16TH ICCRTS

Sixteenth International Command and Control Research and Technology Symposium

COLLECTIVE C2 IN MULTINATIONAL CIVIL-MILITARY OPERATIONS

June 21 – 23, 2011

Québec City

Québec Canada

Topic:	Topic 9: Networks and Networking
Title:	**Managing Complex Interoperability Solutions using Model-Driven Architecture**

Authors:

Names:	**Michael Gerz (POC)**
	Nico Bau
Organization:	Fraunhofer FKIE
Address:	Neuenahrer Straße 20
	53343 Wachtberg-Werthhoven, Germany
Phone:	+49 228 9435 414
E-Mail:	{michael.gerz | nico.bau}@fkie.fraunhofer.de

Names:	**Francisco Loaiza**
	Steve Wartik
Organization:	Institute for Defense Analyses
Address:	4850 Mark Center Drive
	Alexandria, VA 22311
E-Mail:	{floaiza | swartik}@ida.org

Managing Complex Interoperability Solutions using Model-Driven Architecture

Michael Gerz, Nico Bau, Francisco Loaiza, Steve Wartik

Abstract

The increasing need to exchange information in joint operations has resulted in interoperability standards of significant complexity. For instance, the *Joint Consultation, Command, and Control Information Exchange Data Model* (*JC3IEDM*) that is part of the MIP interoperability solution has almost doubled in size since 2002.

The configuration management of complex data models, especially in the context of international standardization activities, is a challenging task, because many different artifacts (examples, documentation, etc.) must be kept synchronized as the model evolves.

Ideally, data models are expressed at several levels of abstraction in the form of computation-independent (business) models, platform-independent models (PIM), and platform-specific models (PSM). Efficient transformation tools are desirable to automate the generation of PSMs from a PIM.

Another important aspect concerns the collaboration of different *Communities of Interest* (*COIs*). Diverging information exchange requirements call for a modular or even federated data model rather than a monolithic, indivisible one.

In this paper, we demonstrate how the use of formal languages and the concepts of *Model-Driven Architecture* (*MDA*) can be applied to the JC3IEDM in order to improve the maintenance, consistency, and comprehension of the model and to provide implementers with products that allow them to build interoperability solutions for their C2 systems more easily.

1 Introduction

The increasing need to exchange information in joint operations has resulted in interoperability standards of significant complexity. The configuration management of large data models, especially in the context of international standardization activities, is a challenging task. There are several reasons why:

- *Consistency of Products.* Data models are delivered with several artifacts such as formal definitions, business rules, model diagrams, examples, and free-text documentation. The data model itself is typically expressed at several levels of abstraction in the form of computation-independent (business) models, platform-independent models (PIM), and platform-specific models (PSM). Furthermore, it is ideally complemented by implementation-specific products such as XML schemas, ontologies, database schemas, and source code.

- *Traceability.* Whenever the specification is updated, it is important that all changes can be identified easily by all interested parties. Furthermore, the rationale for each change should be documented to make the new specification more accessible. Unfortunately, known modeling tools do not have an integrated change tracking capability that fulfills the requirements of an international standardization group, such as proposing and voting on changes prior to applying them to the model.
- *Harmonization of Different Communities of Interest (COIs).* Multiple parties are interested in the standardization and would like to see their specific requirements and concerns covered by the specification.

In this paper, we demonstrate how the use of formal languages, management tools, and the concepts of *Model-Driven Architecture* (*MDA*) can be applied to data models in order to improve their maintenance, consistency, and comprehension and to provide implementers with artifacts that allow them to build interoperability solutions for their C2 systems more easily. We will also look briefly at the collaboration of different Communities of Interest (COIs). Diverging information exchange requirements ask for a modular or even federated data model rather than a monolithic and indivisible one.

Case study – MIP JC3IEDM The concepts have been applied to the *Joint Consultation, Command and Control Information Exchange Data Model* (*JC3IEDM*) [2] that is a core product of the *Multilateral Interoperability Programme* (*MIP*) [4]. MIP is an international standardization program consisting of 29 member nations and NATO. It covers operational, procedural, and technical aspects of command and control information exchange. MIP develops specifications with focus on exchange of land C2 information in coalition environments. The latest version of the MIP solution, *MIP Baseline 3*, was released in October 2009.

The JC3IEDM standardizes data elements exchanged by C2 systems. Over the past five years, the Multilateral Interoperability Programme has been exploring the applicability and possible adoption of the MDA framework to support the development and maintenance of the next generation of information systems. As part of the transition to a future product line, the JC3IEDM has been converted from a platform-specific entity-relationship model to a platform-independent class model in the *Unified Modeling Language* (*UML*) [9].

Table of Contents This paper is structured as follows: Section 2 gives a brief introduction to the concepts of model-driven architecture. Section 3 describes the *Query/View/Transformation* (*QVT*) technology and its realization in a MIP-developed prototype tool. QVT is a standard promulgated by the *Object Management Group* (*OMG*) [8] that promotes the development of complete and unambiguous models that can be kept consistent and up to date with acceptable effort levels. The transformation of the JC3IEDM towards a platform-independent model is sketched in section 4. Section 5 outlines the benefits of using formal languages and tools for keeping data model artifacts consistent. Section 6 addresses aspects of change management in a standardization process. Finally, different approaches to the collaboration of communities of interest are discussed in section 7. The paper concludes with a short summary in Section 8.

About the Authors The *Institute for Defense Analyses* (*IDA*) provides technical support to the US delegation that participates in the Multilateral Interoperability Programme. In that capacity, IDA has been exploring the applicability of the MDA framework to the MIP activities. The *Fraunhofer FKIE* supports the German delegation of the MIP. The research institute has been involved in the transformation of the JC3IEDM and has contributed several tools for data model management.

2 Model-Driven Architecture

One of the biggest problems in C2 system development, deployment, and use is the difficulty of maintaining and working with the many different models created and used by architects, analysts, engineers, and end users in defining, validating, and implementing them.

Computer-Aided Software Engineering (CASE) tool vendors promote their products as a way to eliminate or substantially reduce model inconsistency. In the past, many CASE tools let a user create a high-level model and then transform it into a lower-level model. These CASE tools, although useful, tend to be brittle. Most only support a reduced number of transformations, e.g., from an RDBMS physical schema to the corresponding SQL script needed to instantiate the tables in a relational database engine.

As a result, communities end up investing substantial resources to develop special purpose tools to fill the CASE tool gaps. This approach is suboptimal and explains in large part the two conditions that plague modern information systems:

1. Lack of interoperability, caused by the difficulty of transforming the information of one system's model into that of another.
2. Incompleteness, ambiguity, obsolescence, and inconsistency of the underlying models and their documentation during the system's life cycle.

OMG promotes the *Model Driven Architecture* (*MDA*) technology as a means to alleviate many of the issues mentioned above. In MDA, a model has a rigorous enough definition to allow its mechanical transformation into another model. MDA envisions a product's life cycle as a series of transformations between increasingly detailed, rigorous models, until finally one generates models detailed enough to implement as hardware or source code.

MDA is an approach to system and software development that recognizes the central importance of models. In engineering, a model serves several purposes:

1. It presents an abstract view of a complex system or of a complex information structure. The abstract view lets a user concentrate on certain fundamental properties while ignoring details that are unnecessary to gain an understanding of those properties. Consider a software system written in an object-oriented programming language such as Java

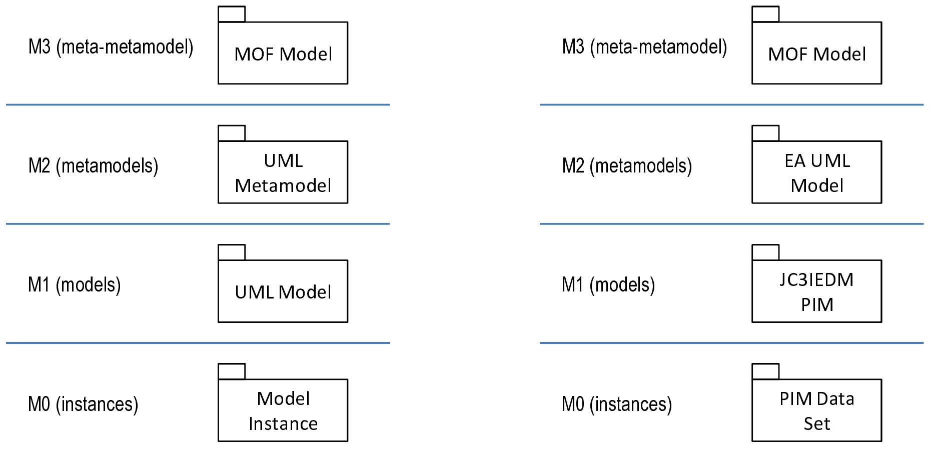


Figure 1: MOF layers

or C++. A user who is interested in understanding the class inheritance may want to view the system using a UML class diagram, even though the class diagram omits the algorithmic details.

2. It permits a limited but important degree of formal analysis. Continuing the example from the previous paragraph, a UML class diagram can be analyzed for violations of rules on inheritance (e.g., no multiple inheritance in Java, and no circularity). The amount of formal analysis is model-specific and generally limited in comparison to results obtained by analyzing, testing, and observing an actual system. However, model analysis helps uncover problems early, at the time they are introduced, when fixing them is cheapest [1].

3. It supports, within the context of software development, the automated transformation of higher-level models into lower-level implementation models. Any code written in a computing language is in fact a model of algorithmic computation that a compiler transforms into a virtual machine language (in case of Java) or a computer's native instruction set (C++).

This paradigm encourages both the up-front creation of good models and the maintenance of downstream consistency. If model A can be mechanically transformed into model B, then in general it is possible to check whether modifications to B have made it inconsistent with A. If B is not consistent with A, it is possible to identify exactly where B differs. This does not guarantee that anyone will fix the offending model, but it does help everyone determine, automatically, why some component is not behaving as expected.

The MDA framework uses the *Meta Object Facility* (*MOF*) [5] to define the conversion of UML models to other UML models. As originally conceived and published in MOF 1.4, there are four layers (see figure 1).

In MOF, the superstructure is defined in terms of layer M3. That is, the UML concepts of *class*, *property*, and *association* are instances of the MOF class *Class*.

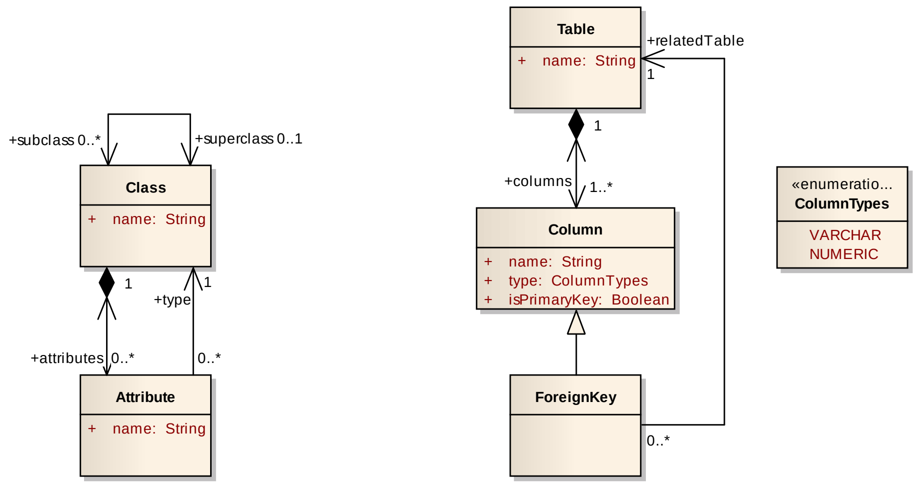


Figure 2: MOF Level 2 Metamodels for a UML Class Diagram (left) and its Corresponding Database Representation (right)

Layer M1 is an instance of a meta-model — what an analyst creates using a UML modeling tool. Layer M0 is an instance of an M1-layer model. In the case of software, the M0 layer would consist of the elements a program manipulates at runtime. The right of figure 1 shows how MOF is used by MIP to express the JC3IEDM. Layer M2 is the version of UML supported by UML tool *Enterprise Architect*[1]. Layer M1 is the JC3IEDM PIM itself. Layer M0 is an instance of a PIM data set, that is, the data a MIP-compliant C2 system collects and reports.

Figure 2 depicts two metamodels that will be used to explain the transformation of a standard UML JC3IEDM class diagram into a model that uses the OMG's database profile. The metamodel on the left allows the instantiation of M1 models that consist of classes and attributes. The classes are related by a single inheritance hierarchy. An attribute has a type, which is a class.

The model on the right is a metamodel for a relational database. It describes tables and columns. A column may be a primary key. It may also be a foreign key, in which case it has an association to the table it relates. A column's type may be either *VARCHAR* or *NUMERIC*.

Figure 3 shows a small subset of instances of the Classes metamodel from figure 2 (left portion). It depicts two JC3IEDM classes (*ObjectItem* and *Person*) and two attributes (*nameText* and *genderCode*). Note that the data type *String* is represented as a class, as is the *GenderCode* enumeration.

Figure 4 is a UML object model. It shows instances of classes in the left-side metamodel from figure 2. That metamodel has two classes, *Class* and *Attribute*. Figure 4 uses these classes to specify two JC3IEDM classes and their properties. Figure 3 shows the instances from figure 4 in the more familiar UML class model form.

[1]see `http://www.sparxsystems.com/`

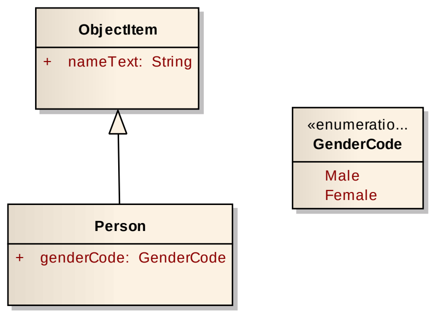


Figure 3: Example of JC3IEDM Instantiation for the MOF Level 2

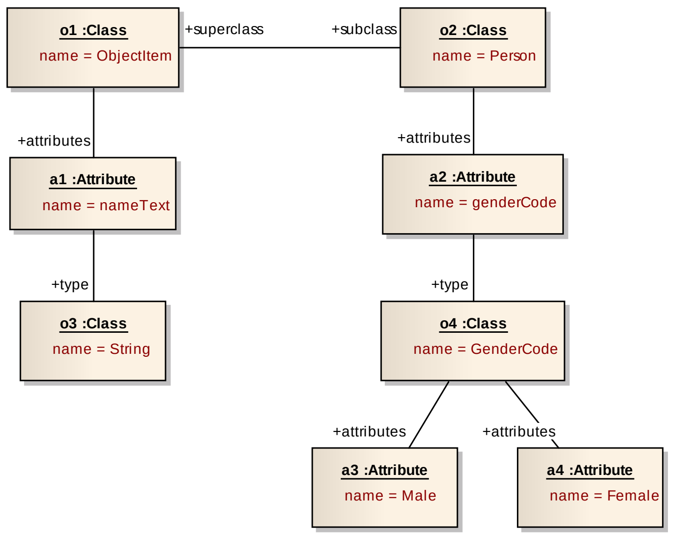


Figure 4: An Object Diagram for the JC3IEDM Classes in Figure 3

3 Automatic Generation of Platform-Specific Models with QVT

OMG has endorsed the *Query/View/Transformation* (*QVT*) [7] technology for describing transformations. Using QVT, one can specify how to transform one model into another. One common use of QVT is to transform a Platform Independent Model (PIM) into a Platform Specific Model (PSM).

A PIM is a model that is independent of the implementation details of a specific platform; a PSM is tied to a particular platform. For example, a C2 information system has an abstract data model, which describes the C2 data the system is prepared to handle. This model is a PIM. The system may want to persist its data using a Relational Database Management System (RDBMS) such as Oracle or MySQL. Each data model for a specific RDBMS is a distinct PSM. Or the system may want to exchange information with other C2 information systems using XML messages. The XML schema for the message structure is another PSM.

Briefly, the approach to using QVT is as follows:

- System architects, analysts, and engineers develop multiple models using UML. These models are at the highest level of abstraction possible such that they can be specified completely and unambiguously. For example, MODAF-compliant[2] users create Operational View (OV) models: interaction models to describe the OV-1 through OV-6 products, and a class model to describe the OV-7 Logical Data Model.
- Engineers develop QVT transformations to related models. On a MODAF-compliant project, they would develop transformations from the OV models to the SV models. For example, an engineer would develop a transformation from OV-7 (Logical Data Model) to SV-11 (Physical Schema).
- System architects and engineers use automated QVT tools to perform the transformations. In other words, the physical schema is automatically generated from the logical data model.
- Engineers develop systems using the transformation results. Inevitably, they find problems that require changes. They use the QVT tools to test whether (and how) these changes affect the source models (i.e., those used as input to transformations). If the transformed model is inconsistent with the source model, system architects and engineers change the source model, then perform the transformation again.

Many languages can be used to describe transformations from one UML model into another, ranging from natural language descriptions to traditional procedural programming languages. QVT provides what is arguably the simplest, most technology-neutral way. QVT, having been designed to perform MDA transformations, contains powerful features lacking in other approaches, and eliminates unnecessary constructs that would otherwise clutter a transformation.

The QVT language states transformations. In the simplest and most common case, a transformation relates two M1-level models. Each model has an M2-level metamodel. The models may share the same metamodel, or they may have different metamodels.

A transformation consists of one or more relations. A relation relates an element from one model's metamodel to an element in the other model's metamodel. Consider the problem of describing how to translate a class model (the PIM) into a database schema (the PSM). Using the metamodels from figure 2, a class-to-schema transformation would contain a relation between *Class* and *Table*. This relation states, in effect, that there is a one-to-one mapping between a class in a class hierarchy and a table in a database.

The transformation would contain a second relation that specifies the one-to-one relationship between an attribute and a column. Because a class and its attributes are related, the class-to-table relation is associated with the attribute-to-column relation.

Some relations describe relationships that are not one-to-one. For example, each table has a primary key. The class model does not usually have key attributes. In other words, there is an element in the database model that is not in the class model.

[2]For more information on the MOD Architecture Framework (MODAF), see `http://www.mod.uk/DefenceInternet/AboutDefence/WhatWeDo/InformationManagement/MODAF/`.

```
transformation Class_DBMS(classModel: Classes, dbmsModel: Tables) {
    top relation Class_Table {
        domain classModel class: Class {} { name <> 'String' };
        enforce domain dbmsModel table: Table {
            name = class.name.toUpper(),
            columns = pk: Column {
                name = class.name.toLower().concat('key'),
                type = ColumnTypes:NUMERIC,
                isPrimaryKey = true
            }
        };
        where { Attribute_Column(class, table); }
    }

    relation Attribute_Column {
        domain classModel class: Class {
            attributes = attr: Attribute {}
        };
        enforce domain dbmsModel table: Table {
            columns = col: Column {
                name = attr.name.toLower(),
                type = if attr.type.name = 'String' or isEnumeration(attr.type.name)
                        then ColumnTypes::VARCHAR
                        else ColumnTypes::NUMERIC
                        endif,
                isPrimaryKey = false
            }
        };
    }

    query isEnumeration(name: String): Boolean {
        let size: Integer = name.size() in name.substring(size−4,size) = 'Code'
    }
}
```

Figure 5: Example of a QVT Script to Transform UML Class Diagrams to its RDBMS Representation

The paragraphs above describe transformation patterns: translate a class to a table; translate an attribute to a column; and so on. QVT is a pattern-based language. Pattern languages are esteemed for providing concise descriptions.

Figure 5 shows a QVT script that expresses these patterns. The script omits some details for the purposes of this example, but it is complete and complex enough to provide the flavor of QVT. The *Class_Table* relation expresses the relationship between a *Class* and a *Table* (note the words at the end of the *domain* lines). Its *where* clause links it to the *Attribute_Column* relation. The *name = class.name.toUpper()* line in the *Class_Table* relation states that the table's name is basically the same as the class's (unlikely in a real transformation). The *columns = pk: Column* line begins a pattern stating that each table has a column that is a primary key. The column's name is the concatenation of the table's lowercased name and the string *key*.

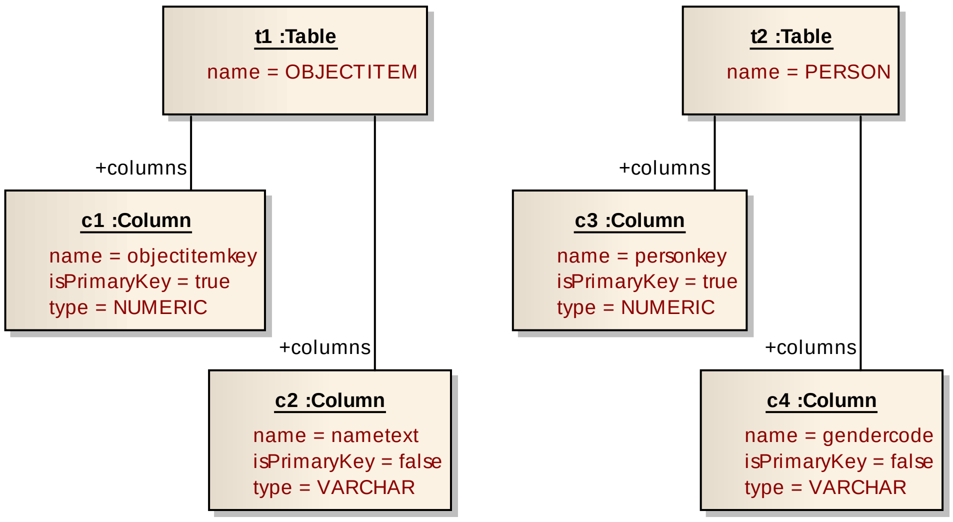


Figure 6: Result of Transformation

The script states how to transform a class model into a database model. More precisely, it states how to transform the instances in figure 4, which are instances of the metamodel in figure 2 (left), into instances of the metamodel in figure 2 (right). It specifies that every instance of *Class* in the figure 2 (left) metamodel maps to an instance of *Table* in the figure 2 (right) metamodel. To avoid creating a table for the primitive *string* data type, the script eliminates the class whose name is *String*.

The script also states that each instance of *Attribute* in the figure 2 (left) metamodel maps to an instance of *Column* in the figure 2 (right) metamodel. The *where* clause relates a class to its attributes and the corresponding table to its columns. Executing the transformation results in creation of the instances shown in figure 6.

The previous paragraphs exemplify the basic principles regarding how one creates instances of one metamodel from instances of another metamodel. Transforming a PIM into any other type of PSM, e.g., to an XML schema representation for the classes, to an OWL ontology, etc., follows essentially the same logic.

The MDA/QVT transformations that have so far been demonstrated, and their roles, are as follows:

- *A transformation from a PIM to a relational database schema.* C2 information needs to be persisted, not just transmitted, and an RDBMS is an efficient mechanism. In the past, systems have maintained JC3IEDM data sets in databases. This transformation, then, provides for C2 data persistence and shows how the JC3IEDM PIM can still be used in conjunction with legacy systems.
- *A transformation from a PIM to a software development kit (SDK).* An SDK is often the most natural way for software developers to access a conceptual data model. The US Army's *C2 Interoperability Group* (*CIG*) built an SDK from an earlier version of the

JC3IEDM. The QVT transformation developed by the IDA study team demonstrates that an SDK can be generated automatically and rapidly from the PIM.

- *A transformation from a PIM to an ontology.* DoD sees ontologies as the best currently available technology to achieve data understanding. A consensus has emerged to write ontologies using the *Web Ontology Language*, known as *OWL*. Transforming a conceptual data model into OWL is effort-intensive. Consequently, many organizations opt to write conceptual data models directly in OWL. This is somewhat unfortunate, as OWL is by design limited in its expressivity — certainly in comparison with UML (for example, OWL cannot express constraints involving arithmetic). The PIM-to-OWL transformation demonstrates that an organization can write a conceptual data model in UML, with all the expressivity that said modeling language supports, and then automatically transform much of the model into OWL. This gives both the power of UML and the interoperability potential of OWL.
- *A transformation from a PIM to an XML Schema Definition (XSD).* Although OWL is the preferred technology for sharing information semantics, it is often unnecessarily complex. OWL-based information exchange can result in both large messages and high computational requirements — especially as compared to raw XML messages. Put another way, OWL specifies semantics, XSD specifies syntax, and sometimes knowing the syntax is all that is necessary. MIP has provided definitions of the JC3IEDM using XSD. The QVT transformations demonstrate that generation of XSD can be automated.

4 A Modular, Platform-Independent JC3IEDM

The data model of MIP baseline 3, the JC3IEDM version 3.0.2, is defined as an Entity-Relationship model in IDEF1X notation. The JC3IEDM has both a logical and a physical view. However, they are structurally identical and only differ with regard to the naming of entities/attributes and a few additional attributes in the physical view.

Weaknesses of the JC3IEDM 3.0.2 The JC3IEDM ER model has some known technical weaknesses:

- The main purpose of the JC3IEDM is to support database replication. Accordingly, the data model includes many database-specific elements. About 40% of all JC3IEDM attributes are primary or foreign key attributes. In addition, *discriminator-code* attributes are defined for every parent entity to denote the type of the subentity.
- The JC3IEDM has the capability to express the logical deletion and update of data. This has led to complicated and ambiguous data structures. Moreover, the built-in "versioning" of data was introduced in an ad hoc and inconsistent manner.
- The use of metadata is inconsistent throughout the data model, again resulting in ambiguous and complicated data structures.

- In the same way, grouping of information is supported only partially and in an inconsistent manner. The problem is partly caused by the fact that the grouping concept and metadata are tightly related to each other in the data model.

The issues listed above have resulted in a series of subtle problems for implementers and operators. For instance, for some information it is impossible to determine whether or not it is part of a specific information group. Therefore, in combination with the MIP data exchange mechanism, information leakage may occur. Moreover, the overall complexity of the data model (in terms of both number of entities and relationships) makes it hard to maintain. Since some concepts have been introduced "when needed" and sometimes even with multiple modeling options, the correct evaluation of JC3IEDM data in a C2 system is algorithmically challenging and requires special business rules. On the other hand, concepts like grouping, metadata, and logical update/delete could be introduced and handled by a pattern-based approach in a generic manner.

Restructuring of the JC3IEDM The MIP has decided to restructure its JC3IEDM for a future baseline and make it a true platform-independent model. In fact, there will be a conceptual, a logical, and a physical data model, where the conceptual model defines the starting point for automatic model transformation, i.e., logical and physical data models are derived from it.

In a first step, the ER model has been transformed into a UML model. In doing so, all database-specific elements (such as keys) have been eliminated and the model has been adapted to general UML design conventions. It has been demonstrated that the database elements can be re-introduced in a platform-specific model, applying the MDA approach (cf. section 3).

In a second step, the cross-cutting concepts (metadata/grouping) have been factored out of the data model "core" and workarounds to support logical update/deletion of data have been removed. These transformations have resulted in a significant reduction of complexity, while preserving the operational concepts of the JC3IEDM.[3]

Key Characteristics of the Conceptual Model The conceptual model describes objects, actions, etc. and their associations as they appear in the real world. It does not describe how objects are perceived and how information about objects, actions, etc. is actually exchanged between C2ISs. This results in two important properties of the conceptual model:

- *"Stateless":* unlike the JC3IEDM 3.0.2, the conceptual model does not describe evolution of objects over time. The model describes objects as such but it does not incorporate the concept of object states, which is introduced in the logical model by transformation. The conceptual model allows characterizing the status of an object but it does not provide data structures to capture the fact that a status is superseded by another status.
- *"Sourceless/contextless":* the conceptual model does not consider conflicting information from different reporters, nor does it reflect the fact that current situation information can

[3]Note: At the time of writing this paper, the technical restructuring has not been completed yet. We expect a stable version by autumn 2011.

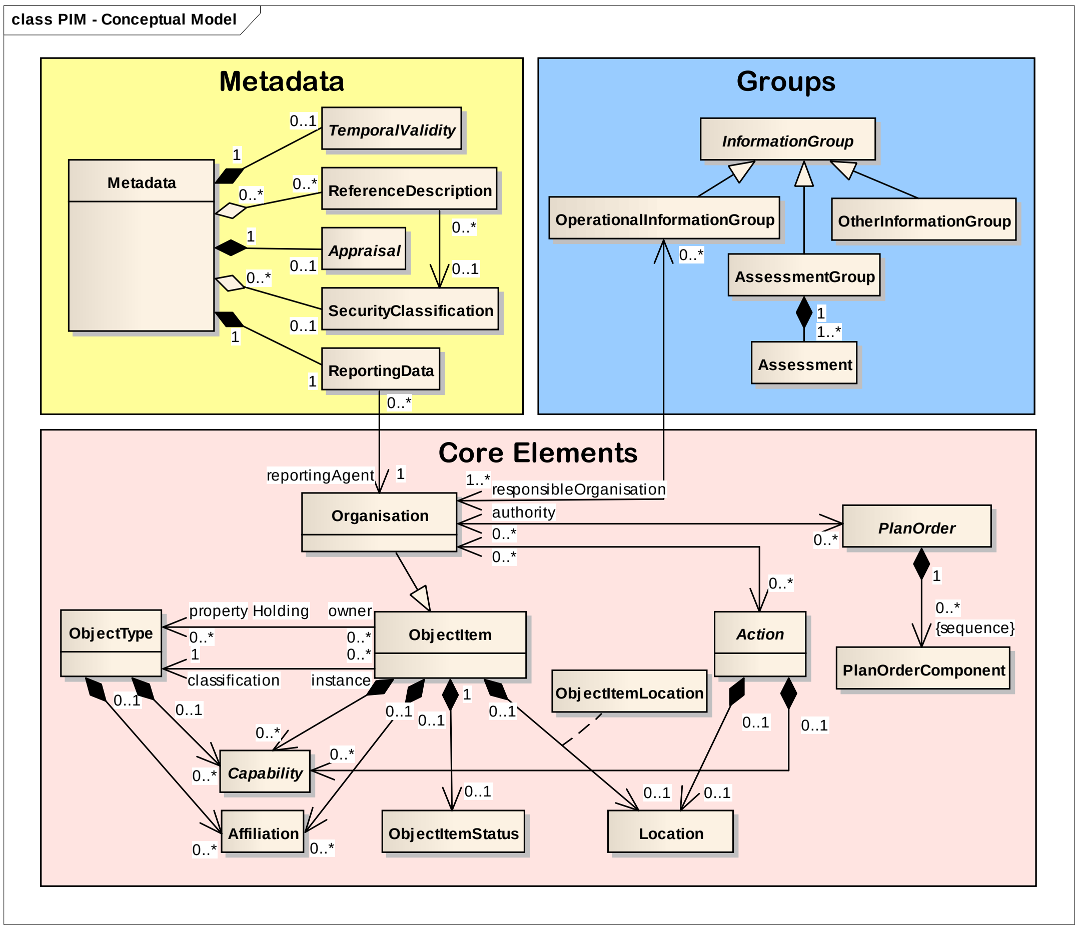


Figure 7: Future JC3IEDM Conceptual Model

coexist with plans. While the conceptual model describes metadata and grouping, it does not provide a mechanism for associating information on objects with metadata/groups. Again, this is introduced in the logical model.

Figure 7 shows the foundational classes and associations of the future conceptual model.

5 Consistency of Artifacts

One of the most important features of a comprehensible specification is accessibility to consistent and concise documentation with formal models and accompanying examples. The JC3IEDM specification of MIP baseline 3 consists of

- An ER model in IDEF1X notation
- A Microsoft Access database describing the IDEF1X model and additional business rules and metadata

- A set of Microsoft Word documents, which include further business rules (in semi-formal and free-text form) and examples on how to use the data model

The difficulty of keeping these artifacts consistent is obvious: While some information is expressed in three different formats, other information is just present in one of the artifacts and there are no tools that provide consistency checks. Thus, all changes to the model in IDEF1X require manual adjustments of the respective information in the Access database as well as in the Word documentation. This process, which includes several quality assurance steps, is extremely time-consuming, effort-intensive, and error-prone.

In the future, the number of manually adjusted products for the JC3IEDM will be reduced, and as much exemplifying documentation as possible will be generated from a single source:

- In the UML version of the JC3IEDM, all business rules have already been added to the model by use of the *Object Constraint Language* (*OCL*) [6]. The availability of formal business rules in OCL means a huge improvement in terms of implementing a business rule checker, because OCL is an established standard and understandable to a much larger base of programmers than the proprietary (or even free-text) format in which MIP baseline 3 defines business rules. Furthermore, OCL allows checking business rules for syntactic and semantic errors and for consistency with the underlying UML model.
- It is also desirable to store example data in a structured way. Since the JC3IEDM is based on UML classes, using UML object definitions for example data is an obvious choice. Depending on the capabilities of the UML modeling tool, consistency between the object models and the class model can be ensured to a certain degree.[4]
- For free-text documentation, it is impossible to ensure consistency with the model automatically. However, if the documentation is annotated in a way that allows a tool to identify references to classes, attributes, and domain values, it is possible to perform an impact analysis and identify those parts of the documentation that need to be reviewed and perhaps updated in response to any specific change to the model. Modern modeling tools have built-in document editing features and allow modelers to introduce hyperlinks among model elements.
- Class diagrams, which illustrate different aspects of the data model (in terms of subviews), need to be updated whenever the model changes. For updates on attributes, UML modeling tools manage to update the affected diagrams automatically but new classes (and associations in some cases) need to be added to class diagrams manually. In the future, it would be beneficial to describe diagrams formally and create them automatically based on their descriptions. For example, a diagram showing a specific class hierarchy should be updated automatically if a new subclass is added to the hierarchy. This approach requires an expressive and user-friendly rule language that allows to specify the classes shown in the diagram based on their associations with other classes rather than by name.

[4]Unfortunately, the tool used by the MIP Community (Enterprise Architect (EA)) only supports limited consistency checks. While EA renames object classifiers in an object diagram if the class name changes, the same does not hold true for renaming attributes in a class. Furthermore, EA does not check if the type of an assigned value of an attribute matches with the specified attribute's type. However, this is not a major concern, since it is possible to write a plugin which performs all those consistency checks.

6 Formal Description of Change Proposals

For a complex specification such as the JC3IEDM, it is vital that all changes to the model are tracked. In a community-driven specification process, all interested parties propose, discuss, and vote on changes prior to applying them to the model and documenting them. In MIP baseline 3, change proposals (CPs) were specified as Word documents, which textually outlined the desired changes and their underlying rationale. Once a CP was accepted, it was manually applied to the various artifacts. In some cases, this required further changes (e.g., generating database keys) which had not been defined in the CP itself.

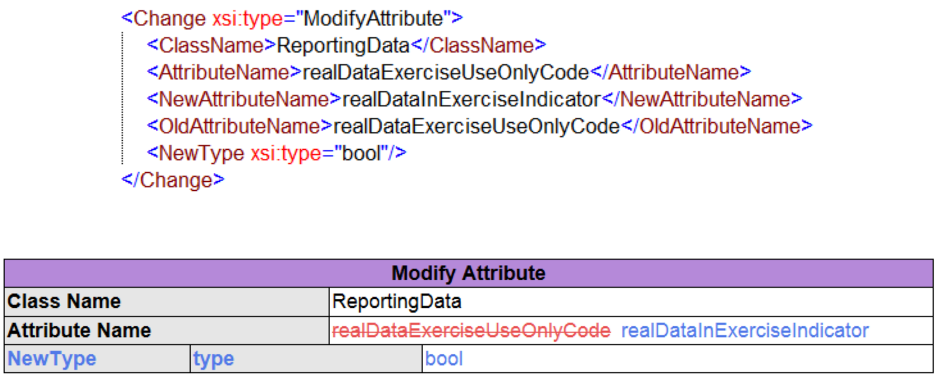
To ensure that accepted changes can be applied to the model without manual intervention, the CPs against the UML version of the JC3IEDM are written as XML files that conform to a specific XML schema. This schema defines a list of operations that can be performed on a UML model, such as adding, removing, or modifying attributes, classes, associations, etc. For each operation, the schema ensures that all information needed to perform the respective change is given. An example of a formal operation is shown in figure 8. Furthermore, the schema requires a CP to include metadata such as rationale and the CP's author(s).

In order to make change proposals more readable, Fraunhofer FKIE has developed a tool that transforms an XML file into an RTF document (see figure 8). Another tool processes an XML CP and applies all listed changes to the UML model. While and after doing so, the tool performs various checks, such as ensuring that the changes do not result in an invalid model. These checks range from generic constraints that are valid for all UML models to very specific tests that are unique to the design patterns of the JC3IEDM. Currently, the following checks have been implemented:

- Each class has at most one direct supertype and no cycle in its type hierarchy[5]
- Each class is connected to at least one other class
- All OCL business rules conform to the underlying UML model
- Naming conventions
  - Class names are written in upper camel case notation (e.g., *ObjectItem*)
  - Attribute names are written in lower camel case notation (e.g., *decoyIndicator*)
- Spurious/unused model elements
  - Enumerations with one or zero attributes (= values) are not allowed
  - Enumerations have to be referenced by at least one attribute

The tool stores the CPs in a specific package in the UML model. Thus, each version of the UML model also includes all CPs that have been applied to it. The CPs are stored as text documents in XML format, so users can search for, e.g., all CPs that modify a specific class.

[5]UML supports multiple inheritance (i.e., one class is the subtype of multiple supertypes). However, multiple inheritance often introduces implementation problems.



```xml
<Change xsi:type="ModifyAttribute">
  <ClassName>ReportingData</ClassName>
  <AttributeName>realDataExerciseUseOnlyCode</AttributeName>
  <NewAttributeName>realDataInExerciseIndicator</NewAttributeName>
  <OldAttributeName>realDataExerciseUseOnlyCode</OldAttributeName>
  <NewType xsi:type="bool"/>
</Change>
```

Modify Attribute		
Class Name	ReportingData	
Attribute Name	~~realDataExerciseUseOnlyCode~~ realDataInExerciseIndicator	
NewType	type	bool

Figure 8: Example of a Formalized Change in XML and RTF

7 Collaboration of Communities of Interest

Several projects beyond MIP make use of the JC3IEDM, e.g., the *Battle Management Language* and the *Joint Dismounted Soldier System*. These *Communities of Interest* (*COIs*) benefit from the international standardization effort in MIP by taking those parts of the specification that are relevant to them and extending the data model in areas where the JC3IEDM is not sufficient for their purposes.

However, this process should not be unidirectional. For the sake of consistency and future reusability of data among all interested parties, it would be highly beneficial to leverage the expertise and work of these COIs to further improve and extend the JC3IEDM. Doing this will require a fine balance between including too much COI-specific information in the JC3IEDM and omitting important general aspects. Too much specific information will make the model convoluted and hard to maintain; missing information that is interesting for multiple COIs may result in different COIs modeling the same aspect in parallel, and inconsistently. In the future, significant care will have to be taken to find and keep this balance.

There has been a long and on-going discussion within NATO, MIP, and on the national level whether a single unified information model is the right answer to tackle interoperability issues or whether a federated model (a hierarchy of domain-specific models) is more appropriate. Presently, NATO is in favor of creating a simple generic model (often referred to as the "common core" or "C2 core"), which can be extended by different COIs in different areas. This approach has two major prerequisites:

- The identification of this common core as a starting point for all modeling activities in all COIs
- An organization structure responsible for supervising all modeling activities and ensuring consistency across the extensions of different COIs

These requirements are necessary to ensure that a) all COIs use the same well-defined core and b) extend the core consistently with regard to their subject matter as well as the common modeling guidelines. It is important to note that these prerequisites are currently not fulfilled.

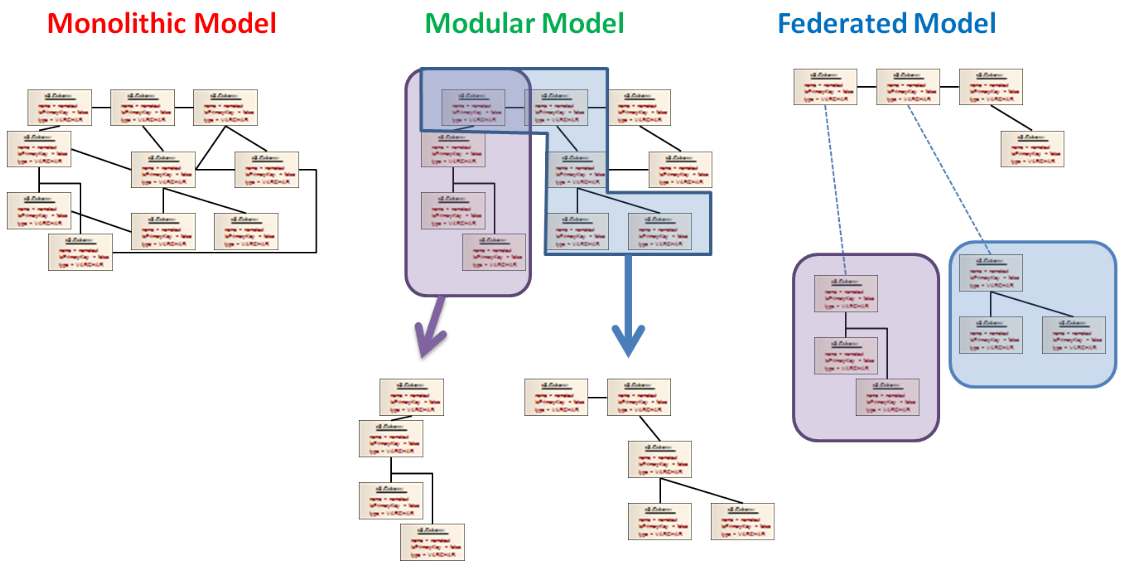


Figure 9: Monolithic vs. Modular vs. Federated Model

Right now, the MIP community follows a modular approach, which could be adapted to a federated modeling approach in the future (see figure 9). Rather than splitting the JC3IEDM into a small core and several COI-specific models (which by itself is a long-term effort and requires a clear understanding of the COIs), the JC3IEDM is kept in its entirety but structured in a way that allows to define submodels in an elegant way. Technically, this is achieved by providing a tool, which allows for different COIs to define their own (semantically meaningful) subset of the JC3IEDM that is created automatically. Then, further modifications, such as additions of missing classes and attributes, can be performed in the de-coupled submodel.

Furthermore, it is possible to perform an impact analysis of MIP change proposals across all COI-specific submodels. Again, the automatic application of model changes, which is already given by the use of formalized CPs, has huge benefits.

Defining proper submodels is considered a long-term process. Later on, it would be possible to identify parts of the JC3IEDM that are only used by one COI, and parts that are shared between different COIs. The latter would be good candidates for a future common core model.

Because MIP grants access to their specifications and tools, other COIs are invited to benefit from the experience of MIP members as well as from a vast toolset. Based on this, a federated modeling approach could be established.

8 Summary

In this paper, we have described various approaches to simplify the management of data models in the context of an international standardization program. As part of the continuous improvement of its interoperability solution, the Multilateral Interoperability Programme has started to convert the JC3IEDM into a platform-independent model in UML, facilitating a high degree of tool automation and product consistency.

As of writing this paper, the JC3IEDM is still undergoing a major revision. An interim version is available at the MIP MDA Website [3] (in particular, see `http://mda.cloudexp.com/DEV/SVN/PIM/trunk`). In addition, various tools have been developed to support the management process. They are publicly available and may be useful for other programs and projects.

References

[1] Boehm, B. W., and Papaccio, P. N.: *Understanding and controlling software costs.* IEEE Transactions on Software Engineering, 14, No. 10, 1988: pp. 1462–1477. 4

[2] MIP: *JC3IEDM Baseline 3.0.2.* October 2009, `http://www.mip-site.org/publicsite/04-Baseline_3.0/JC3IEDM-Joint_C3_Information_Exchange_Data_Model/`. 2

[3] MIP: *MIP MDA unofficial website*, `http://mda.cloudexp.com/`. 17

[4] MIP: *MIP Standard Briefing.* December 2006, `http://www.mip-site.org/publicsite/06-Other_Documents/MSB&MBN-MIPStandardBriefing&MIPBriefingNotes/MSB.ppt`. 2

[5] OMG: *Meta Object Facility (MOF) Core Specification Version 2.0.* January 2006, `http://www.omg.org/spec/MOF/2.0/PDF/`. 4

[6] OMG: *Object Constraint Language Version 2.0.* May 2001, `http://www.omg.org/spec/OCL/2.0/PDF/`. 13

[7] OMG: *QVT 1.1.* January 2011, `http://www.omg.org/spec/QVT/1.1/`. 6

[8] OMG: *OMG Website.* `http://www.omg.org/`. 2

[9] OMG: *UML 2.3.* May 2010, `http://www.omg.org/spec/UML/2.3/`. 2

Nomenclature

CASE	Computer-Aided Software Engineering
COI	Community of Interest
JC3IEDM	Joint Consultation, Command, and Control Information Exchange Data Model
MDA	Model-Driven Architecture
MIP	Multilateral Interoperability Programme
MODAF	MOD Architecture Framework
MOF	Meta Object Facility

OCL	Object Constraint Language
OMG	Object Management Group
PIM	Platform-Independent Model
PSM	Platform-Specific Model
RDBMS	Relational Database Management System
UML	Unified Modeling Language
XSD	XML Schema Definition