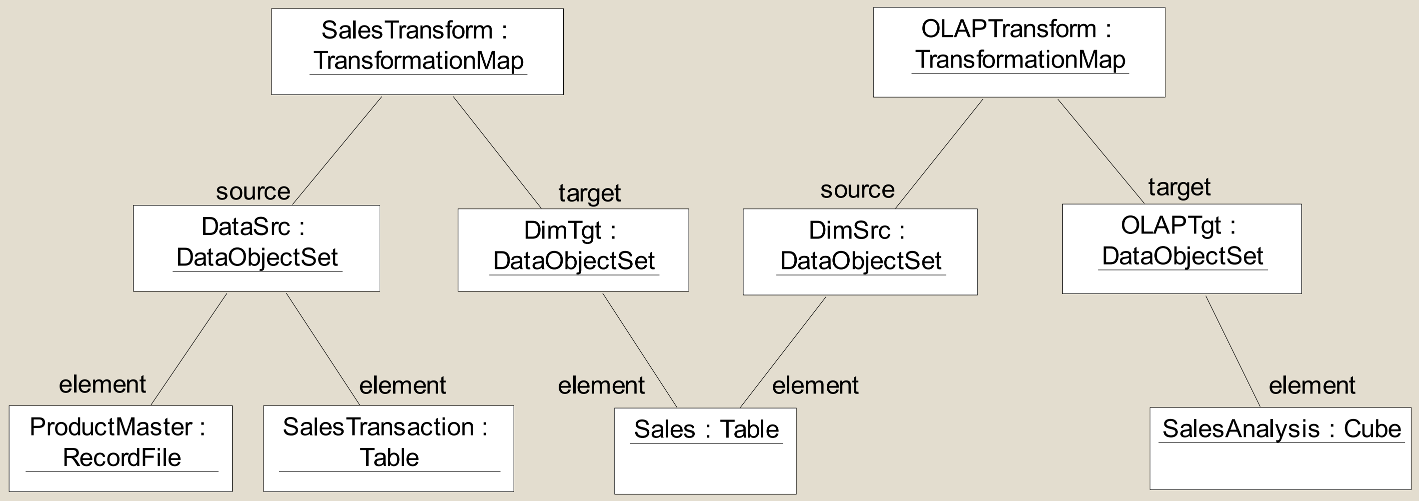
Hyperion

CWM Layered Architecture:

Management	Warehouse Process	Warehouse Operation				
Analysis	Transformation	OLAP	Data Mining	Information Visualization	Business Nomenclature	
Resource	Object	Relational	Record	Multidimensio nal	XML	
Foundation	Business Information	Data Types	Expressions	Keys and Indexes	Software Deployment	Type Mapping
Object Model	Core	Behavioral	Relationships	Instance		

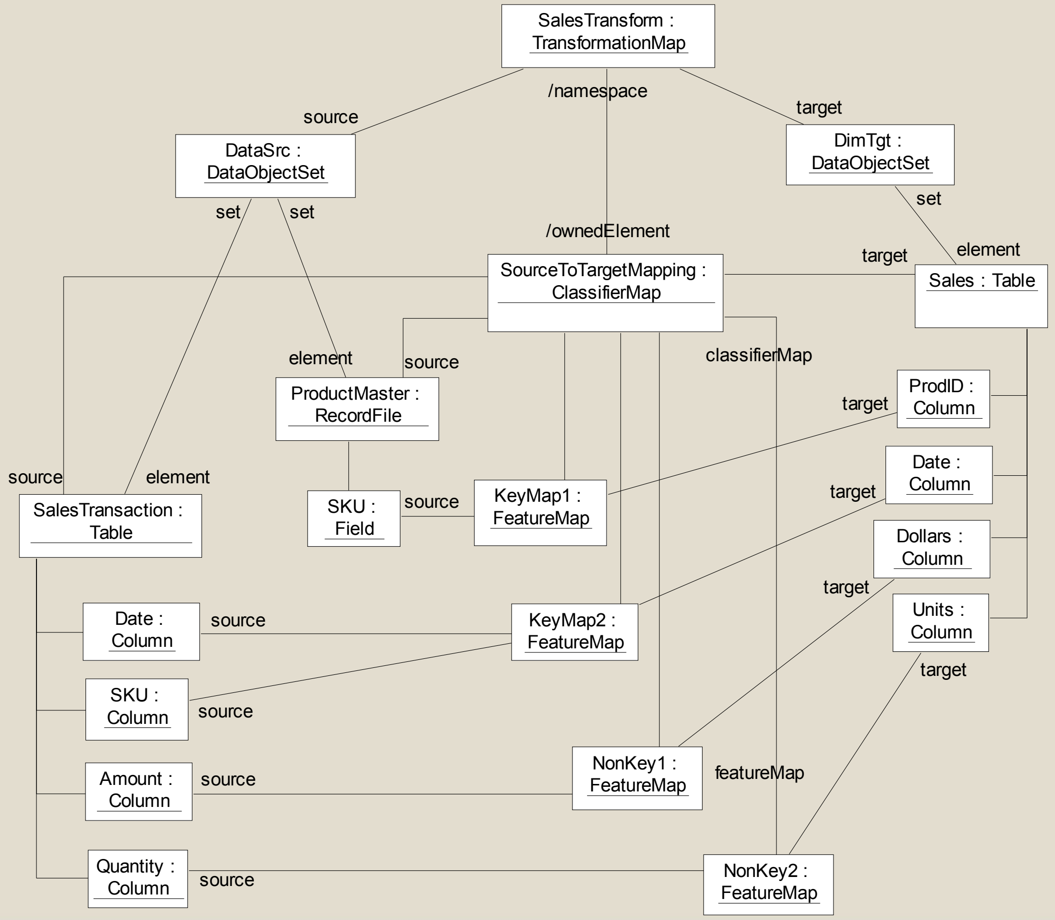
Common Warehouse Metamodel (CWM)

CWM Example #1: High-level model of an ETL process:



Common Warehouse Metamodel (CWM)

CWM Example #2: Detailed model of same ETL process:



Meta Data Interchange Scenario

- Data warehouse architect designs the ETL model using a CWM-aware graphical modeling tool or CWM-aware ETL front-end
- CWM ETL model is stored in a meta data repository
- The ETL back-end process imports the model from the front-end or the repository.
- The ETL back-end process either translates the ETL model into its private, internal meta data, or interprets it directly while carrying out ETL operations (*intelligent defaulting* of information gaps)

Meta Data Interchange Scenario

- The CWM ETL model is readily used by any other CWM-aware tools in the environment that requires ETL meta data (concept of an instance of meta data as an *externalized* and *platform-independent model*, or “PIM” in MDA terms)
- The data warehouse architect can readily maintain and incrementally refine the repository image of the ETL model
- In general, facilitates an agile and iterative approach to system modeling, deployment, and operation

Why Meta Data Interchange Patterns?

- Formal modeling languages like CWM and UML are fully capable of describing domain-specific meta data for purposes of interchange
- CWM and UML are highly expressive and extremely flexible
- But neither CWM nor UML (nor any other formal language, for that matter), provide a means of expressing the *intent* (or *meaning,* or *semantic context*, or *intended effect,* or *intended use*) of any particular model or interchange event
- Furthermore, there is a (potentially) unbounded degree of variation in syntactically-valid models

Why Meta Data Interchange Patterns?

- For example: The high-level ETL process model versus the refined ETL process model with source-to-target mappings
- Collaborative environment of heterogeneous, largely autonomous tools: Potentially many meta data publishers and consumers
- Whether a particular model is *useful* or *meaningful* (from an information perspective) really depends on what some consuming software process expects or requires
- So, we see three (closely related) issues with regard to meta data interchange:
  - Potential for unbounded variation in model structure/content
  - Automated processes need to deal with model variation
  - Human modelers must be able to *describe* model variation in a manner intelligible to *both* humans and automated processes

Why Meta Data Interchange Patterns?

- Early validation of CWM by the CWM Committee within the OMG: Interoperability Showcase
- Participants got a hard lesson in these issues!
- Participants discovered the best way to deal with model variation was to agree on useful, general forms of model structure and composition
- We found that, as long as interchanged models conformed to these general forms, a wide variety of models could easily be understood by our software tools – more specifically, import and export adapters were coded to handle these general forms, rather than any particular models

Why Meta Data Interchange Patterns?

- It became apparent that what we were dealing with were *patterns* for meta data
- By “pattern”, in this particular case, we mean an agreed-upon, general form (or perhaps *idiom*, or *idiomatic usage)* of the CWM language
- It was also apparent that this problem had been solved before, in a somewhat different setting: That is, the famous “gang-of-four”-style *software design patterns* [1, 2]
- We reasoned we could co-opt and effectively apply much of the software design pattern machinery to the problems of unbounded variation and meaning in the case of *highly generalized model interchange*

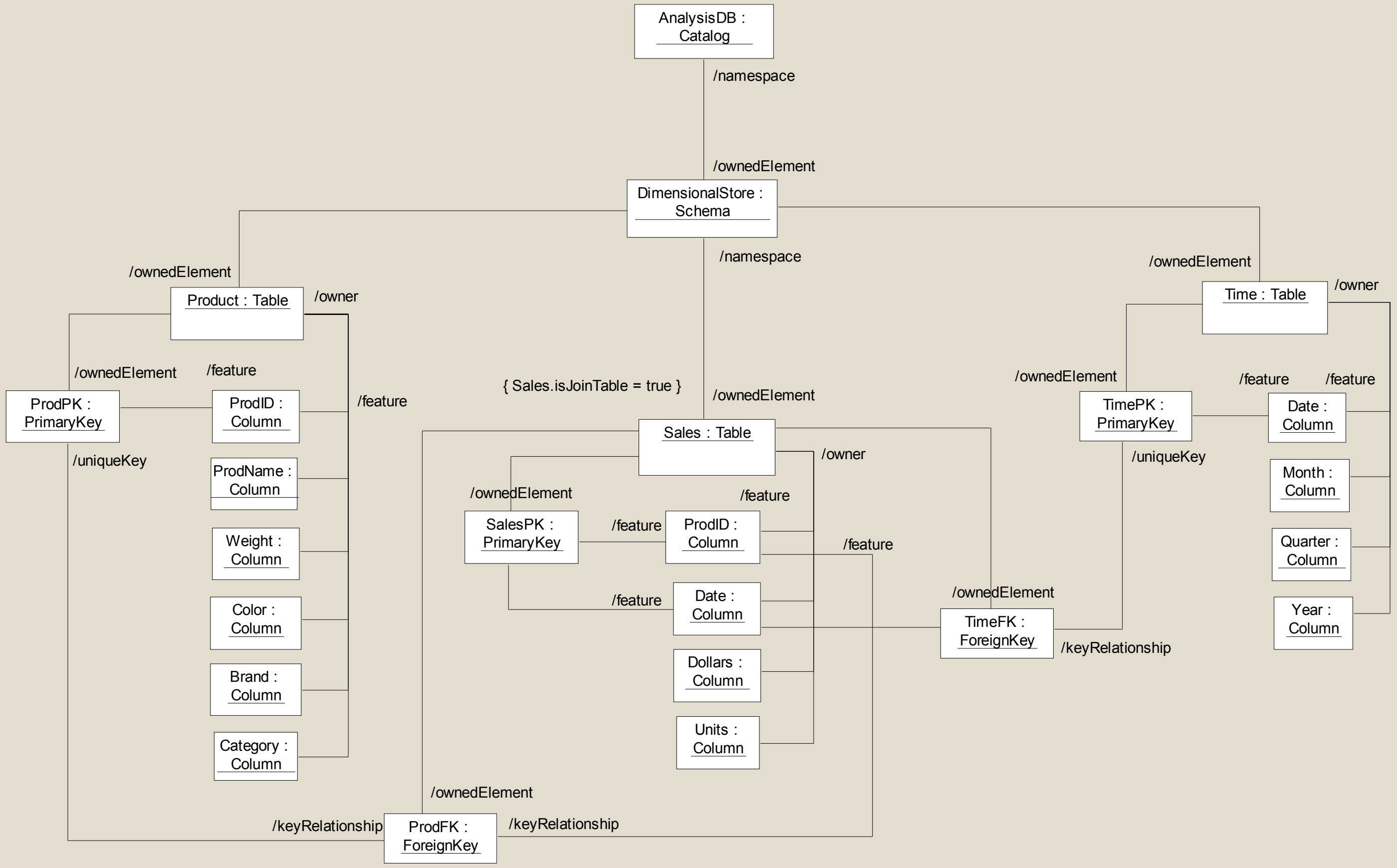
“ A *pattern* is a description of communicating objects and classes that are customized to solve a general design problem within a particular context”
– Gamma, et al [2]

How about another example?

- Consider the famous Relational Star Schema, which forms the basis for much of data warehousing and business intelligence solutions
- The Star Schema is essentially an organizational pattern for relational database tables
- Ralph Kimball’s *Data Warehouse Toolkit* is essentially a collection of Star Schema meta data patterns [5]
- But there is tremendous variation across these Star Schema designs (as Kimball’s book demonstrates)
- How does one devise a single description of all possible Star Schemas?

How about another example?

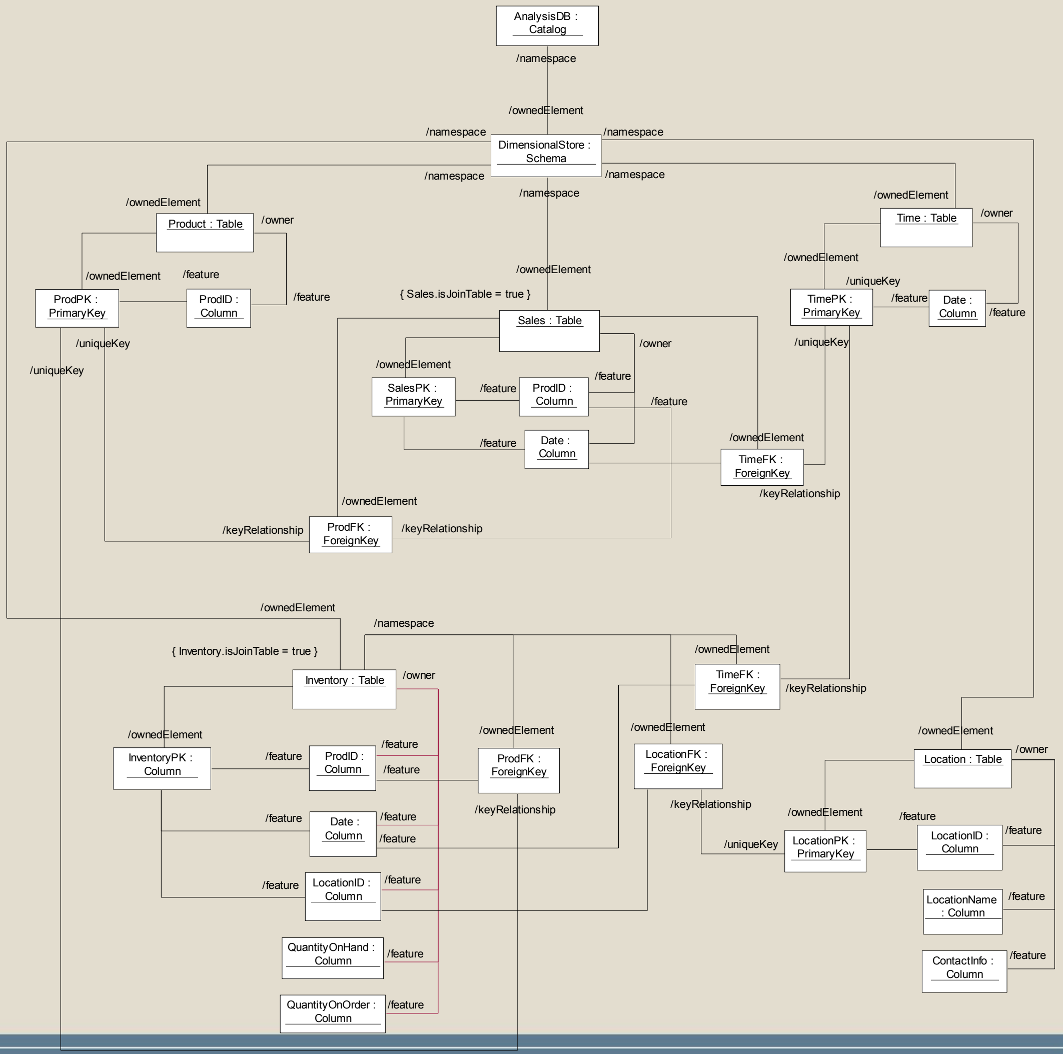
CWM Example #3: A Star Schema database model:



How about another example?

CWM Example #4: A slightly more complex Star Schema model (illustrated on the next slide):

How about another example?



A Major Dilemma Arises…

Hyperion

How do I even *begin* to describe a general form (pattern) that could be used to classify either of the preceding examples (as well as any of Ralph Kimball’s vertical data warehouse models, for that matter) as what one would call a “relational star schema”?

Informal Derivation of the MIP Concept

- It turns out that we can begin to address this issue by asking ourselves three questions regarding the structure and content of any interchanged model:
  - What portion of the underlying metamodel (e.g, CWM or UML) is required to represent the basic structure of any Star Schema database?
  - What restrictions on instances of model classes are reasonable to expect in a Star Schema design (e.g., one fact table, or multiple fact tables)?
  - Is there generally some particular class that constitutes a “root element” or “entry point” of any Star Schema design (e.g., the fact table, or perhaps a schema element containing the rest of the model)?

Informal Derivation of the MIP Concept

- Reasonable answers to the previous three questions might consist of the following (assuming CWM):
  - What portion of the underlying metamodel is required to represent the basic structure of any Star Schema database? **The CWM modeling classes of Catalog, Schema, Table, Column, Primary Key, Foreign Key and all related associations.**
  - What restrictions on instances of model classes are reasonable to expect in a Star Schema design? **At most one instance of Catalog, at most one instance of Schema, and at least three instances of Table. One Table instance in particular has a join relation (primary key / foreign key) to the other Table instances.**
  - Is there generally some particular class that constitutes the “root element” or “entry point” of the Star Schema design (e.g., the fact table, or perhaps a schema element containing the rest of the model)? **Any of Catalog, Schema, or Table.**

Informal Derivation of the MIP Concept

Hyperion

- Assuming that all meta data producers and consumers agree on the preceding questions + answers as a description of the general form of any Star Schema database model, then a wide variety of Star Schema models can be interchange and readily understood by consumers
- The fact that there is an unlimited amount of potential variation in Star Schema design has not prevented us from formulating a simple, finite description of all Star Schema models, based on a reasonable, idiomatic use of our modeling language
- It should be obvious that what we have just defined (informally) is a *Meta Data Interchange Pattern (MIP)*

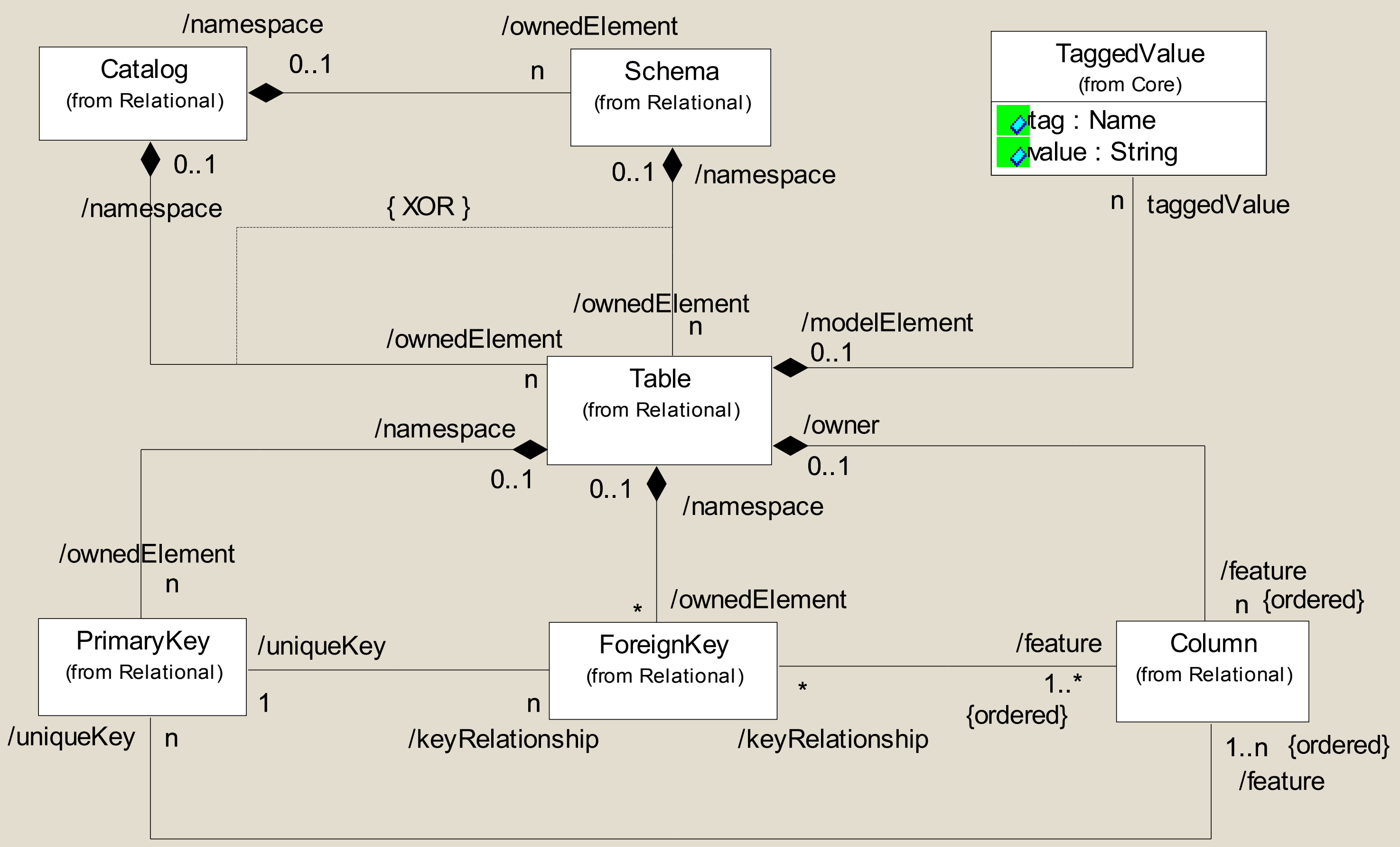
Formal Definition of the MIP Concept

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

A projection has the effect of limiting to domain of discourse to some very specific subset of the underlying metamodel

For example, in the case of CWM and the Star Schema pattern, the CWM projection consists the following subgraph of the CWM UML definition (next slide):

Formal Definition of the MIP Concept



Formal Definition of the MIP Concept

Definition. A *Restriction* is a set of constraints that are imposed on the extent of the metamodel projection, relative to some particular model interchange scenario.

Restrictions have the effect of limiting the number of instances, types, or subclasses, of metamodel classes that may participate in an interchange event

Together, Projection + Restriction(s) have the effect of constraining the structure of any model conforming to them to a precisely specified form – all other aspects of the model are free to vary

Formal Definition of the MIP Concept

Definition. A *Anchor Element* is some metamodel class of the projection whose instances may be used as starting points for the inspection or traversal of a model.

Designation of anchor elements simplify the processing of a model – the software consumer knows where to “start” in its traversal or inspection of what could be a large or very complex model

An anchor element can be regarded as a heuristic or optimization – not exactly a necessary part of the pattern description, but good to have

Formal Definition of the MIP Concept

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

A MIP is always formulated in terms of some some metamodel expressed in some formal language (e.g., ER, OR, UML, CWM, Java, XML Schema, etc.), although the MIP concepts of projection, restriction, and anchor element are completely independent of any particular metamodel or modeling language

Benefits of Meta Data Interchange Patterns

Hyperion

- The intended semantics of any interchange event are precisely specified
- Shared meta data can be described in terms that non-technical domain experts are familiar with
- Pattern-aware consumers have a precise strategy for processing the content of an imported model: Increases reliability of interchange and lowers software construction and integration costs (ROI implications)
- Pattern-aware producers have guidelines for constructing models for export: Lowers software construction and integration costs (ROI, again)

Specification of Meta Data Interchange Patterns

Hyperion

- Now that we know what they are, and (more-or-less) how to formulate them, how do we describe them? How do we publish these descriptions?
- Human-readable meta data interchange pattern description: Facilitates pattern communication between humans (domain experts, programmers, software architects)
- Software-consumable (or machine-readable) meta data interchange: Facilitates pattern communication between automated processes (*pattern-aware, meta data-driven software*)

Human-Readable Pattern Specification

- Based on use of a standard *pattern template*
- Borrows largely from GOF-style and other software design patterns community practices for describing patterns
- Defines new specification artifacts that are particular to meta data and model interchange
- Emphasis on Web-based publication and search: Internet-based *pattern catalogs* and *pattern repositories*
- Community / contributor-orientation: Domain experts are primary authors of patterns, rather than technologists

MIP Template Elements

Hyperion

- Name and Version
- Intent
- Universal Resource Identifier (URI)
- Contributor
- Problem / Forces / Solution
- Classification: *Domain*, *Macro*, *Micro*
- Category: *Interchange*, *Mapping*, *Generation*, etc.
- Projection / Restriction / Parameters
- Related Patterns
- Example

Online MIP Catalog

- Collection of useful MIP specifications
- Segmented / organized by application domain
- Community / contributor-based
- Entries consist of completed MIP templates
- Search by Problem, Classification, Category
- Case study: Hyperion MIP Catalog on HDN [3]
  - Home page: http://dev.hyperion.com/download/sample_applications/cwm_mip_catalog.cfm
  - Template, Sample MIPs, and coding example

Software-Consumable Pattern Specification

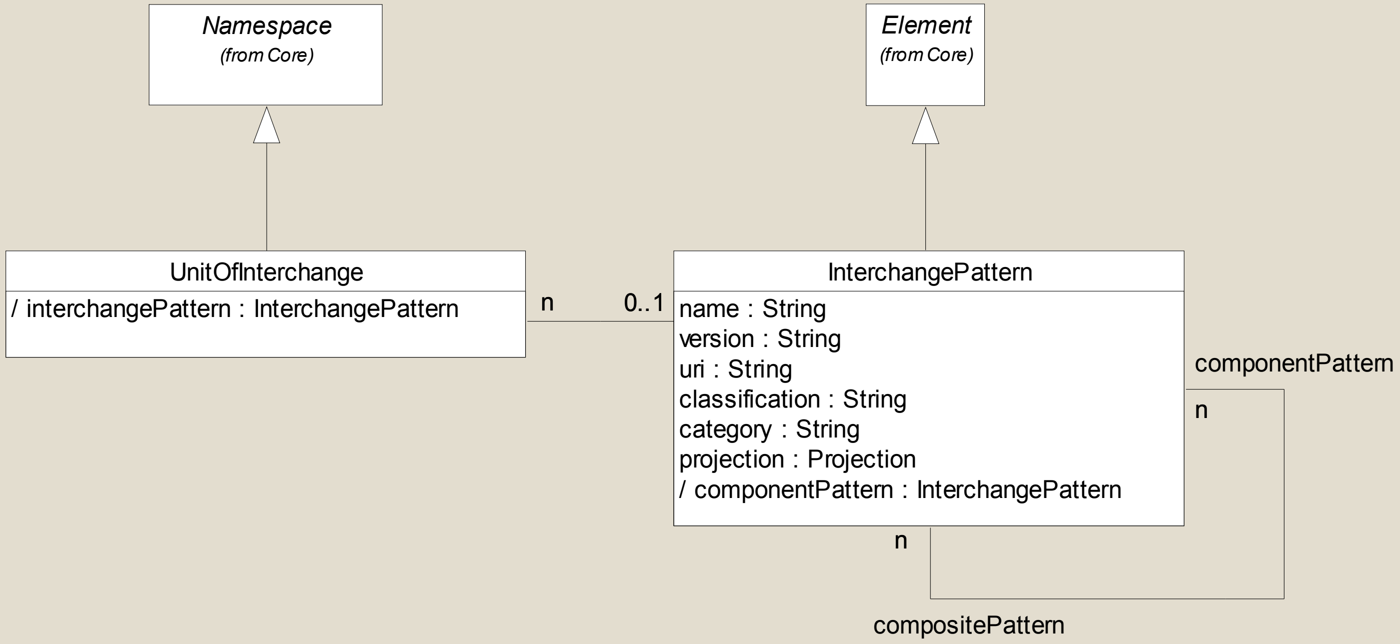
- Essentially an automated representation of the human-readable MIP specification template
- Enables MIP definitions and associated meta data to be interchanged between pattern-aware software tools and stored in repositories
- Next generation visual modeling tools for MIP development become possible
- Next generation pattern-aware / pattern-driven software becomes possible
- Exposes symmetry between traditional *import-export* and *request-response* styles of meta data interchange

OMG's CWM Metadata Interchange Pattern (CWM MIP) Specification

- A new OMG standard metamodel, the CWM MIP Model; developed by Hyperion, Oracle, Unisys [9]
- A non-intrusive, parallel metamodel to CWM
- Provides the means for describing MIPs based on CWM
- Provides the means for interchange of MIP definitions along with corresponding CWM meta data
- Recently finalized by OMG: Version 1.0 final adoption vote took place in February 2004.

CWM MIP Model Overview

CWM MIP Model – High Level classes:

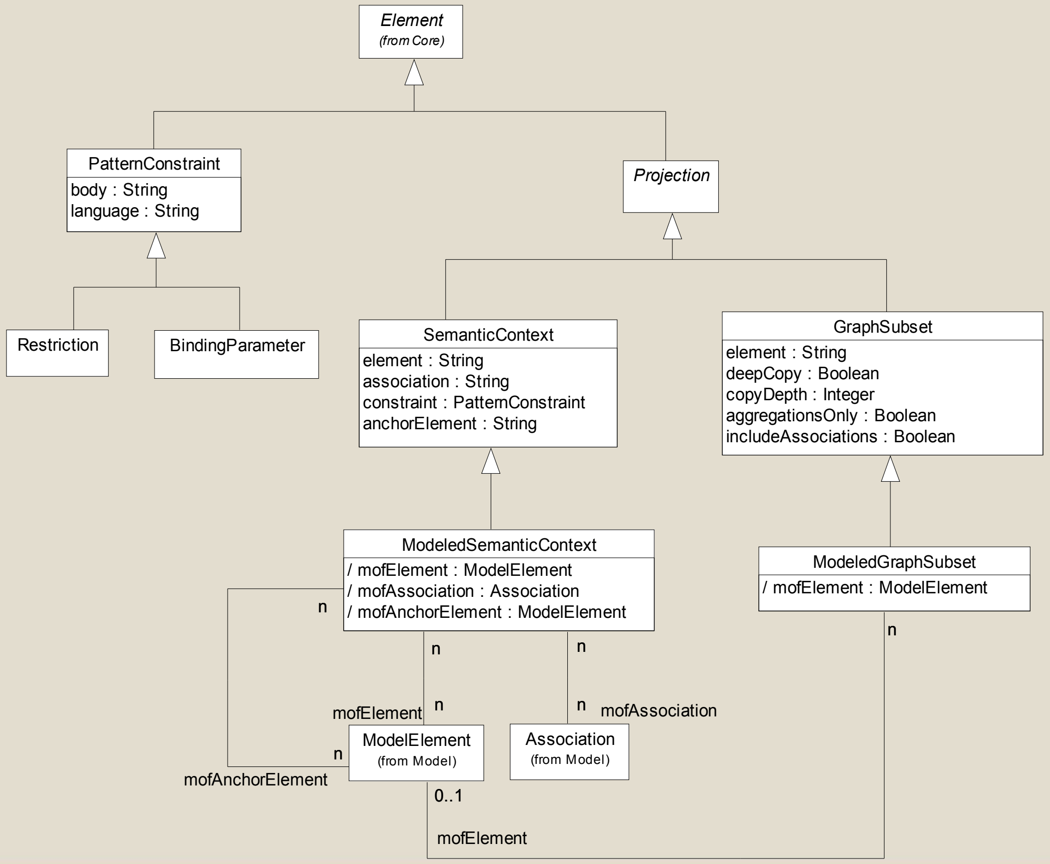


CWM MIP Model – High Level Classes

- *Unit Of Interchange* class: “Logical container” for interchanged meta data
- *Interchange Pattern* class: Models the major MIP components described in the MIP template; can be used to “describe” the content of a Unit Of Interchange
- Possible to interchange and persist MIP instances and MIP definitions independently, or together
- Possible to model arbitrarily complex, composite patterns

CWM MIP Model Overview

CWM MIP Model – Types:



CWM MIP Model – Types

- *Projection* class: Models the MIP concept of “projection” (that is, metamodel subgraph)
- *Pattern Constraint* class: Models the MIP concepts of “restriction” and “parameters”
- Semantic Context subclass: Projection based on enumerated modeling element logical names
- Graph Subset subclass: Projection based on physical sub-graph expression
- “Modeled” projection subclasses: Allow for explicitly modeled projection based on MOF instances

CWM MIP Model – Usage Scenario #1

- Pattern authoring tool creates an Interchange Pattern object
- Defines a MIP by assigning relevant values to object attributes
- Stores Interchange Pattern object in pattern repository or serializes to an XMI document
- MIP has been defined independently of any potentially associated meta data

CWM MIP Model – Usage Scenario #2

- Pattern-aware client locates and imports a MIP object of interest from a pattern repository or XMI document
- Client uses MIP as the basis for automatically expanding its own meta data import capabilities – It has “learned” a new pattern
- Client is now capable of consuming and making intelligent use of meta data conforming to the MIP
- An example of a dynamic, *pattern-driven* tool

CWM MIP Model – Usage Scenario #3

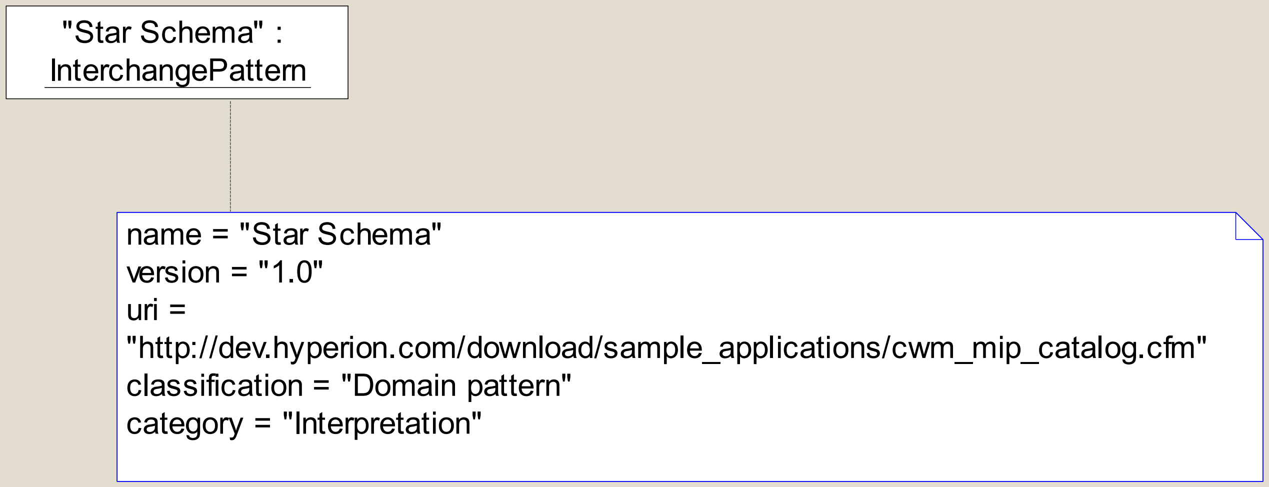
- Pattern-aware producer constructs an Interchange Pattern object defining some MIP
- Producer then constructs three different application models (meta data instances), each represented by its own Unit Of Interchange object
- Producer relates each application model to the common descriptive MIP by creating links based on the Unit Of Interchange – Interchange Pattern association
- Producer serializes this entire object structure to a single XMI document and exports it to the environment

CWM MIP Model – Simple Case Study

- Demonstrate using a Java implementation (based on the JMI specification) of CWM and the CWM MIP Model in creating both meta data and a descriptive MIP
- Use the Star Schema model presented earlier
- Serialize both meta data and MIP to an XMI document
- Persist both meta data and MIP in a pattern-oriented meta data repository
- Case study code available from the Hyperion MIP Catalog on HDN [3]
  - Home page: http://dev.hyperion.com/download/sample_applications/cwm_mip_catalog.cfm
  - Template, Sample MIPs, and coding example

CWM MIP Model – Simple Case Study

Create the Interchange Pattern object:


"Star Schema" : InterchangePattern

name = "Star Schema"
version = "1.0"
uri =
"http://dev.hyperion.com/download/sample_applications/cwm_mip_catalog.cfm"
classification = "Domain pattern"
category = "Interpretation"

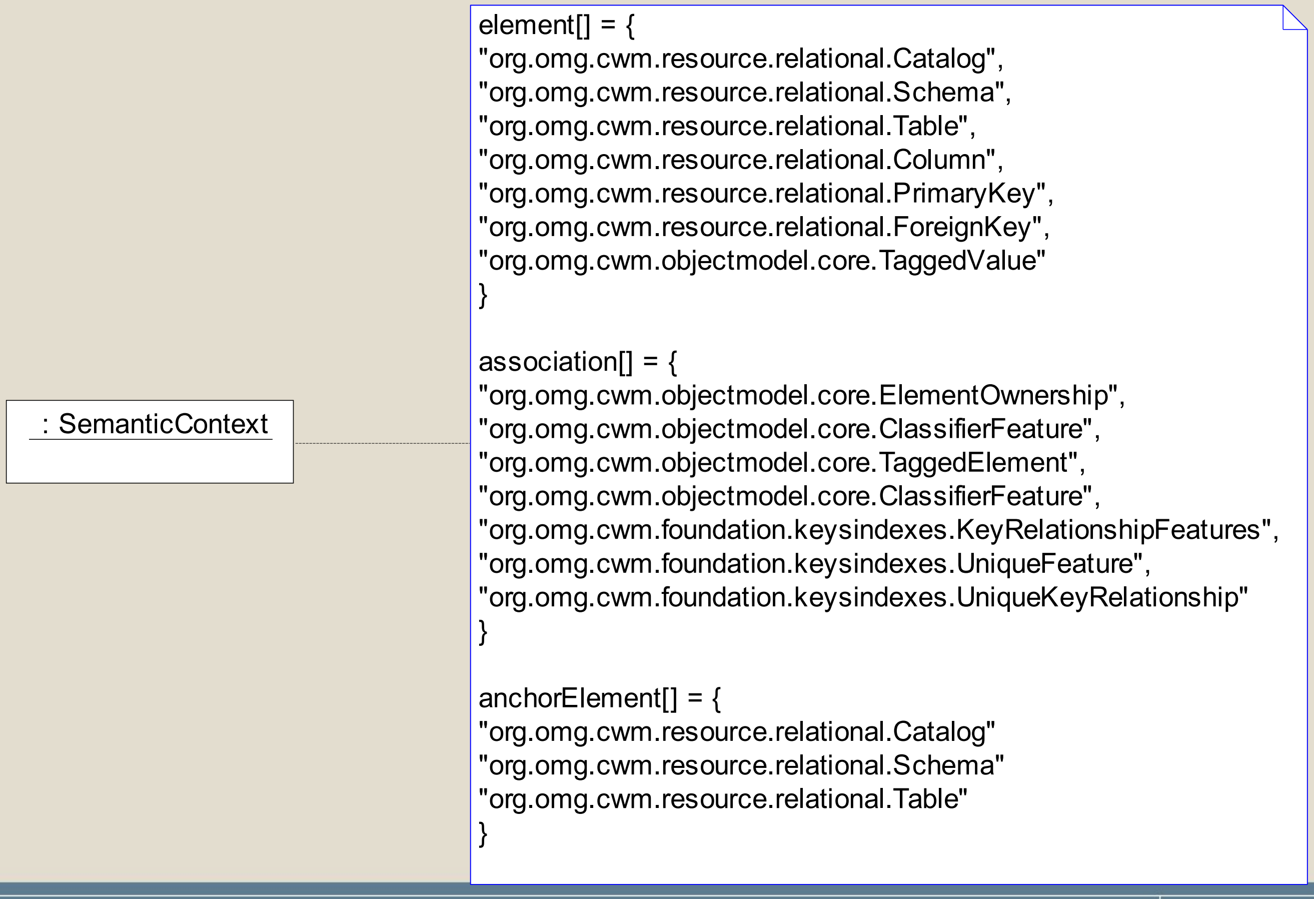
CWM MIP Model – Simple Case Study

Create the Interchange Pattern object:

```
CwmmipPackage cwmmipPkg = cwmmipCompPkg.getCwmmip();
InterchangePatternClass ipClass = cwmmipPkg.getInterchangePattern();
InterchangePattern pattern = ipClass.createInterchangePattern();
pattern.setName( "Star Schema" );
pattern.setVersion( "1.0" );
pattern.setUri("http://dev.hyperion.com/download/sample_applications/
cwm_mip_catalog.cfm" );
pattern.setClassification( "Domain pattern" );
pattern.setCategory( "Interpretation" );
```

CWM MIP Model – Simple Case Study

Create a Projection object:



```
element[] = {
"org.omg.cwm.resource.relational.Catalog",
"org.omg.cwm.resource.relational.Schema",
"org.omg.cwm.resource.relational.Table",
"org.omg.cwm.resource.relational.Column",
"org.omg.cwm.resource.relational.PrimaryKey",
"org.omg.cwm.resource.relational.ForeignKey",
"org.omg.cwm.objectmodel.core.TaggedValue"
}

association[] = {
"org.omg.cwm.objectmodel.core.ElementOwnership",
"org.omg.cwm.objectmodel.core.ClassifierFeature",
"org.omg.cwm.objectmodel.core.TaggedElement",
"org.omg.cwm.objectmodel.core.ClassifierFeature",
"org.omg.cwm.foundation.keysindexes.KeyRelationshipFeatures",
"org.omg.cwm.foundation.keysindexes.UniqueFeature",
"org.omg.cwm.foundation.keysindexes.UniqueKeyRelationship"
}

anchorElement[] = {
"org.omg.cwm.resource.relational.Catalog"
"org.omg.cwm.resource.relational.Schema"
"org.omg.cwm.resource.relational.Table"
}
```

CWM MIP Model – Simple Case Study

Create a Projection object:

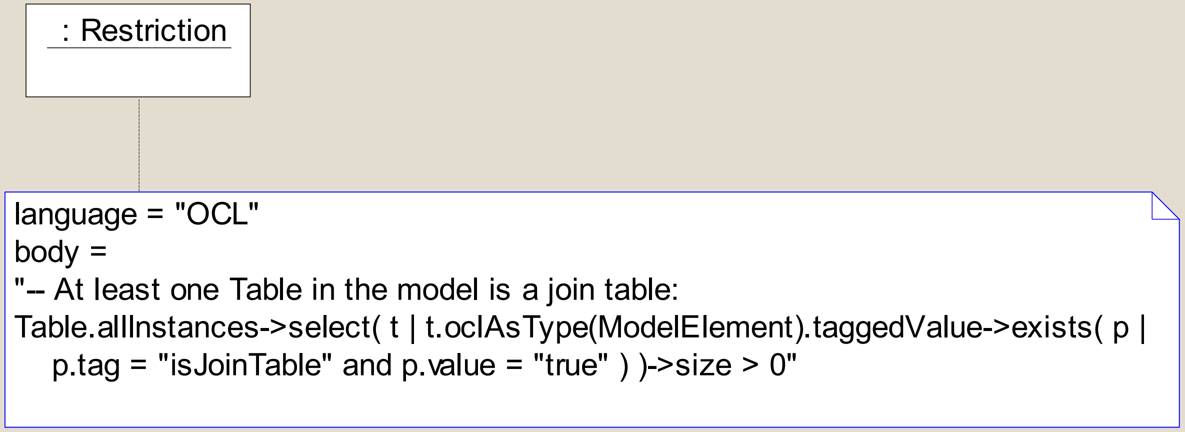
```
SemanticContextClass contextClass = cwmmipPkg.getSemanticContext();
SemanticContext context = contextClass.createSemanticContext();
context.getElement().add( "org.omg.cwm.resource.relational.Catalog" );
context.getElement().add( "org.omg.cwm.resource.relational.Schema" );
context.getElement().add( "org.omg.cwm.resource.relational.Table" );
context.getElement().add( "org.omg.cwm.resource.relational.Column" );

…
context.getAssociation().add( "org.omg.cwm.objectmodel.core.ElementOwnership" );
context.getAssociation().add( "org.omg.cwm.objectmodel.core.ClassifierFeature" );

…
context.getAnchorElement().add( "org.omg.cwm.resource.relational.Catalog" );
context.getAnchorElement().add( "org.omg.cwm.resource.relational.Schema" );
context.getAnchorElement().add( "org.omg.cwm.resource.relational.Table" );
```

CWM MIP Model – Simple Case Study

- Create the Restriction objects (note that we show only one of them here – there are actually four)
- Restrictions are expressed in the Object Constraint Language (OCL) [18], which is a part of the UML



```
language = "OCL"
body =
"-- At least one Table in the model is a join table:
Table.allInstances->select( t | t.oclAsType(ModelElement).taggedValue->exists( p |
    p.tag = "isJoinTable" and p.value = "true" ) )->size > 0"
```

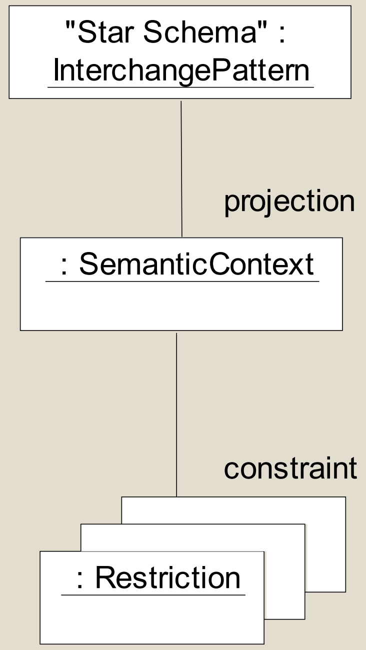
CWM MIP Model – Simple Case Study

Create Restriction objects:

```
RestrictionClass restrictionClass = cwmmipPkg.getRestriction();
Restriction res1 = restrictionClass.createRestriction();
res1.setLanguage( "OCL" );
res1.setBody( "-- At least one Table in the model is a join table: " +
"Table.allInstances->select(t | t.oclAsType(ModelElement).taggedValue" +
"->exists(p | p.tag = \"isJoinTable\" and p.value = \"true\" ))->size > 0" );
```

CWM MIP Model – Simple Case Study

Now, complete the MIP model by linking the various model components created in the previous steps:



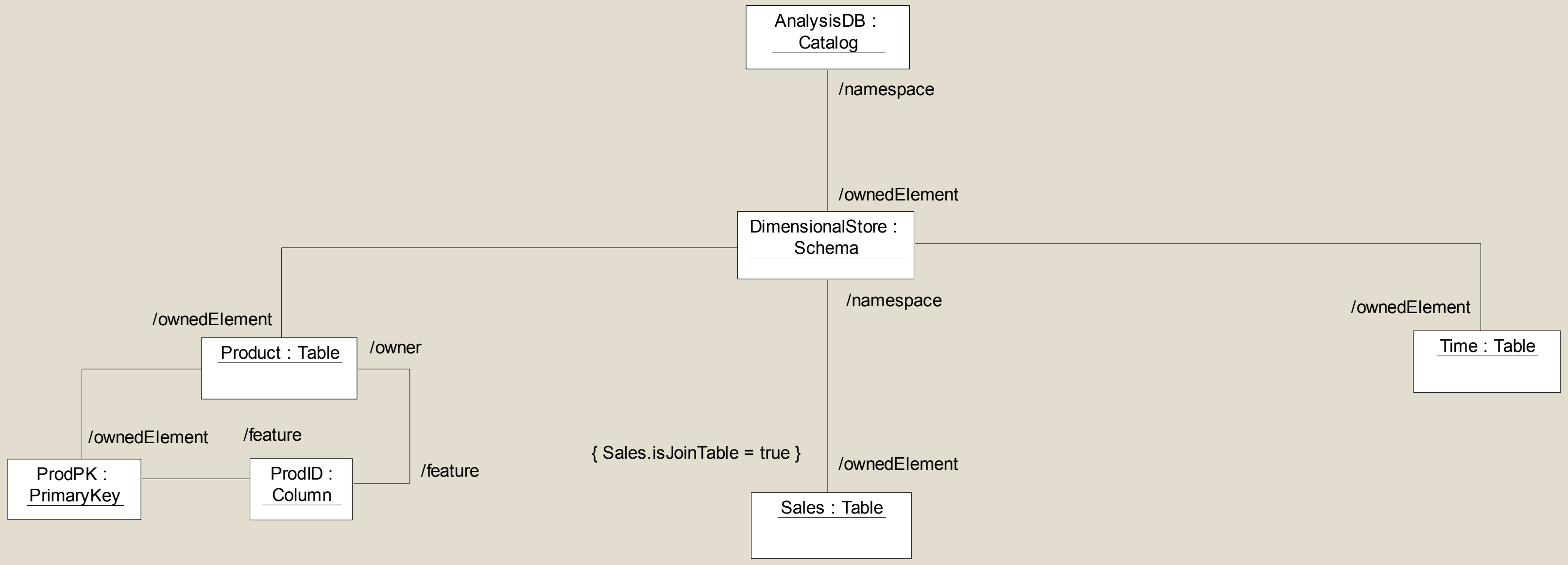
CWM MIP Model – Simple Case Study

Now, complete the MIP model by linking the various model components created in the previous steps:

```
pattern.getProjection().add( context );
context.getConstraint().add( res1 );
…
context.getConstraint().add( res4 );
xmiWriter.write( new FileOutputStream( "StarSchema_v1.0.xmi" ), cwmmipPkg, "1.1" );
```

CWM MIP Model – Simple Case Study

Create an instance of the CWM Relational metamodel representing the Star Schema design presented earlier (this is only a partial illustration):



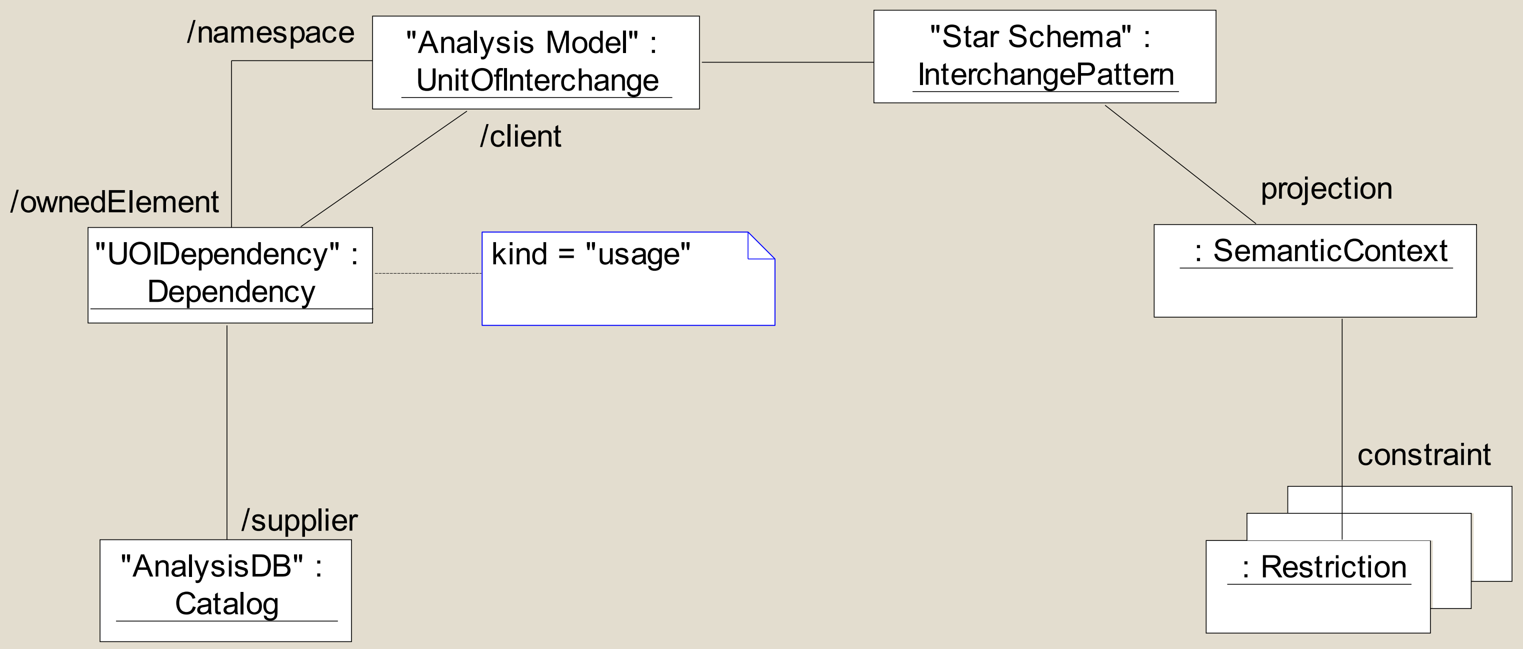
CWM MIP Model – Simple Case Study

Create an instance of the CWM Relational metamodel representing the Star Schema design (partial):

```
RelationalPackage relPkg = cwmCompPkg.getRelational();
CatalogClass catalogClass = relPkg.getCatalog();
org.omg.java.cwm.resource.relational.SchemaClass schemaClass = relPkg.getSchema();
TableClass tableClass = relPkg.getTable();
ColumnClass columnClass = relPkg.getColumn();
Catalog catalog = catalogClass.createCatalog();
catalog.setName( "AnalysisDB" );
org.omg.java.cwm.resource.relational.Schema schema = schemaClass.createSchema();
schema.setName( "DimensionalStore" );
Table productTable = tableClass.createTable();
productTable.setName( "Product" );
Column prodID = columnClass.createColumn();
prodID.setName( "ProdID" );
productTable.getFeature().add( prodID );
prodID.setOwner( productTable );
```

CWM MIP Model – Simple Case Study

Now, complete the entire information model by linking the MIP definition to the Star Schema model:



CWM MIP Model – Simple Case Study

Now, complete the entire information model by linking the MIP definition to the Star Schema model:

```
UnitOfInterchangeClass uoiClass = cwmmipPkg.getUnitOfInterchange();
UnitOfInterchange uoi = uoiClass.createUnitOfInterchange();
uoi.setName( "Analysis Model" );
uoi.setInterchangePattern( pattern );
DependencyClass depClass = corePkg.getDependency();
Dependency uoiDep = depClass.createDependency();
uoiDep.setName( "UOIDependency" );
uoiDep.setKind( "Usage" );
uoiDep.getClient().add( uoi );
uoi.getClientDependency().add( uoiDep );
uoiDep.getSupplier().add( catalog );
Collection objects = new HashSet();
objects.add( pattern ); objects.add( uoi ); objects.add( catalog );
objects.add( uoiDep );
xmiWriter.write( new FileOutputStream( "StarSchemaModel.xmi"), objects, "1.1" );
```

Summary

Hyperion

- Meta data interchange is a pre-requisite for integration of most heterogeneous software systems, products, tools and applications
- Meta data interchange requires a common language (metamodel) and interchange format or interface
- Having a common language and interchange format is necessary, but not sufficient, for truly reliable meta data interchange, especially in highly collaborative environments featuring many meta data producers and consumers with diverse requirements
- Reliable and precise meta data interchange requires some means of establishing *bounded variation* on model structure and content

Summary

- By “bounded variation”, we mean that models must somehow conform to fairly general, predictable structures, or general forms
- Within the boundaries of these general forms, model composition is allowed to vary without restriction
- Bounded variation is best described in terms of idiomatic usages of the underlying, formal language
- A Meta Data Interchange Pattern (MIP) is simply a formally described idiom (or composition of idioms)

Summary

- The use of MIPs simplifies software construction and facilitates the development of pattern-based, model-driven systems
- Such model-driven systems based on the use of model patterns can exhibit a very high degree of operational autonomy and flexibility
- MIPs are specified using *templates* largely based on the software design patterns community concept of a template
- MIPs may be organized and published in terms of pattern catalogs

Summary

- An approach to automating the process of MIP publishing and communication involves the use of a machine-readable model of a MIP
- Such a machine-readable MIP model has been recently formulated as an OMG standard: The CWM Meta Data Interchange Pattern (CWM MIP) Specification
- CWM MIP enables CWM models and their corresponding descriptive MIPs to be persisted together in *pattern-based meta data repositories* (based on MOF) and serialized together in XMI documents

Some Directions for Future Research

- Alignment of CWM MIP with the OMG’s forthcoming “2.0-generation” modeling standards: UML 2.0, MOF 2.0, MOF 2.0 QVT.
- Integration of CWM MIP with several, model-based, open-source Java™ IDEs: Eclipse 3.0 + Eclipse Modeling Framework (EMF), and Sun’s NetBeans IDE
- Ongoing collection, codification, and publishing of useful MIPs on Hyperion Developer Network
- Ongoing development and experimentation with pattern-based and pattern-driven tools (e.g., possible extensions to MDA modeling tools)

Bibliography

1. Gamma, Erich, Richard Helm, Ralph Johnson, John Vlissides. “Design Patterns: Abstraction and Reuse of Object-Oriented Designs.” *Proceedings, ECOOP ’93*. Heidelberg: Springer-Verlag, 1993.

2. Gamma, Erich, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison Wesley Longman, Inc., 1995.

3. Hyperion Developer Network. *Hyperion Metadata Interchange Patterns Catalog*. Hyperion Solutions Corp., 2004. Internet: http://dev.hyperion.com/download/sample_applications/cwm_mip_catalog.cfm/.

4. Java Community Process. *Java™ Metadata Interface Specification, Version 1.0*. Sun Microsystems, 2002. Internet: http://jcp.org/.

5. Kimball, Ralph. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. New York: John Wiley & Sons, Inc., 1996.

6. Kleppe, Anneke, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture – Practice and Promise*. Reading, MA: Addison Wesley Longman, Inc., 2003.

7. OMG. *Common Warehouse Metamodel Specification, Version 1.1*. Needham, MA: Object Management Group, 2003. Internet: http://www.omg.org/technology/documents/formal/cwm.htm.

8. OMG. CORBA, XML and XMI Resource Page. Needham, MA: Object Management Group, 2004. Internet: http://www.omg.org/technology/xml/index.htm

9. 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.

10. OMG. Data Warehousing, CWM and MOF Resource Page. Needham, MA: Object Management Group, 2004. Internet: http://www.omg.org/technology/cwm

11. OMG. Model-Driven Architecture Home Page. Needham, MA: Object Management Group, 2004. Internet: http://www.omg.org/mda

12. OMG. Unified Modeling Language Resource Page. Needham, MA: Object Management Group, 2004. Internet: http://www.omg.org/technology/uml/index.htm

13. 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

Bibliography

14. 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/

15. 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

16. 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

17. 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

18. Warmer, Jos, and Anneke Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Reading, MA: Addison Wesley Longman, Inc., 1999.

Q & A

Hyperion

- John Poole and Dr. Jon Siegel
- Approx 15 minutes

Hyperion®