Normative Interaction Specifications for C2: A Comprehensive Type of Rule Models for Use in the Model Driven Architecture Framework

Francisco Loaiza, Ph.D., J.D.
Steve Wartik, Ph.D.
Institute for Defense Analyses
floaiza@ida.org / swartik@ida.org

Abstract

Modeling languages such as UML and IDEF1-X provide only partial coverage for the relations and constraints that apply to information within a given domain of interest. In most cases additional textual narratives are required to capture the full set of pertinent business rules. The "Semantics of Business Vocabulary and Business Rules Specification" (SBVR), an OMG adopted specification, offers an alternative to traditional information modeling with vastly more powerful capabilities and the potential for use within the context of the Model Driven Architecture (MDA) framework. This paper presents our recent work done within the Multilateral Interoperability Programme (MIP) where an initial formalization of the model usage and data integrity rules for the Joint Consultation Command and Control Information Exchange Model (JC3IEDM) using the Object Constraint Language (OCL) has been completed. We discuss next the possibility of extending the OCL formalization to FOL-type of rules following the SBVR specifications, and hypothesize how this in turn could be the basis for an all-inclusive NIS, a normative specification of all the relevant rules that control how information interacts within an enterprise. We conclude the paper with a brief discussion on the potential uses of NIS in the context of MDA, as well as the possibility of applying automated theorem proving methods to enhance the quality of the rule models.

1. Introduction

In the area of C2 one of the most mature specifications is the Joint Consultation Command and Control Information Exchange Data Model (JC3IEDM) [1]. This model is in the form of an RDBMS specification and has been created and is maintained using the modeling language IDEF1-X [2]. Except for validation rules for enumerated domains, all its data quality and integrity rules, as well as additional model usage rules, have been until recently expressed only in textual form.[1]

It is a well-known fact that most information modeling languages used to develop databases, e.g., IDEF1-X, UML [3], provide only partial graphical depiction capabilities when it comes to expressing constraints and applicable rules controlling the creation, use and maintenance of the data that is being modeled. Most CASE tools provide a way to document model constraints in the form of textual narratives. One obvious disadvantage herewith is that the content of the rules expressed thusly is not readily machine-processable. Coupled to this is the high degree of ambiguity in natural languages, which cannot be easily removed.

A step in the right direction has been undertaken by the Object Management Group (OMG) with the release in 2006 of the updated specifications of the Object Constraint Language (OCL 2.0) [4]. Statements written in OCL can be linked to any of the objects modeled in a given UML diagram. Because of the formal character of OCL the degree of ambiguity can be substantially reduced or completely eliminated.

Given the above our team undertook as part of the work in support of the U.S. Army as member of the MIP an assessment of the applicability of OCL as a means to capture in a formal way the JC3IEDM business rules. Section 2 below describes how we have proceeded to convert the IDEF1-X specifications of the JC3IEDM into UML. Section 3 discusses how

[1] The current release of the JC3IEDM uses tables in the MIP Information Resource Dictionary (MIRD) database to capture some of the rules.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
21 MAY 2008	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Normative Interaction Specifications for C2: A Comprehensive Type of Rule Models for Use in the Model Driven Architecture Framework	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Institute for Defense Analyses**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
AFCEA-GMU C4I Center Symposium "Critical Issues In C4I" 20-21 May 2008, George Mason University, Fairfax, Virginia Campus, The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**24**	

once the former entities are expressed as UML data classes and class properties the pertinent OCL rules can be generated and linked to them. Section 4 addresses the possibility of leveraging the Semantics of Business Vocabulary and Business Rules (SBVR) specification, which permits the use of first-order logic (FOL) statements using structured English, to expand the scope of the original OCL rule model described in Section 3 into a Normative Interactions Specification (NIS), a normative specification of all the relevant rules that control the information interactions within an enterprise. Section 5 concludes the paper with a discussion of how once all pertinent rules are in the form of FOL statements one can begin to use other techniques such as automated theorem proving to analyze and improve a given NIS by, for example, identifying and eliminating internal inconsistencies that would be very hard to detect by manual inspection.

2. Migration from IDEF1-X to UML

The methodology for transforming into UML an entity-relationship (ER) model written in IDEF1-X is fully documented elsewhere [5]. The process consists of basically four steps:

1. Converting the names of the ER entities and attributes to the commonly used object-oriented naming conventions, i.e., UpperCamel and lowerCamel.
2. Generating a UML class model out of the ER model.[2]
3. Generating the UML profiles needed to recast the original UML classes from step (2) above into a form that removes all the entity-relational baggage contained in the original IDEF1-X specification.
4. Applying the UML profiles from step (3) to the UML classes and verifying the consistency of the results.

3. Generation of the OCL Rules

As noted above IDEF1-X has some inherent modeling limitations. For example, IDEF1-X cannot capture attribute interrelationships. Thus the IDEF1-X specification of JC3IEDM can express the one-to-many relationship between an entity such as ObjectType and ObjectItemType, but cannot restrict the value of airframeDesignCode in the class AircraftType based on the values of the categoryCode.

The JC3IEDM specification makes up for this deficiency by capturing the business rules in the form of textual narratives and tabular expressions contained in annexes. Annex G1 contains model use rules stated using natural language. Annex G2 contains rules that are conveniently expressed as tables. Rules in Annex G2 are also captured in the MIRD. [3]

There are essentially four types of JC3IEDM business rules:

1. Intra-table business rules.
2. Inter-table subtyping consistency business rules.
3. Attribute-specific constraints.
4. Implication constraints.

In the following paragraphs we discuss:

- The nature of rules in each category.
- General approaches for expressing rules of the category in OCL.

The overarching approach to representing rules, common to all categories, is the use of OCL invariants. An invariant states a condition that must always hold for the context in which it exists. The condition is a Boolean expression. Translating a JC3IEDM business rule to OCL is, therefore, a matter of casting that rule as a Boolean expression that uses the operators and functions of OCL.

3.1. Intra-Table Business Rules

The simplest kind of JC3IEDM business rule is the intra-table business rule. The rule ensures data integrity by limiting attribute values to combinations that make sense. A typical example is where the values of a categoryCode in a class restrict the possible values that its subcategoryCode can have. The JC3IEDM applies these kinds of business rules to coded domain attributes. Intra-table business rules appear both in tabular form (Annex G2) and in the MIRD.

An intra-table business rule expresses a constraint that applies to a single table. Furthermore, a constraint violation is detectable and fixable on a per-row basis.

For example, the categoryCode attribute of the class AircraftType has 6 possible values, and the airframeDesignCode attribute has 12 (including the NULL value), so in theory there are 72 possible value combinations. However, a JC3IEDM business rule (Annex G2, Table G2-1) specifies that only 22 of these

[2] This conversion is mechanical, i.e., entities to classes, attributes to class properties, relations to associations. For our work we use the XMI exporting capability offered by ER Modeler 7 (ERwin) from Computer Associates.

[3] The reader is encouraged to visit the URL listed below and download the HTML Browser for the JC3IEDM. The Annexes G1 and G2 have the hyperlinks to all the OCL rules.
https://trac.fkie.fgan.de/JC3XML/wiki

combinations are valid. For example, if the value of the categoryCode is fixed wing, then the airframeDesignCode must take a value that makes sense for a fixed wing aircraft; it cannot be a Balloon or a Helicopter.

3.1.1. Specification in OCL

Formally, each table in Annex G2 that expresses intra-table business rules specifies a disjunction of conjunctions. Each row corresponds to a conjunction. Each conjunction term is an equality or set inclusion specification that denotes a legal set of values for an attribute. For example, Table G2-1 states that:

```
    categoryCode ='Fixed wing' and (airframeDesignCode
  ∈{'Bomber', 'Fighter', 'Glider', 'Transport', 'Not known',
  'Not otherwise specified'} or airframeDesignCode is null)
or    categoryCode ='Lighter than air' and
  (airframeDesignCode ∈{'Balloon', 'Dirigible', 'Not
  otherwise specified'} or airframeDesignCode is null)
or    categoryCode ='Rotary wing' and
  (airframeDesignCode ∈{'Balloon', 'Dirigible', 'Not
  otherwise specified'} or airframeDesignCode is null)
or    categoryCode ='Space vehicle' and
  (airframeDesignCode ∈{'Satellite', 'Not otherwise
  specified'} or airframeDesignCode is null)
or    categoryCode ='Not known' and airframeDesignCode
  is null
or    categoryCode ='Not otherwise specified' and
  airframeDesignCode is null
```

Such an expression maps directly to an OCL expression. The translation is syntactical. It maps:

- A set literal to the OCL Set { ... } operator.
- A set inclusion test to the OCL includes(expr) operator.
- A test for a null attribute to the OCL oclIsUndefined( ) function.

Thus the expression above translates to the following OCL expression:

```
context AircraftType
inv:
  (categoryCode='Fixed wing' and (
    Set {'Bomber', 'Fighter', 'Glider', 'Transport',
        'Not known', 'Not otherwise specified'}
      ->includes(airframeDesignCode)
    or airframeDesignCode.oclIsUndefined()
  ))
  or (categoryCode='Lighter than air' and (
    Set {'Balloon', 'Dirigible', 'Not otherwise specified'}
      ->includes(airframeDesignCode)
    or airframeDesignCode.oclIsUndefined()
  ))
  or (categoryCode='Rotary wing' and (
    Set {'Balloon', 'Dirigible', 'Not otherwise specified'}
      ->includes(airframeDesignCode)
    or airframeDesignCode.oclIsUndefined()
  ))
  or (categoryCode='Space vehicle' and (
    Set {'Satellite', 'Not otherwise specified'}
      ->includes(airframeDesignCode)
    or airframeDesignCode.oclIsUndefined()
  ))
  or (Set {'Not known', 'Not otherwise specified'}
    ->includes(categoryCode) and
    airframeDesignCode.oclIsUndefined())
```

However, this kind of translation is only suited to the smaller tables in Annex G2. It becomes unwieldy on the larger tables, some of which span multiple printed pages. It is also difficult to employ. Assume it is used in an MDA-based approach for the generation of an application such that the application makes a run-time test to ensure that a data set conforms to the invariant. If the test fails, how is the application to determine precisely which clause is violated so as to present the user with a meaningful diagnostic message? (Just listing the entire failed invariant would be unenlightening.) This problem is not intractable, but its solution based on the style described above involves considerable programming effort.

A better approach to OCL translation takes advantage of the analytical effort that has gone into the creation of the tabular representation of these rules as described in Annex G2 of the JC3IEDM documentation. Inspection of said tables shows the following characteristics:

1. Some attributes must always have a value, while others can be NULL. The former can never take the value NULL.
2. An attribute that must have a value always has a single value in each row. An attribute that can be NULL has one or more values, possibly including the NULL value.
3. Each row has a unique combination of attributes that must have a value (i.e., that cannot be NULL).

Attributes that must always have a non-NULL value therefore form the antecedent of an implication, and attributes that can be NULL form the consequent. More precisely, each row of a table in Annex G2 is a single implication whose antecedent is the conjunction of tests that the non-NULL attributes equal the values in their respective columns, and whose consequent is the conjunction of tests that each attribute that can be NULL either has a value from a subset of its possible values or is NULL. Each table row corresponds to one OCL invariant. Table G2-1 of Annex G2 translates to:

```
context AircraftType
inv:
  categoryCode='Fixed wing' implies
  Set {'Bomber', 'Fighter', 'Glider', 'Transport', 'Not known',
    'Not otherwise specified'}
      ->includes(airframeDesignCode)
    or airframeDesignCode.oclIsUndefined()
```

```
inv:
   categoryCode='Lighter than air' implies
      Set {'Balloon', 'Dirigible', 'Not otherwise specified'}
         ->includes(airframeDesignCode)
      or airframeDesignCode.oclIsUndefined()
inv:
   categoryCode='Rotary wing' implies
      Set {'Autogyro', 'Helicopter', 'Transport',
         'Not known', 'Not otherwise specified'}
         ->includes(airframeDesignCode)
      or airframeDesignCode.oclIsUndefined()
inv:
   categoryCode='Space vehicle' implies
      Set {'Satellite', 'Not otherwise specified'}
         ->includes(airframeDesignCode)
      or airframeDesignCode.oclIsUndefined()
inv:
   Set {'Not known', 'Not otherwise specified'}
      ->includes(categoryCode) implies
   airframeDesignCode.oclIsUndefined()
```

Although most applications can ensure data integrity with just the previous set of rules, it may also be important to capture the converse invariants. The following exemplifies the converse invariants:

```
context AircraftType
inv:
   Set {'Bomber', 'Fighter', 'Glider'}
      ->includes(airframeDesignCode)
      implies categoryCode='Fixed wing'
inv:
   Set {'Balloon', 'Dirigible'}->includes(airframeDesignCode)
      implies categoryCode='Lighter than air'
inv:
   Set {'Autogyro', 'Helicopter'}
      ->includes(airframeDesignCode)
      implies categoryCode='Rotary wing'
inv:
   airframeDesignCode='Satellite'
      implies categoryCode='Space vehicle'
inv:
   airframeDesignCode='Transport'
      implies Set{'Fixed wing', Rotary wing'}
         ->includes(categoryCode)
```

The implication form is easier for humans to understand and maintain than is the conjunctive form. This ease extends to users as well as to developers. If an application detects violation of an implication-based invariant, it can present its user with a succinct set of valid choices. By contrast, determining the offending clause of a conjunctive form could entail considerable effort if the conjunction is large.

3.2. Inter-Table Subtyping Consistency Business Rules

An inter-table subtyping consistency business rule is a JC3IEDM rule that concerns two or more attributes in two or more tables. Although this is potentially a broad rule category, the JC3IEDM currently has only one such set of rules. It is presented in Table G2-19. It "enforce[s] the consistency of subtyping" for an ObjectItem and its associated ObjectType. In other words, it ensures that the type of an ObjectItem is sensible. An Airfield should not be typed as an OrganizationType.

There are two possible kinds of association between an ObjectItem and an ObjectType. One denotes the classification of an ObjectItem. The other describes the holdings of an ObjectItem. Although Annex G2 does not say so, Table G2-19 denotes classification, not holdings.

3.2.1. Specification in OCL

Translating a rule from Table G2-19 to OCL is more complex than translating an intra-table business rule. An intra-table business rule focuses on a single class. An inter-table subtyping consistency rule focuses on multiple classes and the relationships between them. Furthermore, the OCL must account for the UML model's class hierarchy. Checking conformance to a rule involves determining the class of an ObjectType instance. Depending on the rule, it may also involve constraints on a categoryCode attribute.

The general form of a business rule from Table G2-19 is an implication:

```
context ObjectItem
inv: self.oclIsKindOf(ObjectItemSubtype)
   implies self.is_classified_as->forAll(ot:ObjectType |
ot.oclIsKindOf(ObjectTypeSubtype))
```

Or, if the business rule involves an attribute test:

```
context ObjectItem
inv: self.oclIsKindOf(ObjectItemSubtype)
   implies self.is_classified_as->forAll( ot: ObjectType |
      ot.oclIsKindOf(ObjectTypeSubtype) and
      ot.oclAsType(ObjectItemSubtype).categoryCode = value
   )
```

For example, the two invariants below state the business rules constraining subtypes of instances of DryDock and Bridge, respectively:

```
context ObjectItem
inv: self.oclIsKindOf(DryDock)
   implies self.is_classified_as->forAll(ot: ObjectType |
      ot.oclIsKindOf(FacilityType) and
      ot.oclAsType(FacilityType).categoryCode = 'Dry-dock'
   )
inv: self.oclIsKindOf(Bridge)
   implies self.is_classified_as->forAll(ot: ObjectType |
      ot.oclIsKindOf(BridgeType) )
```

There is no need to specify a categoryCode value for BridgeType: BridgeType has no categoryCode attribute.

Table G2-19 has several rows that necessitate extending this general form. These rows constrain the consequent category code to a set of values. As with intra-table business rules, sets can be handled through

the OCL includes operator. However, some rows specify the allowed values by stating what values the categoryCode attribute *cannot* have (e.g., the row for Facility). There are two ways to handle these rows. One is by using the expression:

```
not Set { ... } -> includes(categoryCode)
```

where the ellipses represent the forbidden values. The other way is to determine the values the attribute *can* have (by examining all possible values and removing the forbidden ones), then using the expression:

```
Set { ... }->includes(categoryCode)
```

(where the ellipses represent the values determined to be permitted) as is done in OCL invariants above.

Either way is acceptable. It is best to choose the form that uses fewer set literals, because doing so increases an invariant's readability. This is especially true for category codes that have a large range of values.

The inter-table business rules in Table G2-19 can also be associated directly with the class of the antecedent instead of with ObjectItem. That is,

```
context ObjectItem
inv: self.oclIsKindOf(Bridge)
   implies self.is_classified_as->forAll(ot: ObjectType |
      ot.oclIsKindOf(BridgeType) )
```

is equivalent to:

```
context Bridge          <!-- Note the different context -->
inv: self.is_classified_as->forAll(ot: ObjectType |
      ot.oclIsKindOf(BridgeType) )
```

The latter form is a simpler expression, and it associates the business rule directly with the class concerned rather than with an ancestor. Its disadvantage is that it splits the business rules. Annex G2's concise presentation of inter-table subtyping consistency rules in a single place is lost.

In the UML implementation of the JC3IEDM we use the first form, associating all inter-table subtyping consistency business rules with class ObjectItem. The business rules in Table G2-19 were automatically translated to OCL from information in the MIRD, information that was most easily translated to the first form.

It's worth noting that the lost information can be recovered. The simplest way is to name invariants. For example, if all inter-table subtyping consistency rules could be named "itsc", then the above rule would be written as follows:

```
context Bridge
inv itsc: self.is_classified_as->forAll(ot: ObjectType |
            ot.oclIsKindOf(BridgeType) )
```

With this convention, the set of inter-table subtyping consistency rules can be recovered by searching the UML model of the JC3IEDM for all OCL rules named itsc. An automated tool can also analyze each rule, looking for those that fit a general pattern:

```
context C1
inv: self.assoc->forAll(ot: C2 | ot.oclIsKindOf(C2-subtype) )
```

where the italicized items denote replaceable items in the pattern. Implementing this approach, however, is more work.

3.3. Attribute-Specific Constraints

The business rules in Sections 3.1 and 3.2 above are from JC3IEDM Annex G2, which amalgamates business rules for coded domains into tabular forms. The other rules, those listed in Annex G1, express interoperability constraints textually. These constraints do not exhibit patterns as specific as those in Annex G2. Techniques for expressing them in OCL are more ad hoc.

The business rules in Annex G1 can be categorized by whether they apply to a single attribute or to multiple attributes. If they apply to a single attribute, they can be further categorized in ways that help to define approaches for converting them to OCL. This section therefore covers rules that apply to a single attribute. Section 3.3.1.3 discusses the remaining rules from Annex G1, those that do not fit into any pattern-based category.

Some JC3IEDM constraints apply to individual attributes. These constraints may be grouped into the following categories:

1. Domain constraints on types. For example, the IDEF1-X representation of the JC3IEDM states that the attribute egressDirectionAngle of the entity ActionAircraftEmployment is to be represented in a database by the type NUMBER(7,4). It further states that egressDirectionAngle's domain is angle-optional, constraining the attribute's value to at least 0 and less than 360 degrees.
2. Size constraints on associations.[4] For example, Table G1-8 specifies that MaterielType and OrganisationType must have at least one affiliation. This is equivalent to stating situations under which the is_ascribed_to association between ObjectType and Affiliation must have non-zero cardinality.
3. Miscellaneous constraints. These tend to be rules stated textually in Annex G1. An example is rule G1.2.1, which forbids changing category codes in the ObjectType hierarchy.

Note that Annex G does not list all these rules. Annex G is concerned with textually stated rules and with rules on coded domains. Domain constraints on

[4] These are attribute-specific constraints because associations are implemented through so-called migrated or foreign keys in an IDEF1-X model.

types do not fall into either category. They are, however, in the MIRD.

3.3.1. Specification in OCL

The approach to specifying an attribute-specific constraint in OCL depends on its category. This section will discuss each category in turn.

3.3.1.1 Specifying Domain Constraints on Types in OCL

The JC3IEDM specifies 558 domains, of which 533 are application-level domains. Of these 533, 472 are coded domains. Coded domains are expressed in UML as OCL invariants stating that an attribute's value must be drawn from a set of strings (Section 4.3.3). For example, attribute decoyIndicatorCode of entity ObjectType can assume values YES and NO. This translates to the OCL expression:

```
context ObjectItem
inv: Set { 'NO', 'YES' }->includes(self.decoyIndicatorCode)
```

Of the remaining 61 domains, 21 concern JC3IEDM attributes used as database identifiers or primary keys (e.g., action-id). Annex G does not define any rules that reference these domains. The only restrictions on them specify type and length (e.g., action-id is a 64-bit integer whose SQL datatype is NUMBER(20) in Oracle). Because the UML model eliminates the primary keys, these domains are not used in the class model and appear in no OCL expressions.[5]

Of the remaining 40 domains, 19 express numeric boundary constraints. The representation of the domain uses a data type broader than is logical for the domain. (For example, domain temperature is represented as type Real but can never be less than -273.15° Celsius.) Of these, 15 can be grouped into triples in which one member categorizes the triple, one member denotes a mandatory value, and one denotes an optional value (e.g., angle, angle-mandatory, and angle-optional, respectively). No JC3IEDM attribute's domain is a member that characterizes a triple; these domains are simply a grouping mechanism in the IDEF1-X model.

The general form for an OCL rule that expresses a domain constraint on an attribute is a conjunction of inequalities. For example, the constraint for entity FanArea's sectorSizeAngle attribute takes the following form in OCL:

```
context FanArea
inv:  self.sectorSizeAngle >= 0 and
      self.sectorSIzeAngle <= 359.9999
```

The sectorSizeAngle attribute's domain is angle-mandatory. If the domain indicates that the attribute is optional, the invariant must test for that possibility:

```
context ActionAircraftEmployment
inv: self.egressDirectionAngle.oclIsUndefined() or
   (self.egressDirectionAngle >= 0 and
    self.egressDirectionAngle <= 359.9999)
```

It is worth noting that the UML version of the JC3IEDM does not always use OCL to specify domain constraints. JC3IEDM textual domains have a maximum length. This fact could be captured through an OCL invariant, e.g.:

```
context Action
inv: self.nameText.size() <= 50
```

For technical reasons we don't use this approach in the UML version of the JC3IEDM. For example, the OCL invariants generally represent conditions that must be checked dynamically. By contrast, an SQL database schema specifies the maximum length of a text field at the time the database is created.

The decision to specify maximum string length using tags thus reflects the JC3IEDM's heritage. Modern programming languages, as well as knowledge bases, do not generally impose a maximum length on a string, so any length testing must be done dynamically.

3.3.1.2 Size Constraints on Associations

A JC3IEDM size constraint on an association can be expressed in OCL using the built-in size() operator. For example:

```
context MaterielType
inv: is_ascribed_to->size() >= 1
```

requires a MaterielType instance to have at least one affiliation, as per Table G1-8.

Creating these expressions requires examining the JC3IEDM to determine the name of the association. Annex G1 does not specifically state that is_ascribed_to is the association linking MaterielType to Affiliation. However, a quick check of the JC3IEDM reveals is_ascribed_to to be the only logical candidate.

3.3.1.3 Miscellaneous Constraints

Some JC3IEDM constraints on attributes have nothing in common with other constraints. These constraints must be dealt with individually.

The only such business rule that has been identified in Annex G1 is rule G1.2.1, which states that:

> Category codes in the ObjectType hierarchy are not to be changed. New instances of ObjectType must be created with the appropriate category codes in the ObjectType hierarchy.

[5] A domain's type and length (e.g., a 20-digit integer for action-id) must be known when transforming the UML version of the JC3IEDM to a relational database schema. The details are outside the scope of this paper, but it suffices to know that the current approach relies on the JC3IEDM standards for database key representation.

These two sentences impose two distinct business rules. The second sentence is a conjunction of implications: if an instance is of type AirfieldType, BridgeType, HarbourType, or MilitaryObstacleType, then the categoryCode attribute of the FacilityType superclass must be FA; if an instance is of type RouteType, then the categoryCode attribute of the ControlFeatureType superclass must be RTETYP and the categoryCode attribute of the FeatureType superclass must be CF; and so on. As rule G1.2.1 is only stated textually and not in the MIRD, it must be translated manually.

The first sentence is more interesting. It cannot be expressed in the UML version of the JC3IEDM as currently formulated. It deals with a change of state, and the UML representation of the JC3IEDM does not express state. Modeling the first sentence therefore requires making a design decision on how to express state in the converted model. There are two ways:

1. The ObjectType hierarchy of the UML model could be changed such that:
   1. categoryCode elements are query operators rather than attributes.
   2. Classes have constructors that require necessary categoryCode values. The necessary value is that for the class being instantiated, not for any of its superclasses; superclass values can be inferred.

   This approach uses UML class model features, not OCL, to enforce the business rule.
2. The second approach would be to add state models to the UML version of the JC3IEDM. In these models, there would be explicit statements that the value of a categoryCode in the ObjectType hierarchy does not change. These statements can be effected through OCL's @pre construct.

The first approach seems better. It encapsulates the business rule, placing it entirely within the ObjectType hierarchy. The second approach potentially requires restating the business rule each time a new state model is added to the JC3IEDM. The first approach has the disadvantage of modeling one hierarchy inconsistently from all others, using operators instead of attributes.

3.4. Implication Constraints

JC3IEDM Annex G1 contains rules that are similar to stating an implication: The existence in a JC3IEDM data set of some combination of rows, column values, or associations implies the existence of some other combination. These rules can be distinguished from those involving coded domains (Sections 3.1 and 3.2 above) in that, quite simply, they don't necessary involve coded domains. Their antecedents and consequents can be arbitrary Boolean expressions. For example, rule G1.3.1 states that:

> For the instances where the Minefield is a MinefieldLand, then the destructionDateTime is filled only where persistenceCode is "Remote activated destruction" or "timed automatic destruction".

which can be rewritten as the implications:

> If the persistenceCode of a MinefieldLand is neither "Remote activated destruction" nor "timed automatic destruction", then its destructionDateTime is null.
>
> If the persistenceCode of a MinefieldLand is either "Remote activated destruction" or "timed automatic destruction", then its destructionDateTime is not null.

(OCL has no Boolean equivalence operator.)

By definition, an implication constraint fits into the OCL form:

```
context K
inv: antecedent implies consequent
```

Implication constraints have arbitrary expressions in their antecedents and consequents, so no other general rules can be defined to cover how they can be expressed in OCL. That said, several OCL operators appear in many of the invariants. This probably reflects the kinds of business rules that characterize an IEDM, so it is worth briefly discussing these operators.

- The oclIsKindOf() operator is often used to determine whether an instance of some class is also an instance of some subclass. In cases where the operator appears in the antecedent, the use of oclIsKindOf() usually reflects a design decision on how to translate business rules. For instance, rule G1.4.1 contains the invariant:

  ```
  context ObjectItem
  inv: (self.oclIsKindOf(GeographicFeature) or
  self.oclIsKindOf(MeteorologicFeature)) implies
  self.has_affiliation->size() = 0
  ```

  (As noted above, this rule could be rewritten as:

  ```
  context GeographicFeature
  inv: has_affiliation->size() = 0
  context MeteorologicFeature
  inv: has_affiliation->size() = 0
  ```

  The same advantages and disadvantages apply.)
- The forAll iterator is often used to test a condition about an instance's associations. For example, rule G1.4.2.3:

  > The attribute operationalStatusModeCode in MaterielStatus applies only to instances of Materiel that are classified as EquipmentType.

  translates to the OCL expression:

  ```
  context ObjectItem
  inv: not (self.oclIsKindOf(Materiel) and
     self.is_classified_as
     ->forAll(oclIsKindOf(EquipmentType))
  implies self.has->select(oclIsKindOf(ObjectItemStatus))
  ->forAll(
     s | s.oclIsKindOf(MaterielStatus) implies
        s.operationalStatusModeCode.oclIsUndefined()
  )
  ```

In the antecedent, the forAll iterator ensures that instances in question are restricted to Materiel classified as EquipmentType.

- The select iterator is often used to narrow associated instances according to some characteristic. The previous OCL expression filtered instances of ObjectItemStatus to account for the use of multiple associations named has that emanate from ObjectItem.
- The exists iterator appears in expressions both to verify that an instance has one kind of association, and that it has no associations of a particular kind. As an example of the latter rule type, rule G1.10.2a states that:

  If Context B is a sub-context of Context A, then Context A cannot be a sub-context of B.

  This translates to the OCL expression:

```
context Context
  inv: not self.is_the_subject_of->exists (
    c | c.is_the_subject_of->includes(self) )
```

Implication constraint business rules can be simple and short, as are the examples shown in this section. They can also be much larger. Some contain conjunctions and disjunctions of many terms. Others nest iterators to three levels. A few handle subexpressions using let clauses. This complexity is to be expected in formalized complex rules. Its implications for automated translation to an implementation have not been explored by the authors.

4. From OCL to SBVR and NIS

A quick review of the results presented in Section 3 above shows how essentially all the model use and data integrity rules of the JC3IEDM can be expressed in OCL.[6] In addition it shows that these rules are formally equivalent to first order logic (FOL) statements that concern either the behavior of sets produced by set-traversal operators; the values of class properties within a given class; or the values of class properties from different classes in the form of if-then implications.

This is a very important realization because the business rule community has been actively pursuing the standardization of a rule language which is also based on FOL type of statements, namely, SBVR [6]. What makes this development important to C2 information modelers is that this language supports not only all the types of logical operators that OCL has, but also a variety of operators not present in OCL such as deontic and alethic operators.

4.1. Structured English Expressions in SBVR

As described in Annex C of the SBVR specification all business rules are expressible as some kind of logical formulation involving one or more operators. The types of operators are:

- Quantification operators, e.g., each, some, at least one.
- Logical operators, e.g., not, and, or, if-then implication, nand, nor, whether-or-not.
- Modal operators, e.g., is obligatory that, is prohibited that, is necessary that, is impossible that, it is permitted that.

Thus, besides data constraints one can also state doctrinal rules that are now very hard or impossible to express using OCL unless one extends the C2 information models to contain additional UML diagrams.

And while OCL is a very powerful means for capturing constraints, it is not only bound to the classes of a given UML model, but it is also hard to read without special training. In contrast to this, SBVR is designed to express all rules via structured English, a subset of regular English with a controlled vocabulary and syntactic templates, which, unlike OCL, is readily understandable.

This permits subject matter experts to capture rapidly and precisely the operational constraints that may apply for some type of C2 process without a need to have formal training in a modeling language, while at the same time providing in an unambiguous form the C2 information modelers require to derive data classes and other important artifacts.

Rewriting some of Section 3's OCL business rules in SBVR will help demonstrate SBVR's increased clarity. Consider rule G.1.4.1, as expressed by two OCL invariants:

```
context GeographicFeature
inv: has_affilication->size() = 0
context MeteorologicFeature
inv: has_affiliation->size() = 0
```

In SBVR's structured English, these rules are expressed as follows:

Each «GeographicFeature» must not have an «Affiliation».
Each «MeteorologicFeature» must not have an «Affiliation».

The SBVR statements would be understandable to anyone who can read English as restrictions on GeographicFeature and MeteorologicFeature, which as specializations of class ObjectItem inherit an association with class Affiliation.

This seemingly normal English is in fact highly specialized and easily parsed. This specialization is

[6] The only type of rule that is not amenable to capture via OCL is the one that requires the use of operators not available in OCL. In the JC3IEDM this means some rules that require the use of trigonometric functions.

more easily seen by rewriting one of the examples using SBVR's recommendations for font styles:

> Each GeographicFeature must not *have* Affiliations

In this sentence:

- Words in the keyword color are SBVR keywords, with precise meaning, and also with predefined context. In this example:
  - "Each" represents universal quantification over the following thing, here a GeographicFeature.
  - "Must not" introduces an obligation: something that must be true (more precisely, the negation of which must be true) in any acceptable instance of a JC3IEDM data set.
- Underlined words are terms that designate a noun concept drawn from a specified vocabulary. Generally the specified vocabulary maps to the classes and attributes of the model in question, i.e., JC3IEDM class and attribute names.
- Double underlined words represent individual concepts, i.e., instances of a concept.
- *Italicized* words are verbs or verb phrases, and designate fact types. They derive from JC3IEDM association names. They are fact types in the sense that an association instance is a fact, which is typical in FOL.

Therefore, the sentence translates, unambiguously, to the following first-order logic expression:

$$(\forall g \in \text{GeographicFeature})(\neg\exists a \in \text{Affiliation})(\text{has\_affiliation}(g,a))$$

Despite its precise structure, SBVR's syntax is flexible and amenable to variations. The example rule may also be written as:

> It is prohibited that each GeographicFeature *has* Affiliations

because "must not" and "it is prohibited that" express equivalent semantics. This flexibility is not unlimited. The statement:

> Each GeographicFeature never *has* Affiliations

has a different, incorrect meaning. It states that it is not possible to create a JC3IEDM data set in which a GeographicFeature has an association to an Affiliation, as opposed to claiming that the existence of this association makes the data set invalid. This difference arises from how SBVR defines "must not" and "never". Thorough knowledge of SBVR is a prerequisite to writing SBVR rules, even if rules, once written, are fairly readable.

It was mentioned above that noun concepts are drawn from a vocabulary. So are most fact types. A full and precise vocabulary is necessary to parse SBVR sentences. This can be seen by analyzing the simple examples presented so far. How, exactly, is "has" to be interpreted? The answer is that it must be linked to the JC3IEDM has_affiliation association that exists between ObjectItem and Affiliation. That is, in the context of a GeographicFeature, the only verbs and verb phrases that are legal are those built into SBVR (e.g., "must") and those that are defined to derive from the JC3IEDM associations in which ObjectItem participates. To make use of a verb in all possible SBVR contexts, it is generally necessary to specify singular and plural forms (the examples use both has and have), as well as the infinitive. It is also necessary to specify singular and plural forms of concepts. A JC3IEDM vocabulary developed for SBVR would describe "Affiliation" as a concept deriving from the JC3IEDM Affiliation class, and list "Affiliations" as its plural. Note that vocabulary terms need not be identical to their JC3IEDM counterparts in the class model. The noun concept "Geographic Feature" could be used if it is judged to improve readability.

Consider the following example. It states one of the rules from Table G2-1 (see Section 3.1.1) in SBVR:

> If the category code *of* an Aircraft Type *is* Fixed Wing then the airframe design code *is* Bomber or Fighter or Glider or Transport or Not Known or Not Otherwise Specified or *is* undefined.

As noted above, double-underlined words are individual concepts – i.e., instances of a concept. An individual concept is also known as a name.

This example uses an if-then formulation to express a business rule. The verb phrase "of" is used to reference an attribute in its parent class. The verb phrase "is" establishes an equality test between an attribute and a name. These are not inherent SBVR semantics. They must be defined in a vocabulary.

SBVR is smart enough to permit elision, which is very helpful in the repeated equality tests. It's not necessary to write "airframe design code is Bomber or airframe design code is Fighter or …". However, SBVR doesn't understand punctuation, so the following form, though shorter and closer to common English usage, is invalid:

> airframe design code *is* Bomber, Fighter, … or Not Otherwise Specified.

These simple forms and principles let SBVR express surprisingly complex rules. The JC3IEDM business rule contained in G1.4.2.3 (see Section 3.4 above) is:

> If it is not the case that an ObjectItem *is a* Materiel and always *is classified as* an EquipmentType then each ObjectItemStatus *of* the ObjectItem that *is a* MaterielStatus must *have* an optionalStatusModeCode that *is* undefined.

which, especially when rendered without font embellishments, is indisputably less technical than its OCL counterpart, and will be understood by a much broader audience.

SBVR can also be used more effectively than OCL to express business rules that deal with operational use of a JC3IEDM data set (SBVR terms these "operative"

as opposed to "structural" rules). Consider rule G.1.8.1c:

> Some uses of a line entail a preferred side, such as forward line of own troops where the symbology calls for dragon teeth on one side. When side has meaning for a line, the left-hand side is interpreted according to the direction of the line as determined from an ascending numeration of the points of the line….

This rule does not constrain a JC3IEDM data set. It specifies how users are to interpret a data set. It is incompletely stated. For one thing, it never specifies the relationship between left-hand side and direction. An SBVR statement of (a part of) the rule might be:

> If a Line *has* South-to-North direction then a user must *interpret* the left hand side *of* the Line *as* West.

This statement explicitly mentions the user, tying his behavior to operative intent. Unlike an OCL invariant, it instructs how to use a JC3IEDM-based system.

In fact, SBVR was devised more to model these kinds of rules than to model the examples previously given, which are structural. This does not imply that SBVR cannot be used, or even should not be used, to model structural rules. The best modeling language should be chosen based on many factors, and if readability is a prime consideration then SBVR may be preferable to OCL. Even if readability is not paramount, OCL is weaker than SBVR (recall SBVR's distinction between "must" and "always"; OCL offers only "must").

4.2. A Normative Interactions Specification

The term "normative" is understood as *pertaining to giving directives or rules, or prescribing an authoritative standard*. Specifically, in the context of interactions these norms prescribe the expected characteristics and values of the relationships that are binding upon the objects that participate in the interactions. In that respect the norms serve to guide, control, or regulate proper and acceptable behavior.

The term "specification" is understood in the sense of being a complete, precise, and verifiable documentation of the norms applicable to the interactions considered.

For every relationship among components specified in an information model there are one or more applicable norms. These norms state the expected acceptable values that are characteristic of the relationships. For example, the relationship between an information element corresponding to the location of a military unit in the battlefield and the system that generates and broadcasts it to the pertinent C2 nodes can have characteristics such as refresh rate, latency, and validity. In a given information model the norm applicable may state that the refresh rate is "once every five minutes". Similarly, values for each of the other characteristics may be stated in that norm.

From the above it is clear that the NIS underlying an information model provides a sufficient baseline for the derivation of most of the traditional descriptions of the domain being modeled. The difference between them and the NIS is that the former have more specialized perspectives. In other words, the NIS for a given information model is the most complete representation of the domain because it is comprised of all the objects of the domain, as well as all their pertinent interactions, which themselves are fully characterized and may have, where applicable, metrics and expected values. Things such as the information model data dictionary, as well as the data requirements needed to support the interactions can be derived from the NIS because the norms are written in terms of the objects of the domain, and they assert the characteristics expected for the relationships.

5. MDA, Formal Proof Methods, and NIS

Section 4.2 above intimates that a finalized NIS for C2 would constitute a full representation of the rules that govern all the objects in the domain.

It would, therefore, be theoretically possible to use it as a form of extended Platform Independent Model (PIM) as defined in the context of MDA. This is in fact already noted in Annex A of the SBVR specification, although the guidance for how to convert business models written in SBVR into either intermediate PIMs or directly into PSMs is something that will have to be worked out. Nevertheless, it is clear from the substantial progress that has been made in the use of the MDA framework over the past couple of years, that once the required level of specificity is present in a formalized representation it will be feasible to write the needed applications that can transform its content into any desired format.

Similarly, the rapid advances in automated theorem proving[7] suggest that one of the possible benefits of formulating a C2 information model such as the JC3IEDM in the form of a NIS would be that one can apply to it these formal proof techniques to verify that there are no internal inconsistencies or contradictions. At present the review of even the rules that control data integrity in the JC3IEDM is quite a laborious undertaking, and is error-prone due to the size of some of the enumerated domains or the sheer complexity of the relationships. Automation is in this case quite appropriate as a means to minimize human error.

[7] For a recent review of some of the tools available see http://www.dwheeler.com/essays/high-assurance-floss.html

Two approaches to formal, automated proofs based on a JC3IEDM PIM have been explored. The first uses OWL-DL, the description logics variant of the Web Ontology Language.[8] The second uses prover9, an automated theorem prover for first-order logic.[9] These approaches are now briefly discussed.

In both approaches, the underlying idea is that most of the JC3IEDM's business rules can be expressed using set-theoretic operators. Section 4.1 gave an example of how a business rule could be stated using FOL notation:

$$(\forall g \in \text{GeographicFeature})(\neg\exists a \in \text{Affiliation})(\text{has\_affiliation}(g,a))$$

In general this kind of formulation is possible for any OCL rule. The rule thus defines a constraint on a set, in the above rule the set of all things that are geographic features.

Suppose someone (erroneously) defines the following business rule elsewhere in the JCIEDM:

$$(\forall g \in \text{GeographicFeature})(\exists a \in \text{Affiliation})(\text{has\_affiliation}(g,a))$$

Together with the last rule, the implication is that a GeographicFeature both must and must not have an Affiliation. This is a logical contradiction. Nothing can be a GeographicFeature; or, in UML terminology, the class GeographicFeature cannot be instantiated. Although UML doesn't explicitly forbid creating such a class, it's of no use in a real-world model, and ought to be a clue that something is wrong.

The automated proof approach identifies these logical contradictions. An OWL-DL model can be used as input to a reasoner, such as Pellet.[10] The reasoner is requested to "classify" the model. The result of the operation is (among other things) a list of classes that can have no members. This achieves the goal of identifying logical inconsistencies in the business rules.

Description logics is being promoted as the best kind of logic to use in semantic web technologies. Description logics reasoning is fast and predictable, a huge advantage when dealing with large, distributed models. It appears that description logics can be used to evaluate the potential for interoperability between two models. Contradictions would imply that that interoperability cannot occur.

However, description logics is not as expressive as first-order logic. It deliberately omits some first-order logic operators. Practically speaking, these omissions are not critical. Most JC3IEDM business rules can be expressed using description logics. Nevertheless, it remains true that first-order logic is required to provide a full, formal specification of the JC3IEDM.

The limitations of description logics are by design. Many first-order logic problems require non-polynomial time to solve. First-order logic is not usually considered applicable to real-time or near-real-time problem-solving. This limits its utility in a net-centric environment.

Its slowness notwithstanding, first-order logic has an important role to play. Each new release of a model like the JC3IEDM incorporates changes that might make it self-contradictory. There is a strong need for validation of the model. Validation is not a net-centric exercise. It can be conducted over a prolonged period of time (it certainly is currently, considering that it's done manually). In such circumstances it makes sense to bring the full expressiveness of first-order logic to bear. Analyzing a FOL representation of the JC3IEDM has the potential to reveal more mistakes than analyzing a OWL-DL representation.

6. Conclusions

The recent work in MIP where the JC3IEDM specifications have been migrated to UML has opened the door to the formalization via OCL of all the current rules controlling the use of the model and the integrity of the data sets.

The development of a more expressive language for capturing business rules, namely, SBVR, suggests that at a minimum the OCL formulation of the rules should be transformed into SBVR structured English, and that potentially all C2 information interactions could be also formally captured to provide a more robust and stable specification from which one can create through appropriate transformations the required PSMs.

As a bonus, with a NIS written in structured English one could also take advantage of some recent development in automated theorem proofing, many of which accept as input FOL statements.

7. References

[1] http://www.mip-site.org/

[2] http://www.idef.com/IDEF1X.html

[3] http://www.omg.org/technology/documents/modeling_spec_catalog.htm

[4] http://www.omg.org/technology/documents/formal/ocl.htm

[5] IDA Paper P-4274 – Loaiza, Wartik, *A Model Driven Architecture Approach for Migrating Information Models from IDEF1-X to UML*. UNCLASS. 2007. An electronic copy of the paper is also available at https://trac.fkie.fgan.de/JC3XML/wiki.

[6] http://www.omg.org/technology/documents/bms_spec_catalog.htm

[8] See http://www.w3.org/2004/OWL/.

[9] See http://www.cs.unm.edu/~mccune/mace4/.

[10] See http://pellet.owldl.org/.

Rule Models as Semantic Models for Command and Control

Francisco Loaiza
Steven Wartik
Institute for Defense Analyses

TOC

INFORMATION TECHNOLOGY & SYSTEMS DIVISION

- Background
- JC3IEDM
- Business Rules In C2
- Beyond OCL
- Conclusions

Background

State of the Art

INFORMATION TECHNOLOGY & SYSTEMS DIVISION

- Most information modeling languages used to develop databases, e.g., IDEF1-X, UML, provide only partial graphical depiction capabilities when it comes to expressing constraints and applicable business rules controlling the creation, use and maintenance of the data that is being modeled
- UML extends its modeling capabilities for constraints and business rules via the Object Constraint Language (OCL)

Consequences for C2

- **To take advantage of OCL the models must be recast in UML**
- **Our Approach**
  - Convert our test C2 Model from IDEF1-X to UML
  - Rewrite 'constraints' and Business Rules as OCL Statements
  - Assess the applicability of more powerful 'rule languages' (e.g., SBVR)

JC3IEDM

What is the JC3IEDM?

JC3IEDM

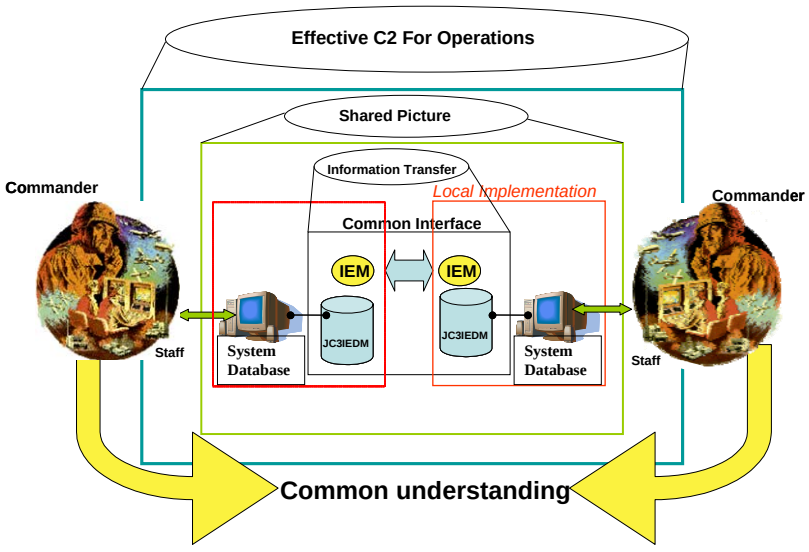
INFORMATION TECHNOLOGY & SYSTEMS DIVISION

The Joint Consultation, Command, and Control Information Exchange Data Model

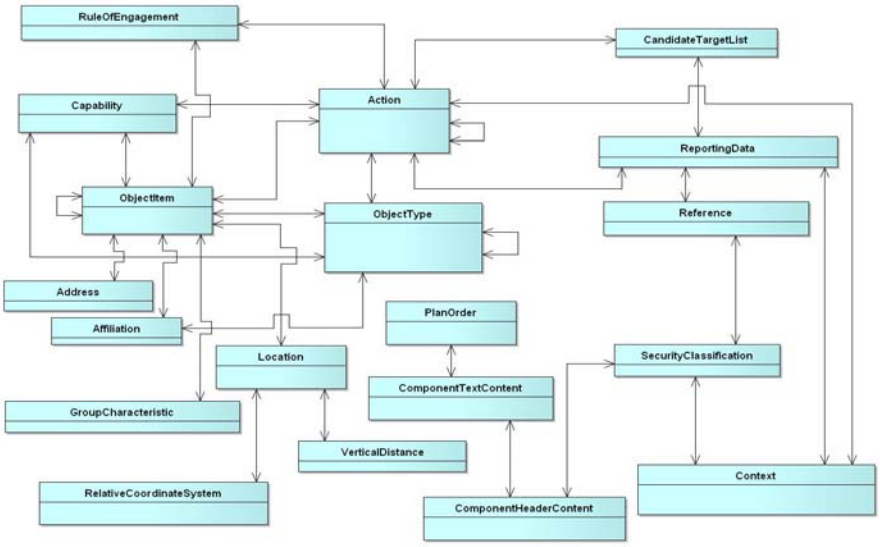
- **Defines the objects in the universe of discourse (**Facilities, Features, Materiel, Organizations, Persons**)**
- **Describes the state of the universe: past, present, and future**
- **Records observed events**
- **Plans to use what you have to achieve objectives**
- **Monitors the execution of planned activity**

JC3IEDM Information Exchange

INFORMATION TECHNOLOGY & SYSTEMS DIVISION



JC3IEDM: Basic Design

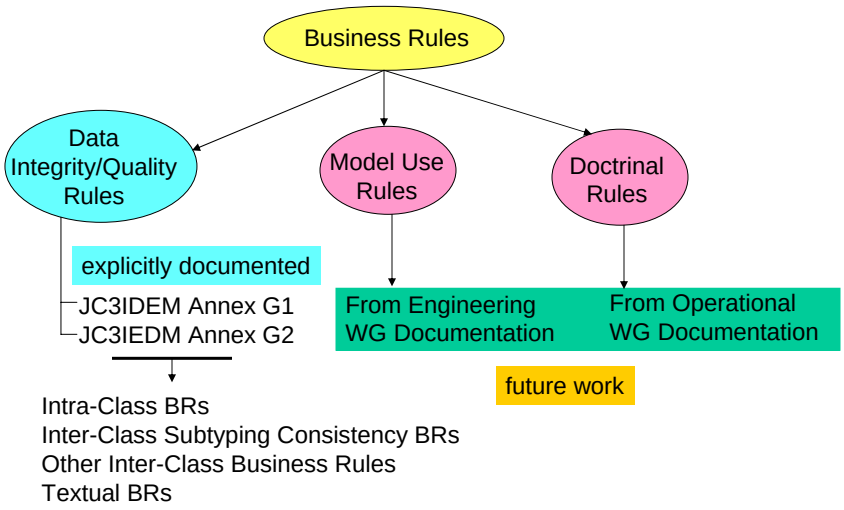


INFORMATION TECHNOLOGY & SYSTEMS DIVISION

Business Rules in C2

Taxonomy of JC3IEDM Business Rules

INFORMATION TECHNOLOGY & SYSTEMS DIVISION



Intra-Class Business Rules

INFORMATION TECHNOLOGY & SYSTEMS DIVISION

When AircraftType.categoryCode = 'Fixed wing' the AircraftType.airframeDesignCode must be a value in the set {'Bomber', 'Fighter', 'Glider', 'Transport', 'Not known', 'Not otherwise specified'} or be NULL

OCL

```
context AircraftType
inv:
        categoryCode='Fixed wing' implies
        Set {'Bomber', 'Fighter', 'Glider', 'Transport', 'Not known',
                'Not otherwise specified'}
                        ->includes(airframeDesignCode)
                or airframeDesignCode.oclIsUndefined()
```

Inter-Class Subtyping Consistency Business Rules

When instances of DryDock, a specialization of ObjectItem, are created, the appropriate corresponding instance of FacilityType, a specialization of ObjectType, must be instantiated with the value of categoryCode set to 'Dry-dock'

OCL

```
context ObjectItem
inv: self.oclIsKindOf(DryDock)
        implies is_classified_as->forAll(ot: ObjectType |
                ot.oclIsKindOf(FacilityType)
                and ot.oclAsType(FacilityType).categoryCode = 'Dry-dock'
        )
```

Other Inter-Class Business Rules

Instances of ControlFeature that constitute elements in the definition of Q-routes entail restrictive associations to conform to the concept of Q-routes.

Subject ControlFeature Typed As	Object ControlFeature Typed As	ObjectItemAssociation:: categoryCode Value	Number of Permissible Associations
Q-route	**Q-zone**	**Is part of**	**1**
Way point	Q-route	Is part of	2 or more

OCL

```
context ObjectItemAssociation
inv:  is_the_subject_of.oclIsKindOf(ControlFeature) and is_the_subject_of.is_classified_as->
        forAll(oclIsKindOf(RouteType) and oclAsType(RouteType).categoryCode = 'QROUTE')
 and is_the_object_of.oclIsKindOf(ControlFeature) and is_the_object_of.is_classified_as->
        forAll(oclIsKindOf(ControlFeatureType)
                and oclAsType(ControlFeatureType).categoryCode = 'QZONE')
   implies categoryCode = 'ISPART' and is_the_subject_of.is_the_object_of->size() = 1
```

Textual Business Rules

INFORMATION TECHNOLOGY & SYSTEMS DIVISION

For the instances where the Minefield is a MinefieldLand, then the destructionDatetime is filled only where persistenceCode is “Remote activated destruction” or “timed automatic destruction”.

OCL

```
context MinefieldLand
inv: Set { ‘Remote activated destruction’, ‘Timed automatic destruction’ }
          ->includes(persistenceCode)
     implies not destructionDatetime.oclIsUndefined()
inv: not Set { ‘Remote activated destruction’, ‘Timed automatic destruction’ }
          ->includes(persistenceCode)
     implies destructionDatetime.oclIsUndefined()
```

Assessment of OCL Capability

INFORMATION TECHNOLOGY & SYSTEMS DIVISION

- **Almost all the use and data integrity rules of the JC3IEDM can be expressed in OCL**
- **The only type of rule that is not amenable to capture via OCL is the one that requires the use of operators not available in OCL. In the JC3IEDM this means those rules that require the use of trigonometric functions**
- **OCL rules are formally equivalent to first order logic (FOL) statements that concern either the behavior of sets produced by set-traversal operators; the values of class properties within a given class; or the values of class properties from different classes in the form of if-then implications.**

Beyond OCL

Can We Do Better?

INFORMATION TECHNOLOGY & SYSTEMS DIVISION

- **Shortcomings of OCL**
  - OCL rules are always written against the classes defined in a specific UML model
  - OCL does not support mathematical operators
  - Not suited to technophobes:
    - Syntax is non-intuitive
    - Somewhat cumbersome

SBVR

INFORMATION TECHNOLOGY & SYSTEMS DIVISION

- **Semantics of Business Vocabulary and Business Rules**
- **Rules can be written in Structured English**
- **Language supports:**
  - Quantification operators, e.g., **each**, **some**, **at least one**
  - Logical operators, e.g., **not**, **and**, **or**, **if-then**, **nand**, **nor**, **whether-or-not**
  - Modal operators, e.g., **is obligatory that**, **is prohibited that**, **is necessary that**, **is impossible that**, **it is permitted that**
- **Models written in SBVR also support MDA approach**
- **SBVR rules are FOL statements**

SBVR as a Superset of OCL(1)

INFORMATION TECHNOLOGY & SYSTEMS DIVISION

OCL

```
context GeographicFeature
inv: has_affilication->size() = 0
context MeteorologicFeature
inv: has_affiliation->size() = 0
```

SBVR

Each «GeographicFeature» must not have an «Affiliation»
Each «MeteorologicFeature» must not have an «Affiliation»

SBVR as a Superset of OCL(2)

When side has meaning for a line, the left-hand side is interpreted according to the direction of the line as determined from an ascending numeration of the points of the line….

SBVR

If a Line *has* South-to-North direction then a user must *interpret* the left hand side *of* the Line *as* West.

SBVR explicitly mentions user and ties his behavior to operative intent. This capability is not supported in OCL.

Normative Interactions Specification (NIS)

- **A complete, precise, and verifiable documentation of the *directives or rules* that prescribe the expected characteristics and values of the relationships that are binding upon the objects that participate in the interactions**
- **In that respect the norms serve to guide, control, or regulate proper and acceptable behavior**

NIS Completeness

INFORMATION TECHNOLOGY & SYSTEMS DIVISION

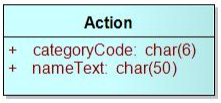
SBVR

Each a Action must *have* a name
The name of an Action must *be written using* ISO-93884 encoding
The name of an Action cannot *exceed* 50 characters
Each a Action must *have* a categoryCode
The categoryCode of an Action cannot *exceed* 6 characters
.
.

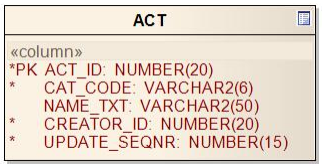
NIS as a PIM

«PIM-to-PSM*»

logical



physical



Automated Consistency Checking

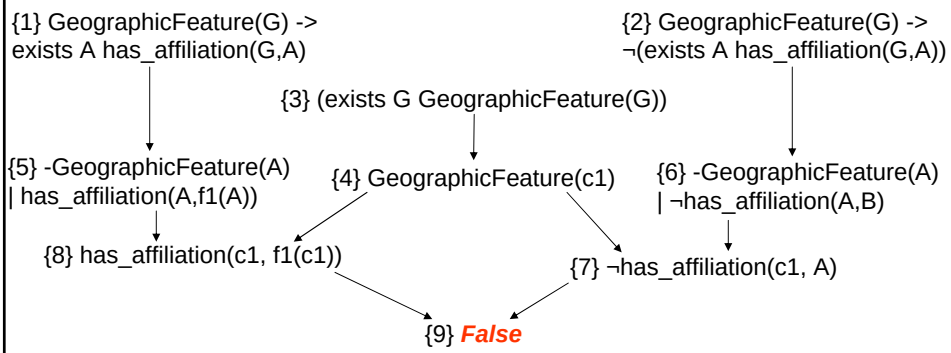
INFORMATION TECHNOLOGY & SYSTEMS DIVISION

Prover9 Input

```
formulas(assumptions).
   GeographicFeature(G) -> exists A has_affiliation(G,A).
   GeographicFeature(G) -> -(exists A has_affiliation(G,A)).
   (exists G GeographicFeature(G)).
end_of_list.
```

Actual rule

Contradiction



Conclusions

- **The recent work in MIP where the JC3IEDM specifications have been migrated to UML has opened the door to the formalization via OCL of all the current rules controlling the use of the model and the integrity of the data sets**
- **The development of a more expressive language for capturing business rules, namely, SBVR, suggests that at a minimum the OCL formulation of the rules should be transformed into SBVR structured English, and that potentially all C2 information interactions could be also formally captured to provide a more robust and stable specification from which one can create through appropriate transformations the required PSMs**
- **As a bonus, with a NIS written in structured English one could also take advantage of some recent development in automated theorem proofing, many of which accept as input FOL statements**