Open Geospatial Consortium

Publication Date: 2015-08-19

Approval Date: 2015-06-05

Submission Date: 2015-05-13

Reference number of this document: OGC 15-024r2

External Identifier for this document is: http://www.opengis.net/doc/PER/tb11-sbvr-guidance

Category: Public Engineering Report

Editor: Johannes Echterhoff

OGC® Testbed 11 Aviation - Guidance on Using Semantics of Business Vocabulary and Business Rules (SBVR) Engineering Report

Copyright © 2015 Open Geospatial Consortium
To obtain additional rights of use, visit http://www.opengeospatial.org/legal/.

Warning

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements.

Document type:	OGC® Engineering Report
Document subtype:	NA
Document stage:	Approved for public release
Document language:	English

Copyright © 2015 Open Geospatial Consortium

License Agreement
Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER’S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR’s sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

i. Abstract

This document is a deliverable of the OGC Testbed 11[1]. It describes the results of developing a tool to automatically derive Schematron code from SBVR constraints. It also documents a vocabulary with a profile of core geospatial terms and concepts, which can be used to express geospatial constraints in business rules.

ii. Keywords

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, aviation, architecture, SBVR, testbed 11

[1] URL of Testbed 11 page on OGC public website is http://www.opengeospatial.org/projects/initiatives/testbed11.

Copyright © 2015 Open Geospatial Consortium

Contents

Page

1 Introduction ... 1
1.1 Scope ... 1
1.2 Document contributor contact points ... 1
1.3 Future work ... 2
1.4 Foreword ... 3
2 References ... 4
3 Terms and definitions ... 5
4 Abbreviated terms ... 6
5 OGC Testbed 11 Guidance Using SBVR - Overview ... 7
6 Geospatial Vocabulary ... 8
6.1 Overview ... 8
6.2 Vocabulary ... 8
6.2.1 Spatial Characteristics ... 8
6.2.2 Other adopted concepts ... 12
6.2.3 Spatial Relationship Operators ... 13
6.2.4 Spatial Analysis Operators ... 14
6.3 Considerations for use in business rules ... 15
6.3.1 Geometry representation ... 15
6.3.2 Spatial operator use ... 16
6.4 Analysis of geometric constraints from the Aerodrome Map Databasse ... 16
7 Guidance for SBVR Rule Development ... 22
7.1 Identify Schema Level that SBVR Rule is written for ... 22

7.2 Be aware of implications when writing rules involving (dynamic) AIXM feature types 22

8 Automated Derivation of SBVR Business Rules to Schematron Rules 26

8.1 Analysis of AIXM SBVR Rules and SBVR Profile for AIXM 26

8.1.1 Assignment / Existence Checks 26

8.1.2 Handling of Choices 27

8.1.3 Handling Inheritance 30

8.1.4 Expressions for XPath Axis Names 34

8.1.5 Regular Expressions 35

8.1.6 Inclusion of External Vocabularies 36

8.1.7 SRS Name 37

8.2 Schematron Derivation 38

8.2.1 Overview 38

8.2.2 Loading the Conceptual Models 38

8.2.3 AIXM Model (Core and Extensions) Merging 40

8.2.4 Loading SBVR Constraints from Excel File 42

8.2.5 Parsing SBVR Constraints to First Order Logic 43

8.2.5.1 Supported SBVR Grammar 44

8.2.5.2 Time Slice Handling 51

8.2.5.3 First Order Logic Language 51

8.2.6 Translation to Schematron 53

8.2.6.1 Recognition of AIXM Extension Elements 57

8.2.6.2 Support for Feature References in Schematron Code 58

8.2.6.3 Realization of ‘of-type’ operator 58

8.2.6.4 Realization of null-checks 60

Copyright © 2015 Open Geospatial Consortium

9 Accomplishments 61

Annex A: Revision history 62

Figures

Page

Figure 1 – Assignment check example – AircraftStand and ApronAreaAvailability in context 26

Figure 2 – ChangeOverPoint and <<choice>> SignificantPoint with its options 28

Figure 3 – Construction of a <<choice>>, equivalent to a <<union>> as defined in the ISO 19100 series of standards 29

Figure 4 – Inheritance example – AirTrafficControlService and Procedure in context 31

Figure 5 – General case of an inheritance relationship 32

Figure 6 - Inheritance example (2) – RunwayCentrelinePoint and NavaidEquipment in context 34

Figure 7 – Schematron derivation process 38

Figure 8 – package structure of AIXM, including the core conceptual schema and extensions 39

Figure 9 - AIXM extension mechanism - RunwayDirection belongsTo Event 41

Figure 10 – SBVR constraint defined directly in the UML model 42

Figure 11 – Grammar – an SBVR rule can be expressed in one of two ways 44

Figure 12 – Grammar – the constituents of a sentence in an SBVR rule 45

Figure 13 – Grammar – noun concept concatenation 45

Figure 14 – Grammar – relative clause expression 47

Figure 15 – Grammar – predicate expression 48

Figure 16 – Grammar – simple predicate 49

Figure 17 – Grammar – verb expression 50

Figure 18 – Overview of the First Order Logic language supported for SBVR constraint derivation 52

Figure 19 – Conceptual schema example for ‘of-type’ operator translation to Schematron 59

Tables

Page

Table 1 – Spatial relationships as identified by the AMDB constraint analysis, and their implementation using standardized operators 17

Table 2 – Different ways of selecting AIXM feature time slices in a business rule 24

Table 3 – The various cases of formulating a rule that includes a <<choice>> 29

Table 4 – The different cases of formulating a rule to identify classes in an inheritance hierarchy 32

Table 5 – Mapping of AIXM stereotypes to well-known stereotypes 39

Copyright © 2015 Open Geospatial Consortium

Table 6 – Quantifiers supported by the tool 46
Table 7 – Examples of rules with explicit and implicit use of the “timeSlice” property 51
Table 8 – Mapping of First Order Logic constructs to XPath/Schematron 54

Listings
Page

Listing 1 – Schematron for rules using ‘of-type’ operator 59

OGC® Testbed 11 Aviation - Guidance on Using Semantics of Business Vocabulary and Business Rules (SBVR) Engineering Report

1 Introduction

1.1 Scope

This document is a deliverable of the OGC Testbed 11. It describes the results of developing a tool to automatically derive Schematron code from SBVR constraints. It also documents a vocabulary with a profile of core geospatial terms and concepts, which can be used to express geospatial constraints in business rules.

1.2 Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Name	Organization
Johannes Echterhoff (editor)	interactive instruments GmbH

1.3 Future work

The following items were identified for consideration in future initiatives:

- **Detailed testing** - The Schematron derivation tool was implemented in less than four months. Only a very short amount of time was left to test the resulting Schematron. Future work should therefore include detailed testing. This is even more important when considering the complexity of the AIXM schemas, both the conceptual schema (also taking into account the Temporality Model) and the XML implementation schema.
- **Complete AIXM schema merging** – Testbed 11 defined a process to merge the information from AIXM extension schemas and the core schema on-the-fly. This is critical for parsing SBVR constraints that use concepts from AIXM extension schemas (for further details, see 8.2.3).
  The merging process defined in Testbed 11 supports features required for the Testbed 11 demonstration. The process must be extended in order to support all aspects relevant for merging AIXM schemas.
- **Add support for choices/unions and association classes** – Due to time and resource constraints the automation tool only implements core schema constructs. AIXM <<choice>> and <<union>> types as well as association classes are not supported yet. Future work should implement additional functionality – also taking into account the considerations on the handling of <<choice>> types as documented in 8.1.2.
- **Continue the work on the geospatial vocabulary** – Work in Testbed 11 focused on the definition of a geospatial vocabulary that includes a profile of core geospatial terms and concepts. Future work should test the use of the vocabulary in actual business rules, and implement support for spatial operators and geometry operands.
- **Regular expressions in SBVR rules** – Some of the AIXM business rules define constraints on the content of properties with textual value type (e.g. that the text value shall have at most three digits). Such free-text descriptions of how a textual value should be structured are very hard if not impossible for an automation tool to parse.
  A solution would be to add a predicate to the SBVR grammar that supports the specification of a regular expression. The regular expression could be translated to Schematron, which if XPath 2.0 is used may not even require an additional function library for the validation. For further details, see 8.1.5.
- **Schema-aware Schematron processor** – The Schematron rules generated by the automation tool could be simplified if they were written for a schema-aware Schematron processor. Especially the check to determine that a given object is of a specific type could be improved this way (using the schema-element() XPath 2.0 function).
- **Support for rules involving elements of external schemas** – At the moment the automation tool cannot generate Schematron code for constraints if they address concepts from external schemas where the XML encoding is unknown.
  The encoding could be unknown because the external schema has a specific

encoding that is not defined by a set of known rules, or because the external schema is not contained in the model that is being processed, or because the external schema does not provide information about the encoding rules that shall be used to derive its XML encoding.
Future work should investigate if mappings between elements from the conceptual schema and their equivalents in the XML encoding can be created. Such mappings could be used by the automation tool to derive Schematron code, also for SBVR constraints that use concepts from external schema.

- **Mapping between First Order Logic (FOL) and Object Constraint Language (OCL)** – SBVR constraints are parsed into First Order Logic as intermediate language, which is converted to Schematron. An analysis should be performed to see if a mapping exists between FOL constructs and OCL. Such a mapping could be used to transform FOL constructs into OCL, and thus leverage already existing OCL parsing and derivation functionality. It could also be used to turn OCL constraints into more human readable expressions, so that they can be understood by non-experts.

1.4 Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

2 References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

Other OGC Documents:

- [OGC 10-088r3] OGC OWS-7 Schema Automation Engineering Report
- [OGC 12-028r1] Use of GML for Aviation Data, OGC Discussion Paper
- [OGC 12-147] OGC OWS-9 Aviation Architecture Engineering Report

Aviation Documents:

- [AIXMTM] AIXM 5 Temporality Model, available online at http://www.aixm.aero/gallery/content/public/AIXM51/AIXM%20Temporality%201.0.pdf (accessed May 8, 2015)
- [AIXMASGEN] AIXM Application Schema Generation, available online at http://www.aixm.aero/gallery/content/public/AIXM51/AIXM_Application_Schema_Generation-1.1.pdf (accessed May 10, 2015)
- [AIXMUML2XSD] AIXM UML to XML Schema Mapping, available online at http://www.aixm.aero/gallery/content/public/AIXM51/AIXM_UML_to_AIXM_XSD_Mapping-1.1.pdf (accessed March 16, 2015)
- Digital NOTAM Event Specification ed 1.0, available online at http://www.aixm.aero/gallery/content/public/digital_notam/Specifications/Digital%20NOTAM%20Event%20Specification%201.0.doc (accessed March 17, 2015)
- [SBVR Profile for AIXM] AIXM 5.1 - Business Rules (data verification) – Using SBVR and Schematron, v0.3 available online at https://extranet.eurocontrol.int/http://webprisme.cfmu.eurocontrol.int/aixmwiki_public/bin/download/Main/AIXM_Business_Rules/AIXM%2D5.1%2DBusinessRules%2DusingSBVRandSchematronV0.3.docx (accessed May 10, 2015)

Other Documents:

- [ISO 19103] Geographic information – Conceptual schema language
- [ISO 19107:2003] Geographic information – Spatial schema
- [ISO Committee Draft 19107:2015] Geographic information – Spatial schema
- [ISO 19111] Geographic information – Spatial referencing by coordinates
- [ISO 19125-1] Geographic information – Simple feature access – Part 1: Common architecture
- [ISO 19162] Geographic information – Well known text representation of coordinate reference systems

Copyright © 2015 Open Geospatial Consortium

3 Terms and definitions

For the purposes of this report, the following terms and definitions apply.

3.1 AIXM business rule

business rule that is under the jurisdiction of the aeronautical information domain.

3.2 business rule

rule that is under business jurisdiction.

NOTE: can be represented via a constraint, e.g. an SBVR constraint

3.3 rule

One of a set of explicit or understood regulations or principles governing conduct or procedure within a particular area of activity ... a law or principle that operates within a particular sphere of knowledge, describing, or prescribing what is possible or allowable.

3.4 SBVR constraint

A constraint (defined for an element of the conceptual schema) which is written in SBVR.

4 Abbreviated terms

AIXM	Aeronautical Information Exchange Model
AMDB	Aerodrome Map Database
DNES	Digital NOTAM Event Specification
DNOTAM	Digital NOTAM
EAP	Enterprise Architect Project
EPSG	European Petroleum Survey Group
FOL	First Order Logic
GIS	Geographic Information System
GML	Geography Markup Language
ISO	International Organization for Standardization
NOTAM	Notice to Airmen
OMG	Object Management Group
OCL	Object Constraint Language
OGC	Open Geospatial Consortium
OMG	Object Management Group
SBVR	Semantics of Business Vocabulary and Business Rules
SQL	Structured Query Language
SRS	Spatial Reference System
UML	Unified Modeling Language
UoM	Unit of Measure
WKT	Well-known-text
XML	Extensible Markup Language
XPath	XML Path Language
XSLT	Extensible Stylesheet Language Transformations

Copyright © 2015 Open Geospatial Consortium

5 OGC Testbed 11 Guidance Using SBVR - Overview

One of the topics addressed by the OGC Testbed 11 Aviation thread was the use of constraints represented using Semantics of Business Vocabulary and Business Rules (SBVR). AIXM business rules are expressed as SBVR constraints. The tasks regarding SBVR in the Aviation thread were to:

- Define a vocabulary/profile with “geospatial” terms for SBVR. For example: object1 (spatially) intersects object2.
- Determine to which extent the implementation of SBVR business rules can be automated. Target implementations: Schematron, maybe also OCL.

The following chapters document the results of the work conducted in Testbed 11 for these tasks.

6 Geospatial Vocabulary

6.1 Overview

A large set of requirements exists for processing of "geospatial" information. ISO and OGC standards cover common requirements including, but not limited to:

- describing the spatial characteristics of a feature, their geometry and topology
- spatial referencing by coordinates and geographic identifiers
- linear referencing
- describing spatial operations

The standards specify a set of terms which form a "geospatial" vocabulary. However, they do not use SBVR to do so. Rather, the terms are specified either explicitly in the "Terms and Definitions" section, or implicitly in the normative text and schemas.

Capturing all the terms from the range of ISO and OGC standards within a "geospatial" SBVR vocabulary would have been a tremendous task far exceeding the scope of Testbed 11. The vocabulary documented in this chapter therefore contains a profile of core geospatial terms and concepts. More specifically, the vocabulary focuses on the spatial operators that most GIS processing software – especially spatial databases – usually supports.

6.2 Vocabulary

6.2.1 Spatial Characteristics

NOTE: *solid* is not defined in this vocabulary because solids are not used in AIXM, and are not supported by ISO 19125 (which is the basis for most GIS processing software).

boundary

Definition:	A boundary is a set that represents the limit of an entity.
Source:	[ISO 19107]

closure

Definition:	The closure is the union of the interior and boundary of a topological or geometric object.
Source:	[ISO 19107]

coordinate

Copyright © 2015 Open Geospatial Consortium

Definition: A coordinate is one of a sequence of N-numbers designating the position of a point in N-dimensional space.

Source: [ISO 19111]

coordinate reference system

Definition: A coordinate reference system is a coordinate system that is related to the real world by a datum.

Source: [ISO 19111]

coordinate system

Definition: A coordinate system is a set of mathematical rules for specifying how coordinates are to be assigned to points.

Source: [ISO 19111]

curve

Definition: A curve is a 1-dimensional geometric primitive, representing the continuous image of a line.

Note: The boundary of a curve is the set of points at either end of the curve. If the curve is a cycle, the two ends are identical, and the curve (if topologically closed) is considered to not have a boundary. The first point is called the start point, and the last is the end point.

Broader concept: geometric primitive

Synonyms: GM_Curve [ISO 19107]

Source: [ISO 19107]

direct position

Definition:	A direct position is a position described by a single set of coordinates within a coordinate reference system.
Source:	[ISO 19107]

exterior

Definition:	The exterior [of a geometric object] is the difference between the universe and the closure [of the geometric object].
Source:	[ISO 19107]

geometric aggregate

Definition:	A geometric aggregate is a collection of geometric objects that has no internal structure.
Note:	No assumptions about the spatial relationships between the elements can be made.
Source:	[ISO 19107]

geometric boundary

Definition:	A geometric boundary is a boundary represented by a set of geometric primitives of smaller geometric dimension that limits the extent of a geometric object.
Source:	[ISO 19107]

geometric object

Definition:	A geometric object is a spatial object representing a geometric set.
Synonyms:	GM_Object [ISO 19107], geometry
Source:	[ISO 19107]

Copyright © 2015 Open Geospatial Consortium

geometric primitive

Definition:	A geometric primitive is a geometric object representing a single, connected, homogeneous element of space.
Broader concept:	geometric object
Synonyms:	GM_Primitive [ISO 19107]
Source:	[ISO 19107]

geometric set

Definition:	A geometric set is a set of direct positions.
Source:	[ISO 19107]

interior

Definition:	The interior [of a geometric object] is the set of all direct positions that are on a geometric object but which are not on its boundary.
Source:	[ISO 19107]

pattern matrix

Source:	[ISO CD 19107]

point

Definition:	A point is a 0-dimensional geometric primitive representing a position.
Note:	The boundary of a point is the empty set.
Broader concept:	geometric primitive

Synonyms: GM_Point [ISO 19107]

Source: [ISO 19107]

surface

Definition: A surface is a 2-dimensional geometric primitive, locally representing a continuous image of a region of a plane.

Note: The boundary of a surface is the set of oriented, closed curves that delineate the limits of the surface.

Broader concept: geometric primitive

Synonyms: GM_Surface [ISO 19107]

Source: [ISO 19107]

topological object

Definition: A topological object is a spatial object representing spatial characteristics that are invariant under continuous transformations.

Source: [ISO 19107]

6.2.2 Other adopted concepts

measure

Definition: A measure is the result from performing the act or process of ascertaining the value of a characteristic of some entity.

Source: [ISO 19103]

Copyright © 2015 Open Geospatial Consortium

6.2.3 Spatial Relationship Operators

The definitions of named spatial relationships between geometric objects follow the definitions in [ISO CD 19107]. They are based on the Dimensionally Extended nine-Intersection Model (DE-9IM). Because the DE-9IM based specification of each named relationship is very formal and cannot be expressed in simple English for each case, the following verb concepts only refer to the source of the definition, but do not provide the definition itself. In addition to the named operators, the "relates" operator has been included to support specific use cases. Where appropriate for use in SBVR rules, suggestions for synonyms have been added.

geometry *contains* geometry

Source: [ISO CD 19107] [section 10.7.5.3.2]

geometry *equals* geometry

Source: [ISO CD 19107] [section 10.7.5.3.1]

geometry *disjoint* geometry

Source: [ISO CD 19107] [section 10.7.5.3.3]

Synonyms: is-disjoint-with

geometry *touches* geometry

Source: [ISO CD 19107] [section 10.7.5.3.5]

Synonyms: meets

geometry *within* geometry

Source: [ISO CD 19107] [section 10.7.5.3.7]

Synonyms: is-within, inside, is-inside

geometry *overlaps* geometry

Source: [ISO CD 19107] [section 10.7.5.3.8]

geometry *crosses* geometry

Source: [ISO CD 19107] [section 10.7.5.3.6]

geometry *intersects* geometry

Source: [ISO CD 19107] [section 10.7.5.3.4]

geometry *relates* geometry as pattern matrix

Note: This operator requires two geometries and a pattern matrix to test against.

Source: [ISO CD 19107] [section 10.7.5.2 – full topological relate]

6.2.4 Spatial Analysis Operators

geometry *is-within-distance* of measure to geometry

Note: This is equal to testing that a) a buffer created around the first geometry (with given measure as distance) and b) the second geometry intersect.

Source: [ISO CD 19107] [section 10.7.6 - ‘DWithin’]

geometry *is-beyond* measure to geometry

Note: This is equal to testing that a) a buffer created around the first geometry (with given measure as distance) and b) the second geometry are disjoint.

Source: [ISO CD 19107] [section 10.7.6 - ‘Beyond’]

Copyright © 2015 Open Geospatial Consortium

6.3 Considerations for use in business rules

6.3.1 Geometry representation

The set of AIXM SBVR business rules that have been analyzed in Testbed 11 applied predicates on variable values. The variables were declared through quantifications, and specific properties of the variable value were selected for use in the predicate via noun and verb concepts.

For example, in rule *“each AirportHeliport shall have availability.operationalStatus equal-to 'CLOSED'”* there is an implicit variable that ranges over all AirportHeliports. The noun *“availability.operationalStatus”* selects the operationalStatus property of AirportHeliportAvailability objects that are owned by a given AirportHeliport.

A binary predicate can be applied by having a variable value on the left-hand side of a binary expression and a literal on the right-hand side. In the example, ‘CLOSED’ is the string literal used on the right-hand side of the ‘equal-to’ comparison operator.

Spatial operators can be used in a similar way. The left-hand side would identify a variable value that has a spatial type while the right-hand side of the operator would be a literal.

In case of the ‘within-distance’ operator the literal should be a measure, consisting of a number and a unit of measure. For other spatial operators we need a literal representation of a geometry. We recommend using the well-known-text (WKT) encoding defined by [ISO 19125-1] to express such literals.

If the coordinates of a geometry represented in WKT are not given in a default spatial reference system (which could be EPSG 4326), then it would be necessary to state the SRS explicitly. This could be achieved using a well-known-text description of the SRS as defined by [ISO 19162]. How the SRS would be included in a business rule would need to be determined in future work.

NOTE: having a variable value point to a spatial property of an AIXM feature has the benefit of precisely identifying a parameter for a spatial operator. However, that property may not be easy to identify. On the one hand, it may not be directly contained in the feature itself (but for example in an object referenced by the feature).On the other hand, the actual geometry of an AIXM feature may need to be computed on-the-fly – an Airspace is an example (with the geometry being defined by a combination of airspace volumes).

Writing business rules with spatial constraints would be much easier if each AIXM feature had a well-known ‘geometry’ property. The SBVR to Schematron conversion could detect that the ‘geometry’ property is used and delegate the computation of the geometry to another software component.

The service based computation of geometry for AIXM features has been investigated in OGC Testbed 9 (see [OGC 12-147], chapter 8). If rules existed to compute a geometry for each AIXM feature type then business rules could make use of the ‘geometry’ property as a shortcut.

6.3.2 Spatial operator use

Business rules using spatial operators could be formulated as follows:

An AirportHeliport with contact.address.country equal-to 'Germany' shall have ARP within POLYGON((5.75 55.5,15 55.5,15 47,5.75 47,5.75 55.5)).

A business rule may require that none of the geometries to be tested by a spatial relationship operator is a literal. An example of such a rule in free text is:

A RunwayMarking feature shall be contained in a RunwayElement feature and/or a RunwayDisplacedArea feature and/or a Stopway feature and/or a RunwayIntersection feature and/or Blastpad feature.

This rule requires that the spatial relationship of a given RunwayMarking geometry is checked against the geometries in the range of all RunwayElement, RunwayDisplacedArea, Stopway, RunwayIntersection and Blastpad features. In order to implement this we need variables to identify the geometries of RunwayMarking objects as well as the other objects. So far the variable for the left-hand side of a binary expression is implicitly defined in a quantification. The SBVR grammar would need to be extended to support declaration of the other variables, either implicitly or explicitly[2]. This requires further analysis and development.

6.4 Analysis of geometric constraints from the Aerodrome Map Databasse

An analysis of Aerodrome Map Database (AMDB) constraints, more specifically the geometric constraints and the keywords they use, was provided as input for the development of the geospatial vocabulary. The analysis contained diagrams that had been created to explain the meaning of the keywords. This section provides an analysis of these keywords and their potential implementation using the spatial relationship operators defined in the vocabulary.

[2] The Object Constraint Language (OCL) uses 'let' expressions to explicitly declare variables that can be used in following expressions.

Table 1 – Spatial relationships as identified by the AMDB constraint analysis, and their implementation using standardized operators

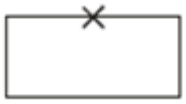
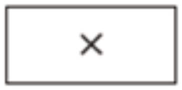
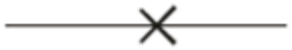
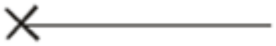
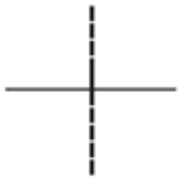
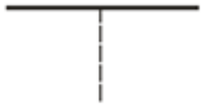
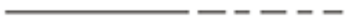
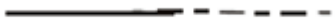
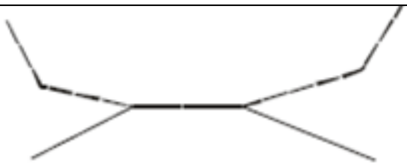
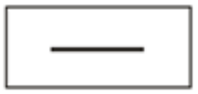
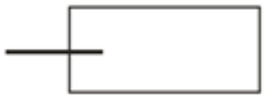
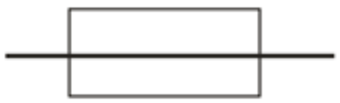
Image depicting the spatial relationship **(source: AMDB constraint analysis)**	**Textual description of spatial relationship using geometric keywords** **(source: AMDB constraint analysis)**	**Analysis with suggestion for implementation using vocabulary terms (based on standardized spatial relationship operators)**
	Point is located at the edge of Polygon	Point *touches* Polygon
	Point is contained in Polygon	Point *is-within* Polygon
	Point is located on Line	Point *intersects* Line In this situation one could also say that: Point *is-within* Line
	Line ends at Point	Line *touches* Point
	Line1 crosses Line2	Line1 *crosses* Line2
	Line1 starts/ends at the edge of Line2	Line1 *touches* Line2
	Line1 is connected to Line2	If the intention is to ensure that the end points of the two lines intersect but nothing else then one can use the *relates* operator,

Image depicting the spatial relationship **(source: AMDB constraint analysis)**	**Textual description of spatial relationship using geometric keywords** **(source: AMDB constraint analysis)**	**Analysis with suggestion for implementation using vocabulary terms (based on standardized spatial relationship operators)**
		checking that: the boundary of Line1 shall intersect the boundary of Line2 but the interior of Line1 shall not intersect the interior of Line 2
	Line1 overlaps Line2	Line1 *overlaps* Line2
	Line1 is attached to Line2	Line1 *overlaps* Line2 and Line1 does not *cross* Line2
	Line is contained in Polygon	Line *is-within* Polygon
	Line intersects Polygon	Line *intersects* Polygon The depicted relationship would also be covered by the *crosses* operator: Line *crosses* Polygon
	Line crosses Polygon	Line *crosses* Polygon In order to ensure that the end points of the line are outside the polygon one could also say that: Line *crosses* Polygon and the boundary of the Line does not

Copyright © 2015 Open Geospatial Consortium

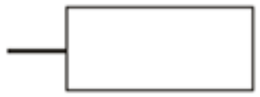
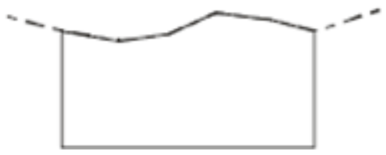
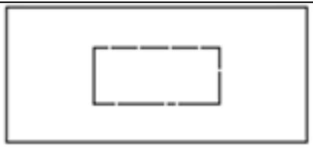
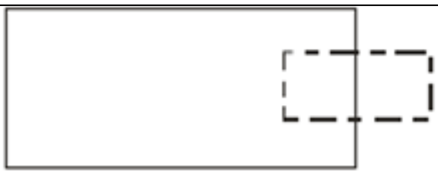
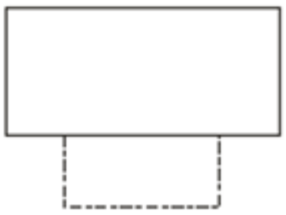
Image depicting the spatial relationship **(source: AMDB constraint analysis)**	**Textual description of spatial relationship using geometric keywords** **(source: AMDB constraint analysis)**	**Analysis with suggestion for implementation using vocabulary terms (based on standardized spatial relationship operators)**
		intersect the closure of the Polygon
	Line starts/ends at the edge of Polygon	Line *touches* Polygon
	Line is attached to Polygon	The depicted relationship can be represented with varying degrees of complexity (and detail). We can start with: *Line touches Polygon* In order to ensure that the intersection is not just a point, we can also check the dimension by adding: *and the dimension of the intersection between the interior of the Line and the closure of the Polygon is equal to 1* Furthermore, in order to ensure that the end points of the line are not on the border of the polygon, we can add: *and the intersection between the boundary of the Line and the closure of the Polygon is empty*

Image depicting the spatial relationship **(source: AMDB constraint analysis)**	**Textual description of spatial relationship using geometric keywords** **(source: AMDB constraint analysis)**	**Analysis with suggestion for implementation using vocabulary terms (based on standardized spatial relationship operators)**
		We can see that, depending on the desired outcome of the relationship test, we need to use a combination of spatial relationship operators. The *relates* operator would allow us to express a complex relationship directly.
	Polygon1 is contained in Polygon2	Polygon2 *contains* Polygon1
	Polygon1 overlaps Polygon2	Polygon1 *overlaps* Polygon2
	Polygon1 is attached to Polygon2	Again, the depicted relationship can be represented with varying degrees of complexity. We can start with: *Polygon1 touches Polygon2* In order to ensure that the intersection is not just a point, we can also check the dimension by adding: *and the dimension of the intersection between the closure of Polygon1 and the closure of*

Copyright © 2015 Open Geospatial Consortium

Image depicting the spatial relationship **(source: AMDB constraint analysis)**	**Textual description of spatial relationship using geometric keywords** **(source: AMDB constraint analysis)**	**Analysis with suggestion for implementation using vocabulary terms (based on standardized spatial relationship operators)**
		Polygon2 is equal to 1 The *relates* operator would allow us to express the intended complex relationship directly.

<u>Conclusions & Recommendations</u>

The spatial relationships identified by the AMDB constraint analysis can be implemented using the set of named spatial relationship operators standardized by ISO/OGC. In addition, the “relates” operator can be useful to identify specific relationships.

We recommend that a geospatial vocabulary used in the Aviation community uses the spatial relationship operators standardized by OGC and ISO. This would support understanding of geometric constraints by experts within the wider geospatial community.
Nevertheless, if other names for spatial relationship operators have already been established within the aviation community, it would be possible to use those names in an aviation specific vocabulary while still adopting the definition and meaning of the standardized names. However, this requires that no conflict of terms is created (for example, defining “intersects” differently).

Depending upon the digitalization quality of geometries in aeronautical data, spatial relationship tests may require the use of buffering, i.e. the distance-within and beyond operators. For example, for a line to be touching a point (fourth example in Table 1), the line end and the point must have the same coordinates. Any gap between the two points will cause the *touches* relationship operator evaluating to false.

7 Guidance for SBVR Rule Development

7.1 Identify Schema Level that SBVR Rule is written for

For automated derivation of Schematron from an SBVR rule, it is important to know the schema level for which the SBVR rule has been written[3]. This can be the conceptual level and the implementation level.

If a rule has been written for the conceptual schema level, avoid using concepts that are only available in the implementation schema.

For example:

- XPath like notation for noun concept concatenation using “/” and qualified names (ns:element).
- Specific axis notation that is built upon the XML structure like *descendant-or-self* (see 8.1.4 for further details).

The [SBVR Profile for AIXM] should clearly identify for which schema level a construct defined by the profile is applicable. For example, “is-descendant-of” and “has descendant” should only be used for rules on the XML implementation schema level. An exception would be if such constructs were also clearly defined for the conceptual level.

In general, we recommend writing business rules only for the conceptual level. This allows deriving rule information for different implementation schema, for example XML (in form of Schematron code) but also SQL (as SQL constraints). This would also avoid that the business expert that writes the rules needs to understand a specific implementation. Last but not least, it is also better because a specific implementation technology may be replaced at some point in the future by another technology, which would require that rules are re-written.

7.2 Be aware of implications when writing rules involving (dynamic) AIXM feature types

Each AIXM feature is a dynamic feature. Conceptually, these features are represented by a set of time slices. The AIXM UML model does not explicitly show this. The abstract AIXMFeature class has a set of timeSlices. The model does not show that AIXM feature types such as AirportHeliport inherit from AIXMFeature. Rather, this relationship is implicit. Furthermore, according to the UML model the properties of actual AIXM feature types belong to the feature. However, conceptually speaking they are owned by the according time slice type (e.g. AirportHeliportTimeSlice, which can be found in the XML implementation schema but not in the conceptual schema). This has implications for how business rules should be written.

[3] Instead of “written for” one can also say “targets”: an SBVR rule targets a specific schema level

We use the following (made-up) rule as example:

Each AirportHeliport shall have assigned designator value.

NOTE: there is an implicit quantification of "at-least-one" for "assigned designator value" (see description of the SBVR grammar in 8.2.5.1); the rule therefore is actually: "Each AirportHeliport shall have at-least-one assigned designator value."

Let us assume that the intention behind this rule is to ensure that throughout the lifetime of an AirportHeliport feature, it shall always have a 'designator' value that is not null. The designator value is stored within time slices owned by the AirportHeliport feature – both conceptually and in actual XML encoded data. According to the AIXM Temporality Model [AIXMTM] there are different interpretations for time slices: baseline, permdelta, tempdelta, and snapshot. Without going into too much detail here, we can say that the time slices represent the state of an AIXM feature throughout its lifetime. Changes to property values are tracked through time and stored in time slices. A temporary change, for example, is stored in a tempdelta. A permanent change can be stored in a baseline and a permdelta. Last but not least, a snapshot represents the state of a feature for a given point in time. Values for the 'designator' property will be stored in all baselines and snapshots, but likely not in all permdeltas and tempdeltas.

A key question when evaluating AIXM business rules on actual AIXM data is for which time slices a condition must be checked. The rule from the example does not specify this.

Discussions within Testbed 11 revealed that AIXM business rules can be defined for all four or a subset of time slice interpretation types. It therefore is not possible to automatically select a specific subset of time slices – for example just baselines and snapshots – to evaluate a business rule against.

In order to automatically derive Schematron from SBVR rules, a software tool must therefore be able to identify which time slice types need to be checked. This can be achieved in two ways:

- The rule itself contains the selection of time slices, as required.
- Metadata for a rule states which time slice interpretations are to be checked. The excel file that contains AIXM business rules has such metadata.

The solution chosen for Testbed 11 was to express the necessary time slice selection in the business rule itself, for the following reasons:

- A business rule may require different sets of interpretations for different conditions, which cannot be expressed using a single field in an excel spreadsheet.
- The software solution should also support the parsing of SBVR constraints if they are directly contained in the UML model, and not loaded from an excel file.

The following table shows the different ways of performing a selection of time slices based upon their interpretation.

Table 2 – Different ways of selecting AIXM feature time slices in a business rule

Business rule	Selected time slices
Each AirportHeliport.timeSlice with interpretation equal-to (‘BASELINE’,’SNAPSHOT’) shall have at least one assigned designator value.	Here we have an explicit selection of time slices based upon their interpretation value. Each baseline and snapshot is checked to see if it has a designator value (not being null), while tempdeltas and permdeltas are ignored.
Each AirportHeliport.timeSlice shall have at least one assigned designator value.	This can be problematic, because each and every time slice – thus also a tempdelta – must have a designator value (not being null); otherwise the translated rule will evaluate to false. If the AIXM data to be checked with Schematron rules only contained baselines and snapshots then this should not be a problem. The validation of AIXM data and the way that AIXM business rule must be written in order to achieve correct results can thus be influenced by the preprocessing (in this case: selection) performed on the input data.
Each AirportHeliport shall have at least one assigned designator value.	The tool recognizes that AirportHeliport is an AIXM feature type and therefore adds the “timeSlice” property as a segment before all calls to properties of that feature. The rule thus will become: *Each AirportHeliport shall have at least one assigned timeSlice.designator value.* This can be problematic, because if just one time slice – for example a tempdelta – exists that had a designator value (not being null), the translated rule will evaluate to true. The result would also be true if just one out of the possibly many baselines belonging to the feature had a designator value. Of course, if the input data contained only one baseline or snapshot this wouldn’t be a

Copyright © 2015 Open Geospatial Consortium

Business rule	Selected time slices
	problem. So again, the validation depends on the input data.

Conclusions & Recommendations

We can conclude that the way an AIXM business rule takes time slices into account can have an impact on the validation result and that the validation result can be influenced through a selection of the input data.

We recommend that AIXM business rules be written with explicit selection of time slices. This should be done whenever the rule states a condition that must be fulfilled for an AIXM feature property.

NOTE: Testbed 11 did not analyze the implications of other common time slice properties such as valid time as well as sequence and correction numbers. Together with the time slice interpretation these properties are essential when determining the value(s) of an AIXM feature for any point in time. It is not clear if Schematron tests should be concerned with the actual state of an AIXM feature (that would need to be computed from all time slices on-the-fly). If the actual state is important a preprocessing step could be introduced. It would compute a snapshot which can be tested by the Schematron code.

8 Automated Derivation of SBVR Business Rules to Schematron Rules

8.1 Analysis of AIXM SBVR Rules and SBVR Profile for AIXM

NOTE: for simplicity reasons the rules presented in this section do not take time slice selection into account. For a detailed description of that topic, see section 7.2.

8.1.1 Assignment / Existence Checks

Example

AIXM-5.1_RULE-1A8519: *It is obligatory that each AircraftStand with assigned availability value isOperationalBy exactly one ApronAreaAvailability*

Analysis

In the context of AircraftStand, noun “availability” and verb “isOperationalBy” refer to the same property value, which is of type ApronAreaAvailability.

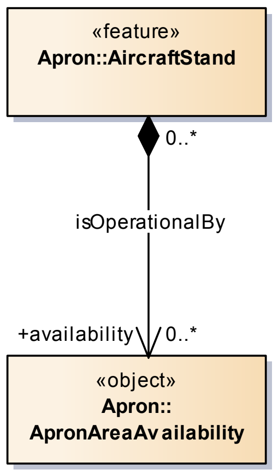


Figure 1 – Assignment check example – AircraftStand and ApronAreaAvailability in context

In this example, the cardinality of *availability* is 0..*. In other cases, it can be 0..1.

The “assigned” keyword is an existential quantification. The SBVR profile adds to say that “… the referent thing is not null”. The use of “assigned” boils down to a null / existence check. It is a shortcut for the existential quantifier which, according to the SBVR profile, can be expressed with “at least one”.

“assigned” is part of a predicate that is (potentially part of) a filter on a set of values. In the example, this set is the set of all AircraftStand objects. In other words, the rule targets only those AircraftStand objects that have at least one *availability* value that is not null.

This essentially constitutes a selection of AircraftStand objects. The expression "*isOperationalBy exactly one ApronAreaAvailability*" contains a quantification that targets the same property as the assignment used in the selection: *availability* (because "isOperationalBy" is the name of the association where *availability* is the role that represents a property of AircraftStand).

An existential quantification checks that a certain number of elements of a given set[4] exist. For an element to exist, it must not be null. A quantification of a variable X therefore implies an assignment check for that variable.

In summary, the rule can be simplified to: *it is obligatory that each AircraftStand isOperationalBy at most one ApronAreaAvailability*

If the rule was intended to ensure that exactly one *availability* exists for each AircraftStand, it could be written as follows:

- *It is obligatory that each AircraftStand isOperationalBy exactly one ApronAreaAvailability* or
- *Each AircraftStand shall have exactly one availability.*

8.1.2 Handling of Choices

Example

AIXM-5.1_RULE-1A853D: *It is obligatory that each ChangeOverPoint with assigned airportReferencePoint value isLocatedAt exactly one AirportHeliport*

Analysis

The conceptual model of a ChangeOverPoint has an optional "location" property of type "SignificantPoint" (see Figure 2).

[4] In the example that is the set of all *availability* property values of a given AircraftStand.

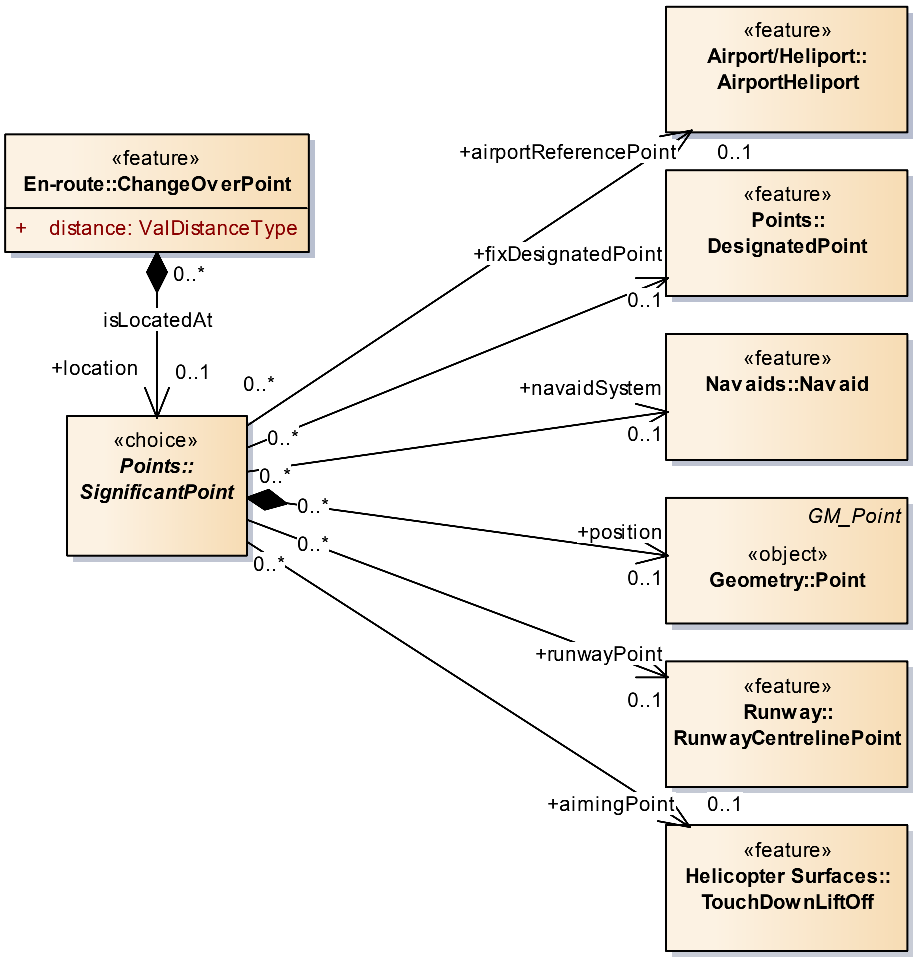


Figure 2 – ChangeOverPoint and <<choice>> SignificantPoint with its options

The location.SignificantPoint has not been included in the SBVR rule because SignificantPoint is a "choice" class and at first it was decided to consider them "transparent".

Using the association name as verb that identifies the choice and then the name of the class that represents the desired choice works if all choices have different types. However, this is not true in general. Consider the following figure.

Copyright © 2015 Open Geospatial Consortium

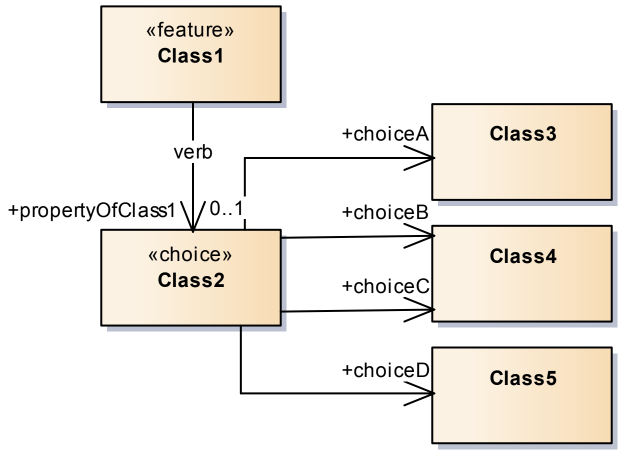


Figure 3 – Construction of a <<choice>>, equivalent to a <<union>> as defined in the ISO 19100 series of standards

It is perfectly valid to have two choices of the same type, because the semantics of each choice would be different.

The rule "*It is obligatory that each Class1 verb exactly one Class4*" would be ambiguous, because it is not clear whether choiceB or choiceC is allowed.

Thus, it is better to be explicit when formulating a rule that involves a <<choice>> type.

Three cases need to be considered:

1. The <<choice>> itself shall be identified – for example for a general existence check.
2. One of the available choices shall be identified.
3. Two or more of the available choices shall be identified.

Rules for the different cases can be formulated as shown in Table 3.

Table 3 – The various cases of formulating a rule that includes a <<choice>>

	Using shall / shall not	**Using obligation / prohibition**
Case 1	Each Class1 shall have *quantification* propertyOfClass1.	It is obligatory that each Class1 has *quantification* propertyOfClass1.
	Each ChangeOverPoint shall have a location. [a)]	It is obligatory that each ChangeOverPoint has a location. [a)]
Case 2	A Class1 shall have *quantification* choiceB as propertyOfClass1.	It is obligatory that a Class1 has *quantification* choiceB as

		propertyOfClass1.
	A ChangeOverPoint shall have an airportReferencePoint as location.	It is obligatory that a ChangeOverPoint has an airportReferencePoint as location.
Case 3	A Class1 shall have *quantification* choiceA or choiceB or choiceC as propertyOfClass1.[b)]	It is obligatory that a Class1 has *quantification* choiceA or choiceB or choiceC as propertyOfClass1.[b)]
a) The cardinality in the example ( ) would theoretically allow a SignificantPoint (rather: a non-abstract subtype of SignificantPoint) without any actual choice value, because the choices themselves are optional. However, the <<choice>> class does not appear in the AIXM XML encoding. Instead, the name of an XML element that represents a choice is a concatenation of the role name of the <<choice>> class with the role name of the target class of each choice branch, separated by "_" (see [AIXMUML2XSD] for further details). The UML to GML application schema encoding rules for <<union>> classes prevent this mismatch between conceptual and implementation schema. There, the properties of a <<union>> always have cardinality 1. b) The choices must be concatenated with 'or'.		

We suggest the introduction of the construct "choice1 or … or choiceN as" to identify the choice or choices of interest. This appears to result in a suitable formulation of the rule.

8.1.3 Handling Inheritance

Example

AIXM-5.1_RULE-BB801: *It is obligatory that each Service specialization TrafficSeparationService specialization AirTrafficControlService with assigned clientProcedure value controls exactly one Procedure specialization StandardInstrumentDeparture*

Analysis

According to the SBVR profile for AIXM, "specialisation" is an additional categorisation fact-type used to target a specific non-abstract subclass in an inheritance hierarchy. In this case, the rule shall ensure that an AirTrafficControlService has at most one[5] clientProcedure of type StandardInstrumentDeparture.

[5] Section 8.1.1 explains why the rule results in an „at most one" quantification, rather than an „exactly one".

Copyright © 2015 Open Geospatial Consortium

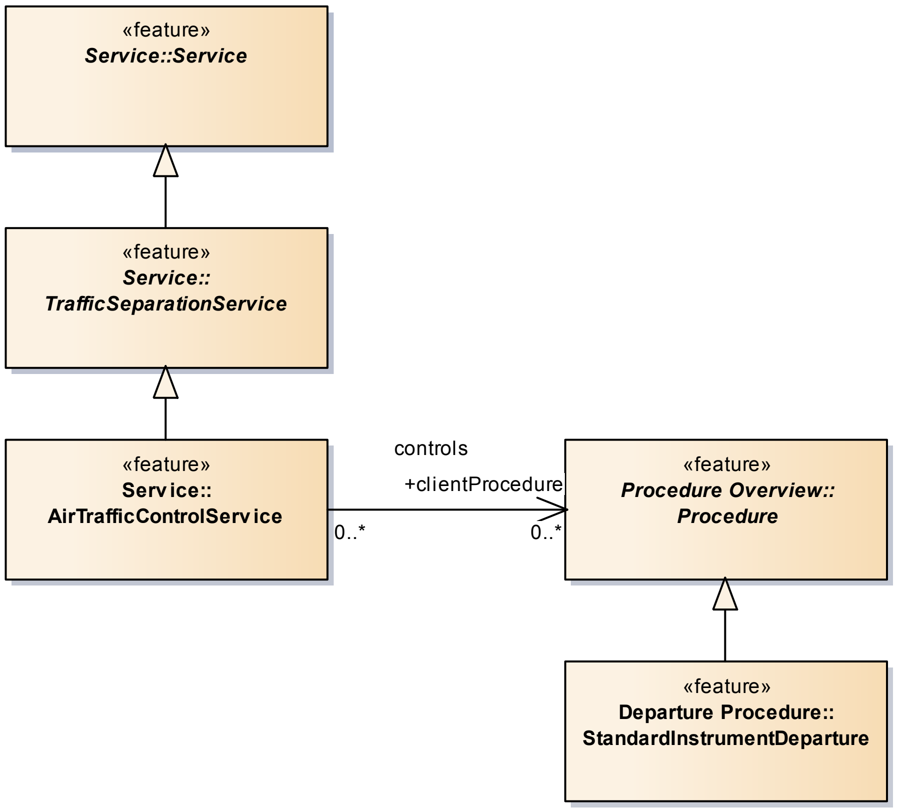


Figure 4 – Inheritance example – AirTrafficControlService and Procedure in context

Documenting all parents of the rule relevant classes is unnecessary and overly complicates the SBVR rule. The inheritance information is already contained in the conceptual schema from which it can be retrieved by the automation tool. The rule can therefore be simplified to:

“It is obligatory that each AirTrafficControlService controls at most one StandardInstrumentDeparture”

The context class – AirTrafficControlService – is clearly identified by the rule. The value type of property “clientProcedure” is the abstract type “Procedure”. It has multiple subtypes, StandardInstrumentDeparture being one of them. The tool can determine that StandardInstrumentDeparture is a valid substitution for Procedure. This should be sufficient information to derive schematron rules.

When it gets to inheritance, we need to consider two cases:

1. The set of classes composed of a given type and all its subtypes shall be identified.
2. Only a specific subset of the subtypes shall be identified.

Rules for the different cases can be formulated as shown in Table 4. The rule formulations in the table not only cover the specific AIXM SBVR rule used as example so far. They also cover a more general example, which is shown in the following figure.

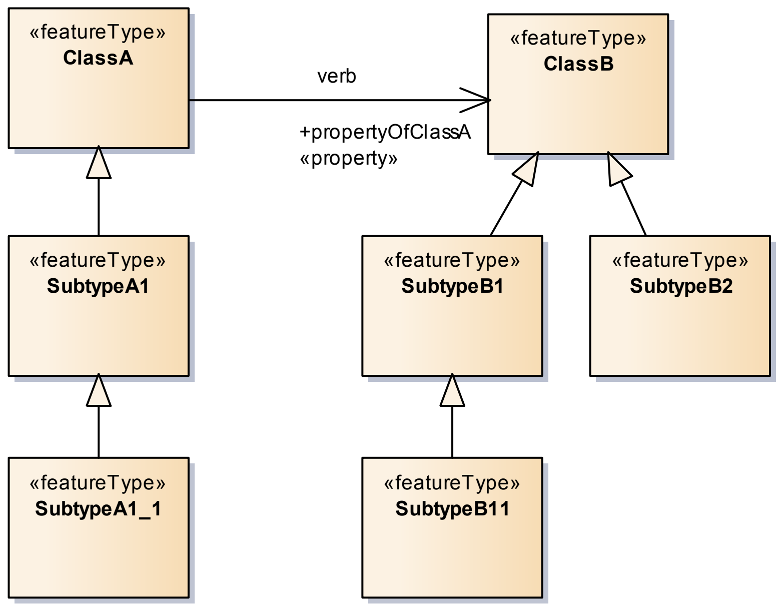


Figure 5 – General case of an inheritance relationship

As we can see, both ClassA and ClassB have a set of subtypes. Keep in mind that any subtype can be used to represent its supertype, following the semantics of inheritance defined by UML. It therefore is not necessary - and in fact would even be wrong – to exclude a specific subtype in a rule when its supertype would be included (for example, excluding SubtypeB1 or SubtypeB11 when ClassB was included). We can therefore focus on rule formulations that allow us to identify those classes we are really interested in. All subtypes of these classes should automatically be identified as well.

Table 4 – The different cases of formulating a rule to identify classes in an inheritance hierarchy

	Using shall / shall not	**Using obligation / prohibition**
Case 1	Each AirTrafficControlService shall have at most one clientProcedure	It is obligatory that each AirTrafficControlService controls at most one Procedure
	Each ClassA shall have *quantification* propertyOfClassA	It is obligatory that each ClassA *verb quantification* ClassB
Case 2	Each AirTrafficControlService shall have at most one	It is obligatory that each AirTrafficControlService controls at

Copyright © 2015 Open Geospatial Consortium

	clientProcedure of type StandardInstrumentDeparture	most one StandardInstrumentDeparture
	Each ClassA shall have *quantification* propertyOfClassA of type *SubtypeB11* or *SubtypeB2*	It is obligatory that each ClassA *verb quantification SubtypeB11* or *SubtypeB2*

Note that when using "shall" and "shall not" the keyword "of type" is introduced to support a more natural formulation of the rule. This is not necessary when obligation and prohibition is used as modality.

This approach allows rules to be expressed against abstract types, removing the need to create rules for each subtype if only properties of the supertype are rule relevant.

Likewise, it would remove the need to list all possible subtypes in the rule. An example for this is:

AIXM-5.1_RULE-BA47A: *It is obligatory that each RunwayCentrelinePoint with assigned navaidEquipment value hasEstablished exactly one NavaidEquipment specialization Localizer or Glidepath or VOR or TACAN or DME or NDB or MarkerBeacon or SDF or Elevation or DirectionFinder or Azimuth*

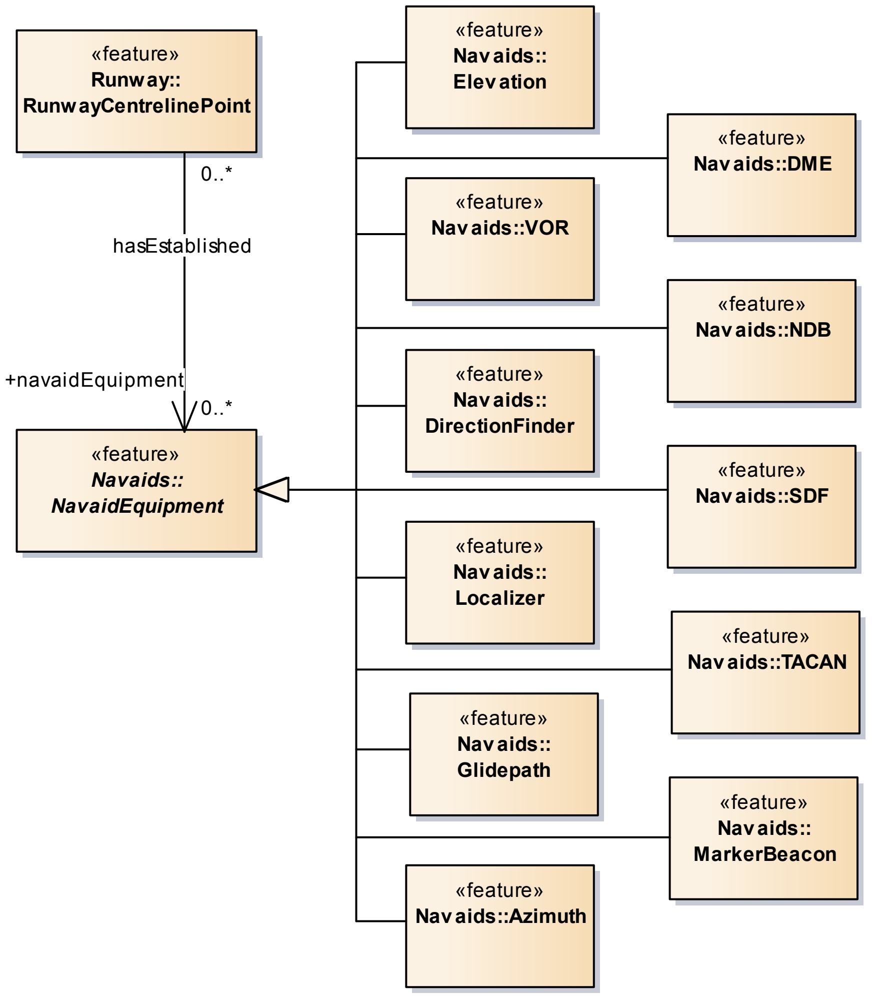


Figure 6 - Inheritance example (2) – RunwayCentrelinePoint and NavaidEquipment in context

When deriving Schematron rules the tool should automatically take inheritance into account. The *schema-element()* XPath 2.0 function would be very useful to identify a type and all possible subtypes. However, schema aware XPath processing may not always be available in Schematron processors. As a fallback, the tool uses the names of all subtypes found in the model. The drawback of that approach is that any potential extension that is not contained in the model and that added further subtypes will not be recognized by the Schematron rules generated by the tool.

8.1.4 Expressions for XPath Axis Names

<u>Example</u>

AIXM-5.1_RULE-1A3EC1: *Each Curve or ElevatedCurve shall not have descendant ArcStringByBulge*

Copyright © 2015 Open Geospatial Consortium

Analysis

The keywords "has/have descendant" and "is-descendant-of" are introduced by the SBVR Profile for AIXM as additional fact-types/verbs. They heavily relate to the *descendant-or-self* axis notation in XPath expressions.

Use of these verbs is suitable when the XML implementation is the schema level that an SBVR rule targets (see 7.1). If the schema level is the conceptual level, i.e. rules are written for the UML model, the "descendant" verbs are not appropriate because the structure of a UML model is different compared to an XML document. The relationships between elements in the UML model are not only parent-child relationships. There can be reflexive associations and cyclic relationships which are not covered by XPath axes.

Apparently, the rule in the example has been written for the XML implementation schema. It uses "Curve" and "ArcStringByBulge" instead of "GM_Curve" and "GM_ArcStringByBulge" which are contained in the conceptual model of ISO 19107.

If the rule was written for the conceptual schema, it should read: *"Each GM_Curve shall not have a segment of type GM_ArcStringByBulge."* Derivation of Schematron for this rule would require additional knowledge about the specific implementation of ISO 19107 in XML. Testbed 11 focused on rules for a given application schema – AIXM – which has well-defined encoding rules to derive XML Schema from the conceptual model. Support for rules written for external models, especially with a very specific XML implementation, is therefore a future work item.

8.1.5 Regular Expressions

Example

AIXM-5.1_RULE-1A4A7A: *Each AerialRefuellingAnchor with refuellingBaseLevel.uom equal-to ('FL', 'SM') shall have refuellingBaseLevel value expressed with 1, 2 or 3 digits*

Analysis

It is very hard for a tool to recognize conditions like "value expressed with 1, 2 or 3 digits". Whenever the textual representation of a property value must comply with a specific structure, regular expressions could be used. They provide a rich feature set to control text structure.

Using a regular expression, the rule from the example can be formulated as follows:

Each AerialRefuellingAnchor with refuellingBaseLevel.uom equal-to ('FL', 'SM') shall have refuellingBaseLevel matching '\d{0,3}'.

Admittedly, if the expert that creates SBVR rules is not experienced with regular expressions, it would be hard for him or her to write such a rule. On the other hand, creating regular expressions is not too hard. The expert could therefore have someone familiar with regular expressions write the expression for him. The benefit would be that the tool is then able to automatically derive Schematron code from the rule.

XPath 1.0 does not support regular expression matching. Its successor, XPath 2.0, provides such functionality. By default, Schematron uses the XPath language as used in XSLT 1.0 – which is XPath 1.0. Modern tools also support XSLT 2.0 and thus XPath 2.0 for Schematron validation. At this point it is not clear whether the regular expression support offered by XPath 2.0 is sufficient or not. If not, a new function could be introduced that realizes regular expressions as required[6]. This would have the added benefit that the function can be defined for both XPath 1.0 and 2.0. The cost would be that it requires an extension of the Schematron processor.

8.1.6 Inclusion of External Vocabularies

Example

AIXM-5.1_RULE-29FE0: *A Note shall not (be-descendant-of FeatureTimeslice with assigned descendant event:theEvent value) and have assigned translatedNote.note value using a different character set from {'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z', 'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','0','1','2','3','4','5 ','6','7','8','9','-','?',':','(',')','.',',',''''','=','/','+'}*

Analysis

The presence of "be-descendant-of" indicates that this rule is targeting the XML implementation schema, rather than the conceptual schema (see 8.1.4). The rule would also be a candidate for using a regular expression (see 8.1.5). Of primary interest in this section, however, is the presence of "event:theEvent" in the rule.

"theEvent" is a property introduced by the Digital NOTAM Event Specification (DNES). The construct "event:theEvent" very likely represents a qualified name. In the XML encoding qualified names uniquely identify the namespace to which an element belongs. In this case "event" hints at the namespace assigned to the DNES, because "event" is often used as the XML namespace abbreviation for elements of this schema.

The conceptual schema of the DNES is not included in the core AIXM UML model. The DNES is an AIXM application schema that is an extension of the AIXM 5.1 model. As such, it is not part of the AIXM vocabulary represented by the core conceptual schema of AIXM. According to [1] there are different ways to include a term (noun or verb) from an external vocabulary in SBVR rules written against a given vocabulary (in our case: AIXM):

[6] Such an approach has been taken before, for regular expression matching in OCL expressions (for further details, see OGC 10-088r3 section 6.6.7).

- Adopt the term and its definition
- Quote the term
- Include the external vocabulary

Suffice to say that in the context of writing SBVR rules with a conceptual schema providing most of the vocabulary[7] these approaches essentially represent a schema import. Qualification of a term is only needed in case that the (rule) context of the term does not uniquely identify the vocabulary that the term belongs to. For the example, this means that “theEvent” can be used without using “event:” as qualifier, because the core AIXM schema does not contain a property or association named “theEvent”. Qualification can be achieved in different ways:

- Using qualified names as in XML – example: event:theEvent
- Using a UML style – example: {DNES application schema name}::theEvent
- Using a namespace identifier (often used in [1]) – example: theEvent [DNES namespace]

The latter may be easier to read and write for someone who is not familiar with XML.

In order for a tool to recognize terms that belong to an external schema, the tool must know that schema. In order to recognize DNES terms, the tool would therefore need to know all classes that belong to the conceptual schema of DNES, their inheritance relationships, their attributes and associations, as well as any schemas that DNES imports. The best way to achieve this is to provide a consolidated UML model to the tool, with all these details readily available. The consolidated model would include the core AIXM schema as well as the DNES schema plus any imported schema.

8.1.7 SRS Name

Example

AIXM-5.1_RULE-3E8: *Each assigned srsName value shall be equal-to 'urn:ogc:def:crs:EPSG::4326'*

Analysis

“srsName” is not part of the conceptual model of ISO 19107. Apparently the SBVR rule is written for the XML implementation schema of AIXM, not its conceptual schema. It makes sense to do so because this covers all cases where an SRS name can occur with just a single rule.

Note that this rule creates a bit of tension, because (OGC 12-028r1) implies that – at least theoretically – EPSG::3395 could be used to express a rhumbline where the latitudes of two consecutive points of a curve segment are different.

[7] The rest is provided by the SBVR profile for AIXM.

8.2 Schematron Derivation

8.2.1 Overview

The process of deriving Schematron code from business rules expressed in SBVR consists of five steps (Figure 7) which are described in detail in the following sections.

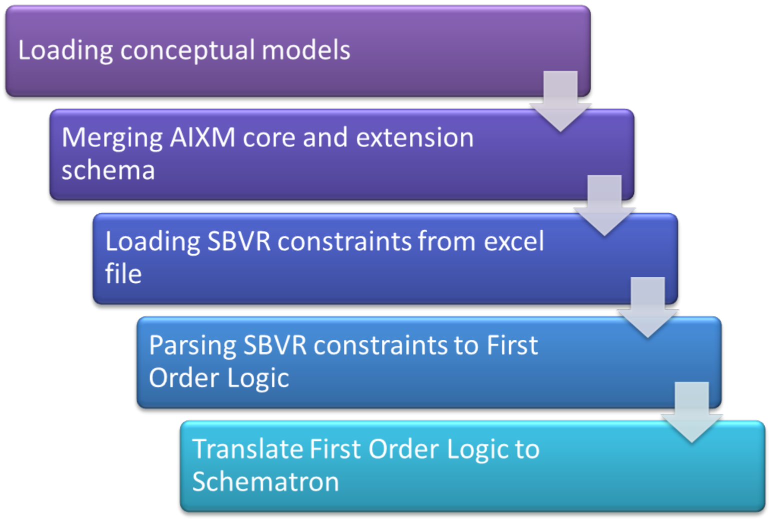

Loading conceptual models
Merging AIXM core and extension schema
Loading SBVR constraints from excel file
Parsing SBVR constraints to First Order Logic
Translate First Order Logic to Schematron

Figure 7 – Schematron derivation process

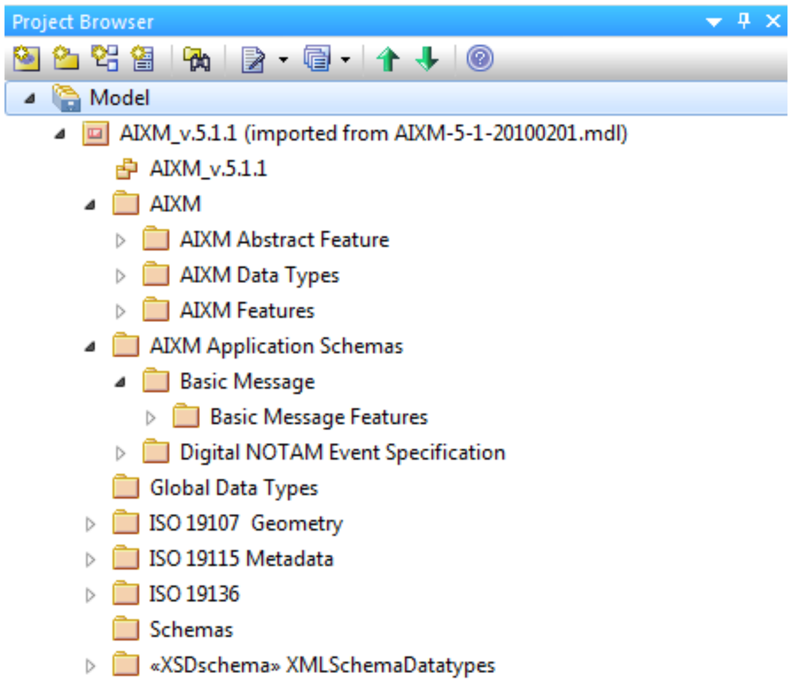
The functions required for the execution of each step are implemented by ShapeChange[8], an open source tool that converts application schemas in UML to GML application schemas, Schematron schemas, JSON schemas and many other representations.

8.2.2 Loading the Conceptual Models

The conceptual schema of AIXM 5.1 was made available via a Sparx Systems Enterprise Architect Project (.eap) file. Within Testbed 11, the conceptual schema for the Digital NOTAM Event Specification has been added to the model as well.

[8] http://shapechange.net

Copyright © 2015 Open Geospatial Consortium



Model
AIXM_v.5.1.1 (imported from AIXM-5-1-20100201.mdl)
AIXM_v.5.1.1
AIXM
AIXM Abstract Feature
AIXM Data Types
AIXM Features
AIXM Application Schemas
Basic Message
Basic Message Features
Digital NOTAM Event Specification
Global Data Types
ISO 19107 Geometry
ISO 19115 Metadata
ISO 19136
Schemas
«XSDschema» XMLSchemaDatatypes

Figure 8 – package structure of AIXM, including the core conceptual schema and extensions

Loading the model from the .eap file is achieved using readily available ShapeChange functionality. Only the packages "AIXM" and "Digital NOTAM Event Specification" (see Figure 8) are loaded as application schema.

While loading the schema contents, stereotypes used in AIXM are mapped to stereotypes that are well-known by ShapeChange. Well-known stereotypes for the purposes of ShapeChange are those defined in the UML specification, and in the ISO 19100-series of specifications, specifically ISO/TS 19103:2005, ISO 19109:2005 and GML 3.2/ISO 19136:2007. Table 5 documents the stereotype mapping applied in Testbed 11.

Table 5 – Mapping of AIXM stereotypes to well-known stereotypes

Stereotype used in AIXM model	**Well-known Stereotype**
object, message	<*none*> (an empty stereotype identifies an object type)
choice	union
feature	featureType

Once the AIXM schemas have been loaded, they are merged as described in the following section.

8.2.3 AIXM Model (Core and Extensions) Merging

AIXM has a specific way of adding information to core schema classes that is different to how it is usually done in UML and in ISO application schema. Where usually information is added through subtyping, AIXM has the concept of extensions. An extension schema is a schema that:

- Can define <<extension>> classes that extend:
    - either a specific feature or object type
    - or all feature types[9]
- Can define <<codelist>> classes that extend code lists from the core schema
- Can define new code lists as well as feature and object types

The objective of this approach is described in [AIXMASGEN] section 1.3 as follows:

"The core AIXM model provides the definition of standardised aeronautical information features. In order to use AIXM for a specific application, a Community of Interest (COI) will have to agree upon how instances of AIXM features are to be exchanged and communicated in the community. […]

In the definition of the AIXM Application Schema, the COI might also want to extend the core AIXM with additional properties and features. Some principles that regulate such extensions include:

- *An extension of an existing AIXM feature should remain valid against the definition of the core AIXM XSD element with the same name (for that purpose, the AbstractSomeFeatureExtension element is provided in the core AIXM XSD). A consequence is that it is not possible to extend <<datatype>> classes. Only <<codelist>> may be extended.*
- *An additional feature and objects shall follow the core AIXM modelling conventions (stereotypes, naming, data types, etc.)"*

A consequence of this approach is that actual AIXM data can contain information that is specified by multiple extensions, and that an AIXM processor is able to ignore unknown extensions to core AIXM features.

With the AIXM extension mechanism, AIXM feature and object types conceptually own all the properties that are added to them via <<extension>> types. This is useful for writing business rules, because it allows rules like the following:

Each RunwayDirection.timeSlice that belongsTo Event with scenario equal-to 'RWY.CLS' and with version equal-to '2.0' shall have exactly one assigned availability

[9] An example for such an extension is the "AnyAIXMFeature" type defined in the Digital NOTAM Event Specification, which adds "theEvent" property to all AIXM features. The property allows the time slice of an AIXM feature to reference the *Event* it belongs to.

value and shall have availability.ManoeuvringAreaAvailability.operationalStatus equal-to 'CLOSED'.

“belongsTo” is the name of an association that is added via the Digital NOTAM Event Specification to all AIXM feature types (see Figure 9).

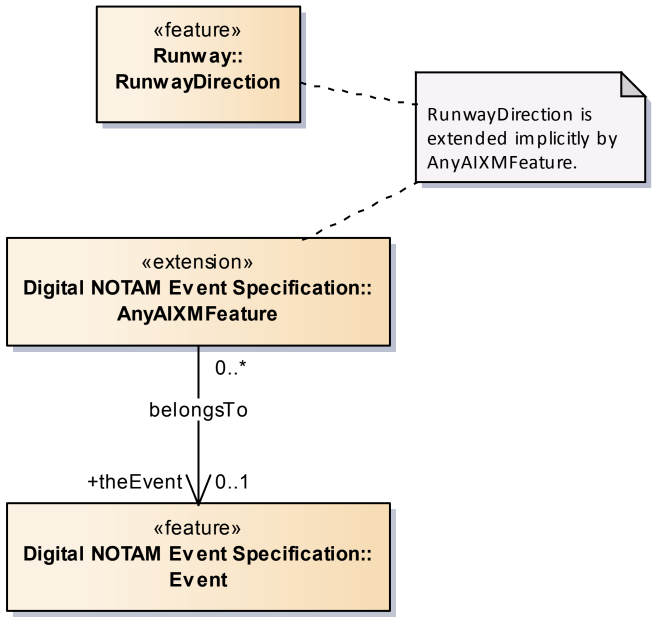

«feature»
Runway::
RunwayDirection
RunwayDirection is extended implicitly by AnyAIXMFeature.
«extension»
Digital NOTAM Event Specification::
AnyAIXMFeature
0..*
belongsTo
+theEvent
0..1
«feature»
Digital NOTAM Event Specification::
Event

Figure 9 - AIXM extension mechanism - RunwayDirection belongsTo Event

In order to parse AIXM business rules and generate Schematron code with correct XPath expressions, ShapeChange checks that the noun and verb concepts declared in business rules are compliant to the conceptual model of AIXM. This means that a business rule can only make statements about properties of an AIXM feature if the feature actually has these properties.

The rule chosen as example in this section makes use of the property “theEvent” via the verb “belongsTo” that refers to “Event”. The property is defined in the DNES extension schema and accessed as if it belonged to the feature type RunwayDirection. According to the conceptual model shown in Figure 9 RunwayDirection does not have this property. ShapeChange would therefore report an error while parsing the business rule[10] – unless the AIXM schemas are merged on-the-fly.

[10] The reason is that there is no explicit inheritance relationship between RunwayDirection and the extension class. The AIXM extension mechanism appears to prefer extensions of specific feature and object types through explicit inheritance relationships. However, the Digital NOTAM Event Specification is an example where an extension is declared for all feature types, not through explicit relationships but rather through implication.

ShapeChange supports a model transformation that merges AIXM extension schemas and the core schema. The result of the merging process is a single schema that contains the feature and object types declared in all schemas, where properties added via extensions have been copied to the relevant types. Merging also adds time slices to AIXM feature types. More specifically, time slice types are defined for each AIXM feature type, with the properties that belong to the feature type. This solves the issue of time slices not explicitly being defined for AIXM feature types on the conceptual level. It also allows access to time slice specific properties like their "interpretation" in business rules (see section 7.2 for further details).

NOTE: due to time constraints in Testbed 11 work on the AIXM schema merging transformation focused on the realization of those aspects that were required to support the Testbed 11 demonstration scenario. Time slice properties like validTime as well as metadata properties are not supported yet. The realization of a complete AIXM schema merge covering all aspects of AIXM schemas is future work.

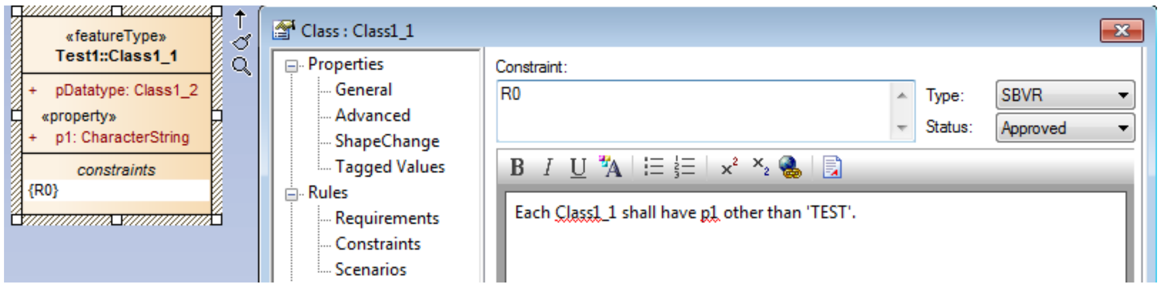
While merging the schemas, ShapeChange keeps track of XML Schema information – like the target namespace and the preferred namespace prefix – for extension schema elements. This is required for creating correct XPath expressions and namespace declarations when creating Schematron code.

After the AIXM schemas have been merged, the AIXM business rules can be loaded into the model.

8.2.4 Loading SBVR Constraints from Excel File

UML class diagrams are useful to model the structure of information, as well as certain semantics. Requirements such as allowed value ranges for numeric properties, especially if dependent on other property values, can best be described using so called *constraints*.

Constraints can be specified on individual model elements. In practice they are usually specified on classes. There are different ways to express a constraint. It can be free text, for consumption by humans. The Object Constraint Language (OCL) is another option to define constraints. OCL is a standard from the Object Management Group (OMG). OCL expressions are very formal, highly expressive, and usually written for automated processing. In Testbed 11 SBVR was introduced as another type of constraint. Figure 10 illustrates that it is possible to define SBVR constraints directly in the UML model, using modeling software such as Enterprise Architect from Sparx Systems.



Constraint: R0

Type: SBVR

Status: Approved

Each Class1_1 shall have p1 other than 'TEST'.

Figure 10 – SBVR constraint defined directly in the UML model

Copyright © 2015 Open Geospatial Consortium

ShapeChange supports parsing of constraints that are defined directly on schema elements.

AIXM business rules are maintained outside of the AIXM UML model. The reason is that not all of them are generally applicable. Some of the business rules only apply to specific use cases. AIXM business rules thus need to be loaded on a case by case basis.

For this reason another ShapeChange transformation has been implemented in Testbed 11. It supports enriching the AIXM schema (once it has been merged, see previous section) with SBVR constraints that are stored in an excel spreadsheet.

The format of the spreadsheet must be as follows:

- the rules must be defined on a single sheet named “Constraints” (case is ignored)
- row numbering must be continuous, and start with 1
- row 1 contains the header information for the following rows
- the following columns are of interest (case of column names is ignored):
    - “Name” (required) - contains the name of a business rule
    - “Text” (required) - contains the text of a business rule
    - “Comments” (optional) – describes the rule
    - “Schema Package” (optional) – name of the package that contains the class the rule is specified for (for cases in which classes with the same name can be found in multiple packages)
    - “Class” (optional) – name of the class the rule is specified for; the name must be exactly as defined in the conceptual schema; therefore, do not use QNames

NOTE: the constraint loading function can be extended in the future to be more lenient with respect to formatting of the excel file. Aspects like column names can also be made configurable, to support the preferences of a given community.

It may appear strange that the class name is optional. The reason for this is that the constraint loader attempts to parse the name of the relevant class directly from the rule text. The current grammar supports this. Class names stated in the spreadsheet are given preference. In other words, class names are only parsed from rules if no class name is provided. If the name of the class that provides the context for a rule could not be found, or if the conceptual schema does not contain a class with that name, then a message is logged and the business rule is ignored.

Once the rules have been loaded into the AIXM schema, they can be processed by ShapeChange. In order to derive Schematron code from the constraints, they must first be parsed into an intermediate language, which is described in the following section.

8.2.5 Parsing SBVR Constraints to First Order Logic

The text of an SBVR constraint is parsed to a First Order Logic (FOL) predicate in a three-phased approach:

1. via a lexical analysis the SBVR text is converted into a stream of meaningful tokens
2. the token stream is then parsed to recognize the structure of the sentence
3. the result is transformed into an FOL predicate

Phase 3 involves the following semantic checks:

- ensure that the properties used in noun concepts or identified by verbs actually belong to the class in the given context
- ensure that there is no mix of ‘and’ and ‘or’ in logical combinations of verb expressions
- ensure that logical combination of relative clauses is not applied on nested expressions

NOTE: for further detail on the last two checks, see the documentation of the SBVR grammar in section 8.2.5.1.

If one or more errors are detected while parsing an SBVR constraint, they are logged by the tool. Processing will then move on to the next constraint.

The next section documents the SBVR grammar that is supported by the SBVR-to-Schematron processing tool. Section 8.2.5.2 provides more details on how time slices are handled during the parsing process. Finally, section 8.2.5.3 describes the First Order Logic supported by the tool.

8.2.5.1 Supported SBVR Grammar

The Schematron derivation tool recognizes a specific grammar which is described in this section. The grammar has been developed based upon information found in the SBVR standard and especially in the [SBVR Profile for AIXM] as well as the large set of AIXM SBVR rules that were available in Testbed 11.

A rule is given in one of two forms (see Figure 11): the modality is expressed using “shall” / “shall not” or it is expressed using “It is obligatory / prohibited that …” (a dot to complete the sentence is optional).

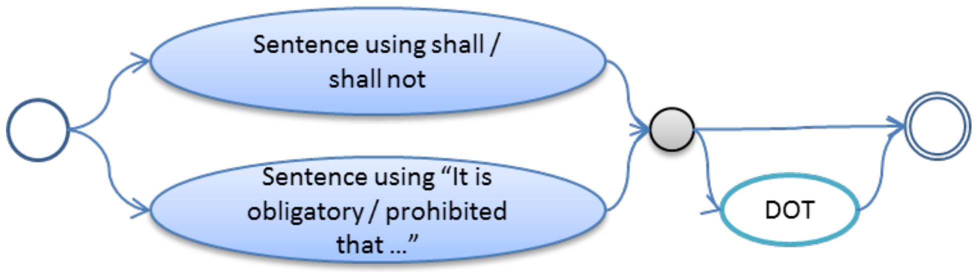

Sentence using shall / shall not
Sentence using “It is obligatory / prohibited that …”
DOT

Figure 11 – Grammar – an SBVR rule can be expressed in one of two ways

The following figure provides a more detailed view of these sentences.

Copyright © 2015 Open Geospatial Consortium

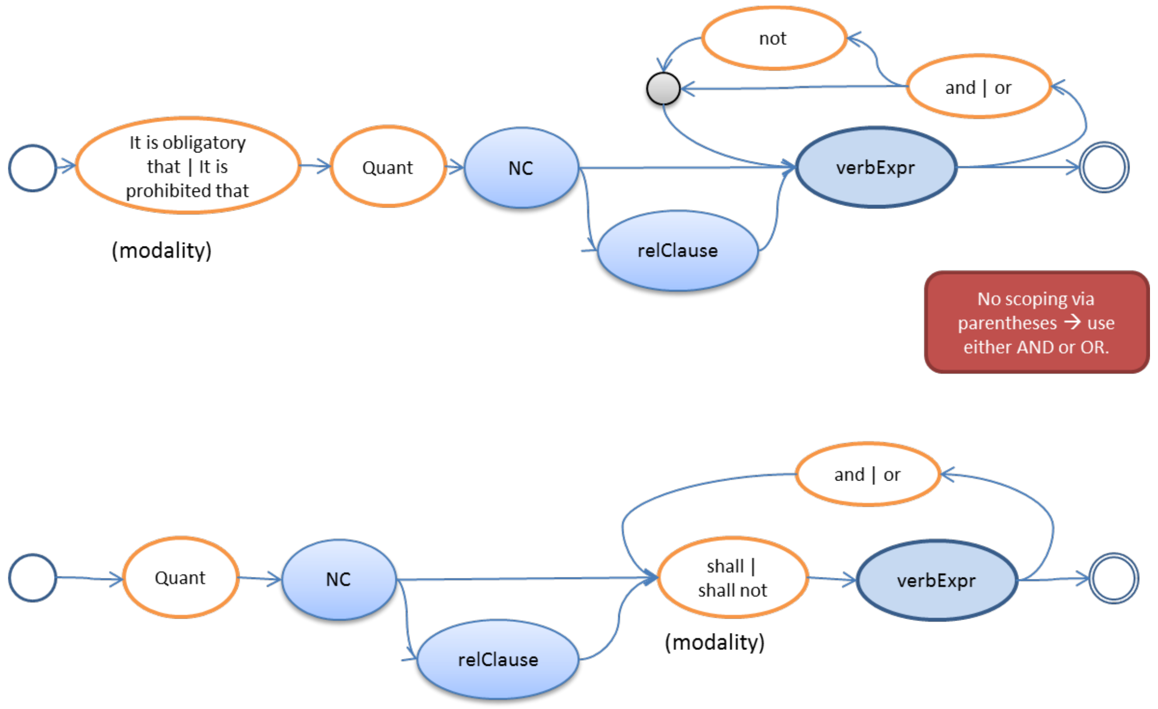


Figure 12 – Grammar – the constituents of a sentence in an SBVR rule

The structure of the two forms in which the sentence of an SBVR rule can be written is quite similar – only the way that modality is expressed is different.

We can see that there is an initial quantification for a noun concept. This is either a single class or property name from the conceptual AIXM schema or a combination thereof using dots as separators (Figure 13).

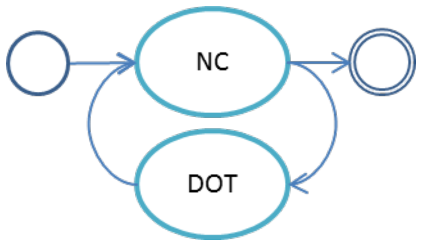


Figure 13 – Grammar – noun concept concatenation

The name of a concept must comply with the following regular expression: [a-zA-Z_][a-zA-Z0-9_]*

The tool supports the quantifiers listed in the [SBVR Profile for AIXM]:

Table 6 – Quantifiers supported by the tool

Quantifier Name	Written as (including the version with first character given in lower case)
universal	‘An’, ‘A’, ‘All’, ‘Each’
existential	‘At least one’
exactly-one	‘Exactly one’
at-most-one	‘At most one’
at-most-n	‘At most *INT*’
at-least-n	‘At least *INT*’
at-least-2	‘More than one’
exactly-n	‘Exactly *INT*’
numeric range	‘At least *INT* and at most *INT*’
Note: *INT* is an unsigned integer value	

The first noun concept provides the context for the whole sentence. It represents a set of objects (e.g. all Airspace features). An optional relative clause (Figure 14) can be used to apply a selection on this set of objects. In other words, we can test that an object satisfies a pre-condition before we check that it fulfills the main condition that is expressed via a (logical combination of) verb expression(s).

NOTE: because scoping via parentheses for logical expressions in SBVR rules is not foreseen at the moment, logical combination of verb expressions can only be created using ‘and’or ‘or’. A mix of ‘and’ and ‘or’ logical operators is not allowed.

Let us focus on the relative clause first.

Copyright © 2015 Open Geospatial Consortium

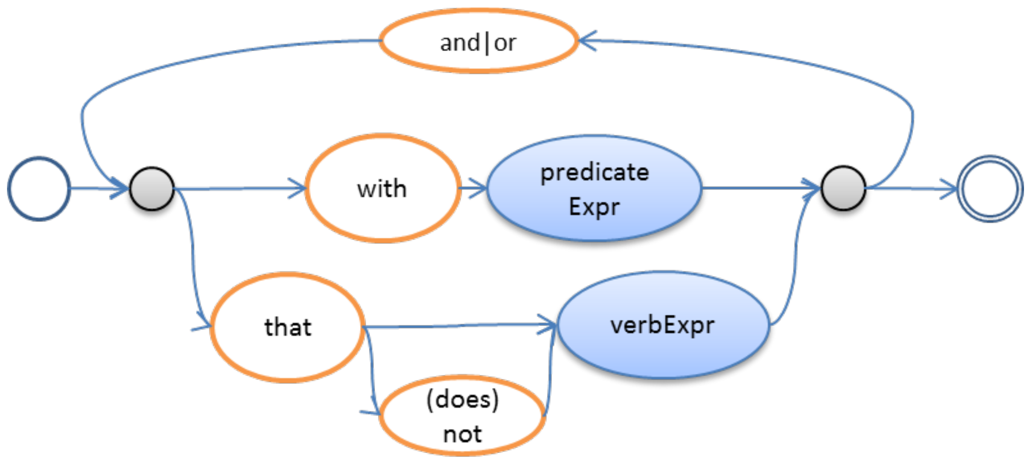


Figure 14 – Grammar – relative clause expression

A relative clause as shown in Figure 14 is a predicate expression (introduced by the keyword ‘with’), a verb expression (introduced by the keyword ‘that’, and possibly negated using ‘not’ or ‘does not’), or a logical combination thereof.

Ambiguity when using a logical combination of relative clauses and nested verb expressions

Note that a logical combination of relative clauses is prohibited if used in combination with nested verb expressions. The problem is that the context for the relative clause that is added using a logical expression would be ambiguous. Consider the following example:

“Each ClassA shall have prop1 that has prop2 that belongs to XYZ and that …”

Without any further punctuation marks the context of the relative clause introduced by “and that” can be prop1 but also prop2!

A predicate expression contains an optional quantifier and either an assignment predicate or simple predicate (Figure 15).

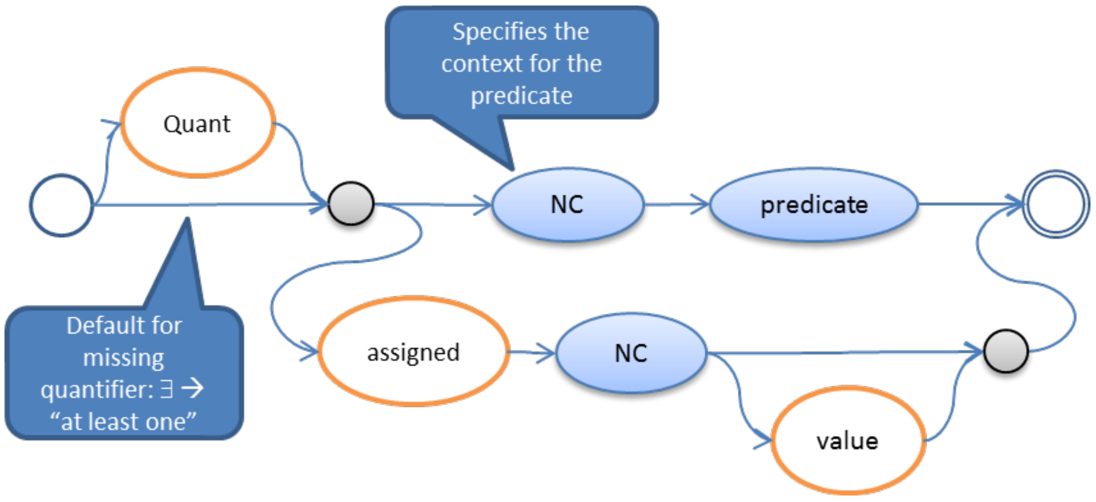

Specifies the context for the predicate
Quant
NC
predicate
Default for missing quantifier: ∃ → "at least one"
assigned
NC
value

Figure 15 – Grammar – predicate expression

By default, the tool interprets a not explicitly stated quantifier as the existential quantifier. The context for the following predicate is given by a noun concept. An assignment predicate is interpreted as a null check. A simple predicate can either be a type check or a simple comparison with a name or number (Figure 16)[11]. The negation of the predicate can be expressed via the optional 'not'.

[11] The range of available operators can be expanded in the future, for example adding spatial relationship operators as suggested in chapter 6.

Copyright © 2015 Open Geospatial Consortium

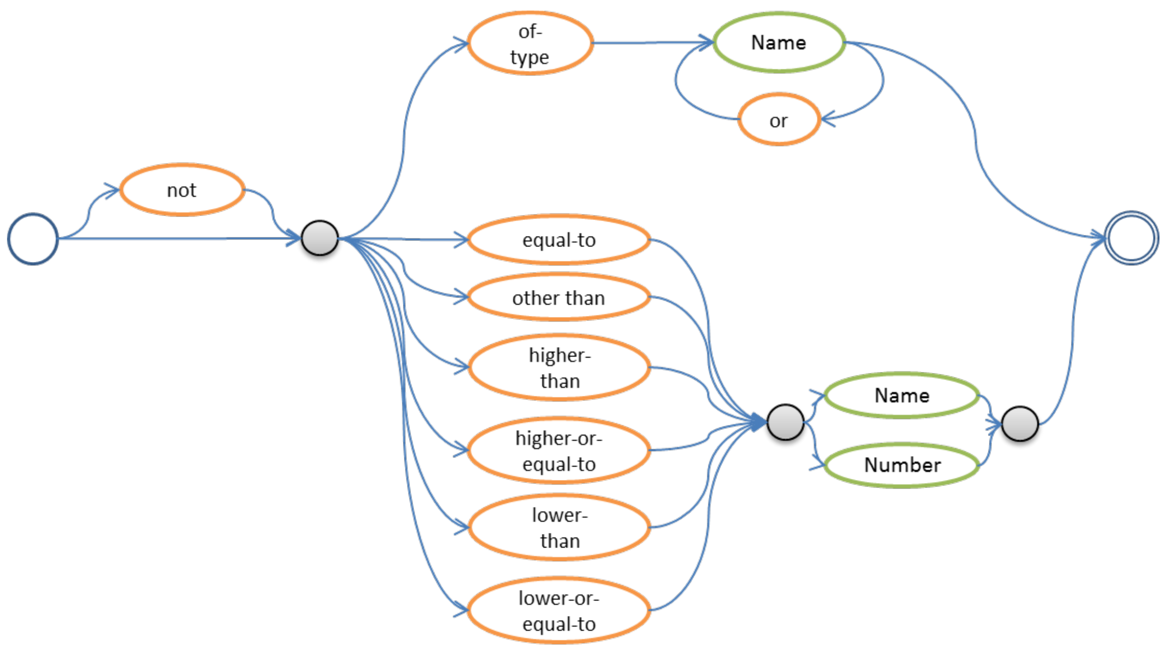


Figure 16 – Grammar – simple predicate

Whenever the grammar shows a "Name", it actually supports provision of a list of names as specified in the [SBVR Profile for AIXM]. A single name must comply with the following regular expression: \'[a-zA-Z0-9-_\.]+\'

A "Number" is an integer or real (with a single dot as separator and at least one digit on each side of the dot). It can be signed (with a minus or plus sign).

The grammar supports the comparison operators that are listed in the [SBVR Profile for AIXM]. The keywords for each operator shown in Figure 16 must be written in lower case, and separated either by spaces or hyphens (not both). Thus, 'lower than' and 'equal-to' are valid operators, but both 'Higher than' and 'lower-or equal to' are not[12].

If 'equal-to' is applied on a list of names, then the outcome is true if the value that is compared is equal to at least one of the names. The comparison operator 'other-than' is translated to not(equal-to(…)). Thus, when applied on a list of names the result is true if the value is not equal to any of the names.

The 'of-type' operator checks that an object is of a specific schema type. The operator expects at least one class name to be provided. Multiple class names can be given by separating them with 'or'. It is also possible to provide a list of class names (which has the same meaning as if they were separated by 'or'). Examples of rules that use this operator are given in 8.2.6.3.

[12] ‚Higher than' starts with an upper case character while 'lower-or equal to' mixes spaces and hyphens as separators.

The SBVR grammar supports verb expressions that are structured as shown in the following figure.

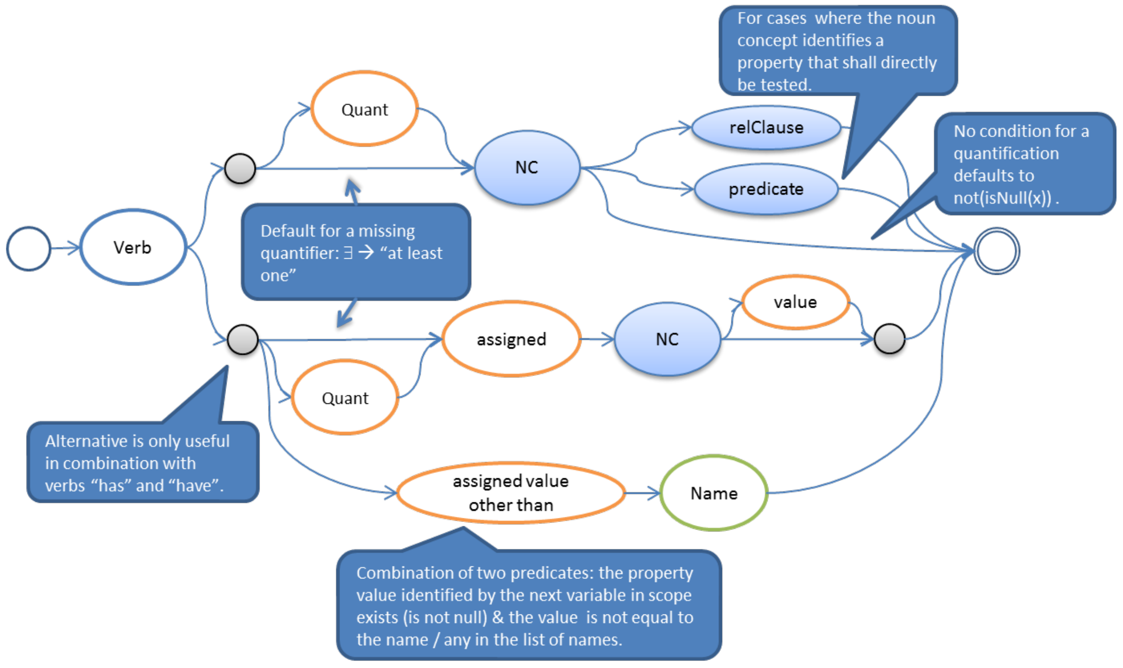

For cases where the noun concept identifies a property that shall directly be tested.
Quant
NC
relClause
predicate
No condition for a quantification defaults to not(isNull(x)) .
Verb
Default for a missing quantifier: ∃ → "at least one"
value
assigned
NC
Quant
Alternative is only useful in combination with verbs "has" and "have".
assigned value other than
Name
Combination of two predicates: the property value identified by the next variable in scope exists (is not null) & the value is not equal to the name / any in the list of names.

Figure 17 – Grammar – verb expression

A verb concept – including the reserved verbs "has" and "have" – opens up a verb expression.

Two special cases are supported to perform assignment checks. Both of them are usually only used in combination with the verbs "has" and "have". An assignment check like we've already seen in a predicate expression can be used, but also an assignment check that is combined with the 'other-than' comparison operator.

A verb expression can also be used to perform a quantification on a property A, more specifically its value(s). The set of values to be quantified can further be checked through predicates. Three cases are supported:

- a relative clause specifies a condition for the sub-properties of a given value (of property A),
- a predicate specifies a condition that must be fulfilled by property A itself;
- if the SBVR rule does not contain a condition for the quantification then by default an assignment check is performed.

Copyright © 2015 Open Geospatial Consortium

8.2.5.2 Time Slice Handling

If an SBVR rule does not explicitly state "timeSlice" as context for the property (represented by a noun or verb) of an AIXM <<feature>> then the parser automatically adds the "timeSlice" step directly in front of the property call, as shown in the following examples.

Table 7 – Examples of rules with explicit and implicit use of the "timeSlice" property

SBVR Rule	First Order Logic Expression
Each Airspace shall have assigned designator value	forall (x1:self.Airspace | exists{1..*} (x2:x1.timeSlice.designator | not (is-null(x2))))
Each RunwayDirection.timeSlice that belongsTo Event with scenario equal-to 'RWY.CLS' and with version equal-to '2.0' shall have exactly one assigned availability value and shall have availability.ManoeuvringAreaAvailability.operationalStatus equal-to 'CLOSED'	forall (x1:self.RunwayDirection.timeSlice | (not (exists{1..*} (x2:x1.belongsTo.Event | (exists{1..*} (x3:x2.timeSlice.scenario | x3 = RWY.CLS)) and (exists{1..*} (x4:x2.timeSlice.version | x4 = 2.0))))) or ((exists{1} (x5:x1.availability | not (is-null(x5)))) and (exists{1..*} (x6:x1.availability.ManoeuvringAreaAvailability.operationalStatus | x6 = CLOSED))))

When converted to Schematron, path expressions are represented by XPath expressions. If a "timeSlice" is added in front of a property call, then the result is an expression that targets all time slices of the feature object.

If multiple conditions need to be checked for an individual time slice or if a condition requires a specific quantifier for time slices, the SBVR rule should explicitly state the "timeSlice" context for the condition(s); also see section 7.2.

Let us take a look at the first example shown in Table 7. Here, the FOL expression that results from automatically adding the missing "timeSlice" step checks that each and every time slice of an *Airspace*, and thus also all TEMPDELTAs, has an assigned *designator* property. If this is only required for BASELINE and SNAPSHOT time slices, the rule can be formulated like this: "Each Airspace.timeSlice with interpretation equal-to 'BASELINE' or with interpretation equal-to 'SNAPSHOT' shall have assigned designator value".

8.2.5.3 First Order Logic Language

The derivation of Schematron from SBVR constraints is achieved by translating the information contained in the constraint text into an intermediate language, the First Order

Logic (FOL) language. Figure 18 provides an overview of the supported language constructs.

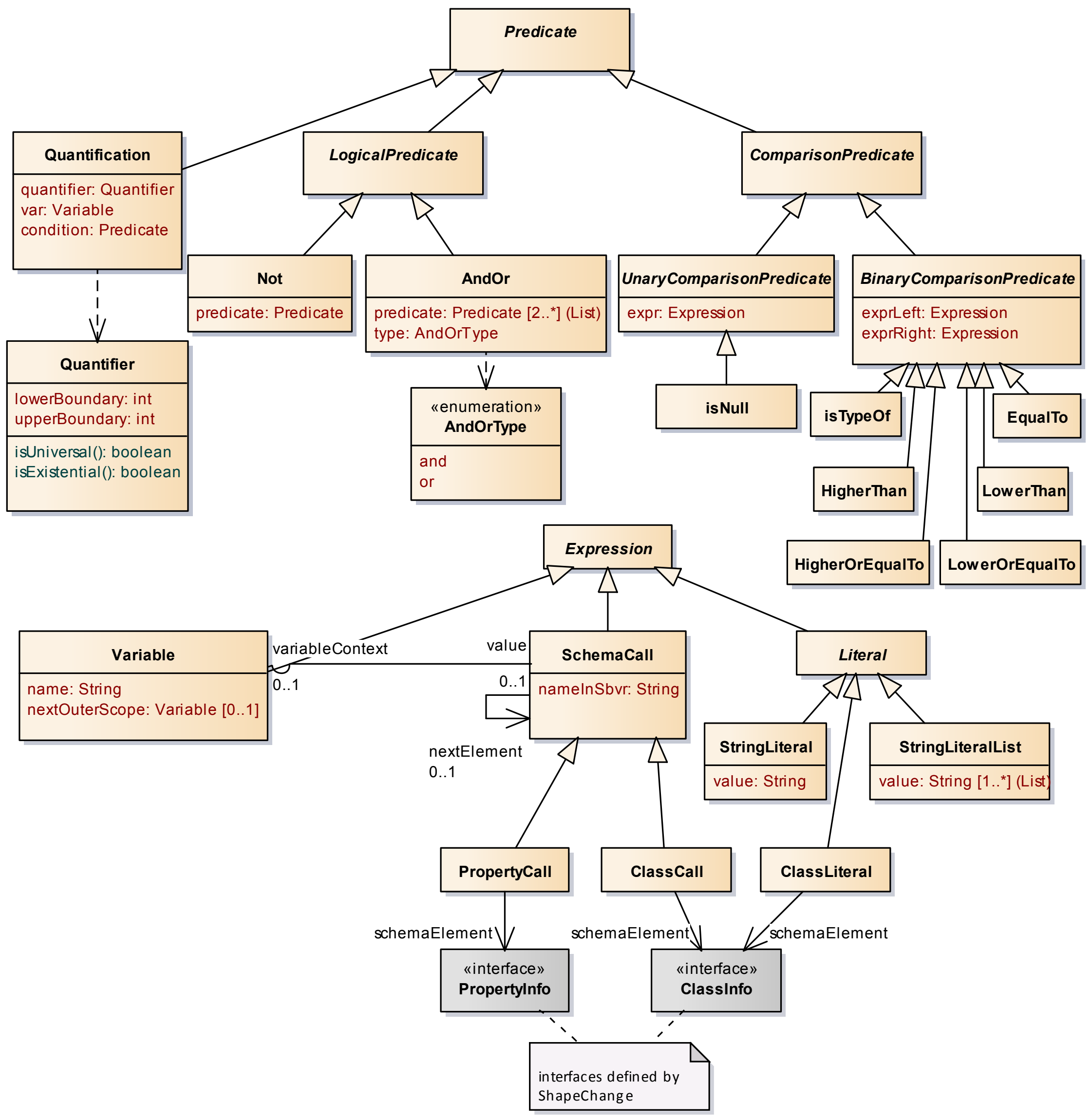


Figure 18 – Overview of the First Order Logic language supported for SBVR constraint derivation

Each sentence in the SBVR grammar is represented by an enclosing quantification. A quantification is used to check that a certain number of objects in the range of a variable satisfy a particular condition. The implementation realized in Testbed 11 supports the usual comparison operators as well as null- and type-checks. Conditions can be combined and negated using logical predicates (and, or, not).

Copyright © 2015 Open Geospatial Consortium

In the SBVR grammar, relative clauses can be used to perform a selection of the objects that must fulfill a specific condition defined by the enclosing quantification. The predicate that defines the selection is folded into the overall condition of the quantification via an 'implies' construct:

let P be the condition of the selection, and Q be the actual condition that must be satisfied, then the overall condition is:

P implies Q, which is equivalent to *not (P and not(Q))*, and thus *not(P) or Q*

The truth table is:

P	**Q**	**P implies Q**
T	T	T
T	F	F
F	T	T
F	F	T

This suits the purpose: only if the selection condition P is true and the actual condition Q is false do we have a mismatch. If the selection is not satisfied for a given element, then we are simply not interested in that element.

8.2.6 Translation to Schematron

Conversion from SBVR to Schematron is performed on the basis of a First Order Logic representation of SBVR expressions.

Though originally stemming from a natural language formulation, SBVR, the syntax of the First Order Logic representation is defined in a recursive way. Therefore the principles of translation from this syntax to another language can best be described using a recursive notation.

For a valid FOL expression x, let τ(x) denote the equivalent XPath expression. The expression x may contain free variables (explicit or implicit), which need to be treated and bound to their definition context when computing τ(x).

Table 8 describes how all particular constructs of the First Order Logic translate to XPath/Schematron.

Table 8 – Mapping of First Order Logic constructs to XPath/Schematron

First Order Logic construct (and category, if applicable)	**Textual representation**	**In words**	**Schematron translation**
Universal quantification	forall(t:x|p(t))	All members of some set x of objects or values shall fulfill predicate p.	*every τ(t) in τ(x) satisfies τ(p(τ(t)))* where t translates to a unique $ prefixed variable name, and τ(x) translates to *current()* if the quantification is at the outmost level.
Existential quantification: „at least l and at most h"	exists{l,h}(t:x|p(t))	The number of members of some set x of objects or values, which fulfill predicate p shall be between l and h.	for $var in count(for τ(t) in τ(x) return if τ(p(τ(t))) then 1 else ()) return ($var>=τ(l) and $var<=τ(h))
Existential quantification: „exactly n"	exists{n}(t:x|p(t))	The number of members of some set x of objects or values, which fulfill predicate p shall be exactly n.	for $var in count(for τ(t) in τ(x) return if τ(p(τ(t))) then 1 else ()) return ($var=n)
Existential quantification: "at most n"	exists{0..n}(t:x|p(t))	The number of members of some set x of objects or values, which fulfill predicate p shall be at most n.	for $var in count(for τ(t) in τ(x) return if τ(p(τ(t))) then 1 else ()) return $var<=n)

Copyright © 2015 Open Geospatial Consortium

First Order Logic construct (and category, if applicable)	**Textual representation**	**In words**	**Schematron translation**
Existential quantification: “at least n”	exists{n..*}(t:x|p(t))	The number of members of some set x of objects or values, which fulfill predicate p shall be at least n.	for $var in count(for τ(t) in τ(x) return if τ(p(τ(t))) then 1 else ()) return ($var>=n)
Variable access	t defined in a quantification, such as forall(t:x|p(t))	the variable name, either explicitly or implicitly provided	The variable name, preceded by ‘$’ – for example $x1, $c1. Note: this is the translation for τ(t), which is mentioned in the Schematron translations of quantifications.
Property call	x.pname	Set of object instances or values reached from the instance or set represented by x by applying property name, pname.	If pname is encoded as an XML attribute: *τ(x)/@pname* Otherwise, if pname is simple-valued or if pname is object-valued and the last segment in the schema call: *τ(x)/pname* Otherwise, if pname is object-valued and encoded inline: *τ(x)/pname/** Otherwise, if pname is realized by reference using xlink:href pointing to object instances in the same document: *//*[concat(α,@gml:id,β)=*

First Order Logic construct (and category, if applicable)	**Textual representation**	**In words**	**Schematron translation**
			τ(x)/pname/@xlink:href] α and β are a prefix and a postfix to adjust xlink:href values and gml:ids. Typically bare name references are used – hence α=# and β=empty. Otherwise (if encoded inline or by reference): *(τ(x)/pname/*)|//*[concat(α,@gml:id,β)= τ(x)/pname/@xlink:href]*

Logical infix	and(x1,...,xn)
	
	or(x1,...,xn)
		Logical combination as indicated	*τ(x1) and ... and τ(xn)*
			
			τ(x1) or ... or τ(xn)

Negation	not(x)	Logical negation of x	*not(τ(x))*
Null check	isNull(x)	Determine if the value of x is null.	If the last segment of the patch expression in *τ(x)* is encoded as an XML attribute: *not(string-length(normalize-space τ(x))))* Otherwise: *τ(x)[@xsi:nil='true']*
Equality	isEqualTo(e1,e2)	Equality of expressions e1, e2	*τ(e1)= τ(e2)* Note: This assumes that equality on sets is fulfilled if at least one pair is equal. Otherwise some more refined code generation will be necessary.

First Order Logic construct (and category, if applicable)	**Textual representation**	**In words**	**Schematron translation**
Type check	isTypeOf(x,(ClassLiteral)z)	X is checked for complying with the type y identified by class literal z.	τ(x)[name()='T_1' or … or name()='T_i'], where T_k is the qualified name of one of the concrete derivations of y, including y, if it is not abstract (names of abstract types are ignored).
String literal	'xxxxx'		same
Numeric literal	123 or 3.1415		same
String literal list	('abc','def',...)	List of 'names'	same
Class literal	class name (e.g. AirportHeliport)	name of the class	name of the class identified by the class literal

The following sections describe specific aspects of the translation.

8.2.6.1 Recognition of AIXM Extension Elements

Classes and properties declared in an AIXM extension schema are translated using the target namespace and preferred prefix declared by that schema. The relevant schema information has been preserved by ShapeChange in the AIXM schema merging process step (see 8.2.3).

In addition, properties of AIXM features that have been specified in an extension receive a slightly modified translation:

- Let *pname* be the name of an extension property, and
- let *nsP* be the preferred namespace prefix for the schema in which the property has been declared, and
- let *nsFT* be the preferred namespace prefix for the schema in which the feature type that owns the property, then:

- if pname is simple-valued or if pname is object-valued and the last segment in the schema call:
  - *τ(x)/nsFT:extension/*/nsP:pname*
- Otherwise, if pname is object-valued and encoded inline:
  - *τ(x)/nsFT:extension/*/nsP:pname/**
- Otherwise, if pname is encoded by reference using xlink:href pointing to object instances in the same document:
  - *//*[concat(α,@gml:id,β)= τ(x)/nsFT:extension/*/nsP:pname/@xlink:href]*
  - α and β are a prefix and a postfix to adjust xlink:href values and gml:ids. Typically bare name references are used – hence α=# and β=empty.
- Otherwise (if encoded inline or by reference):
  - *(τ(x)/nsFT:extension/*/nsP:pname/*)|//*[concat(α,@gml:id,β)= τ(x)/nsFT:extension/*/nsP:pname/@xlink:href]*

8.2.6.2 Support for Feature References in Schematron Code

The SBVR Profile for AIXM states the following:

> *According to the AIXM Feature Identification and Reference document, associations between features can be implemented with abstract or local references. In the current version, the Schematron code provided assumes that all associations are encoded as local references (xlink:href="#..." referring to the gml:id value of the target feature).*

At present, the Schematron derivation tool also supports only local references.

It would be possible to extend the Schematron translation to also support abstract references. However, a pre-processing step for the data that shall be validated could also ensure that abstract references are resolved to local ones, prior to Schematron validation. That would allow the Schematron code to continue only using local references, and thus prevent adding another level of complexity in the resulting code. The validation service or the enrichment service might be suitable candidates for this kind of pre-processing.

8.2.6.3 Realization of 'of-type' operator

The SBVR grammar of the tool (8.2.5.1) supports an 'of-type' operator in predicates. This operator checks that a given object has a specific type, which is specified by a single or a list of class names. Whenever such a class name identifies a supertype in an inheritance hierarchy, the tool automatically includes all known (non-abstract) subtypes in the type check – in addition to the supertype itself (if it is not abstract).

When translated to Schematron, the ‘of-type’ operator is realized as a series of QName checks.

Example

Given the following schema:

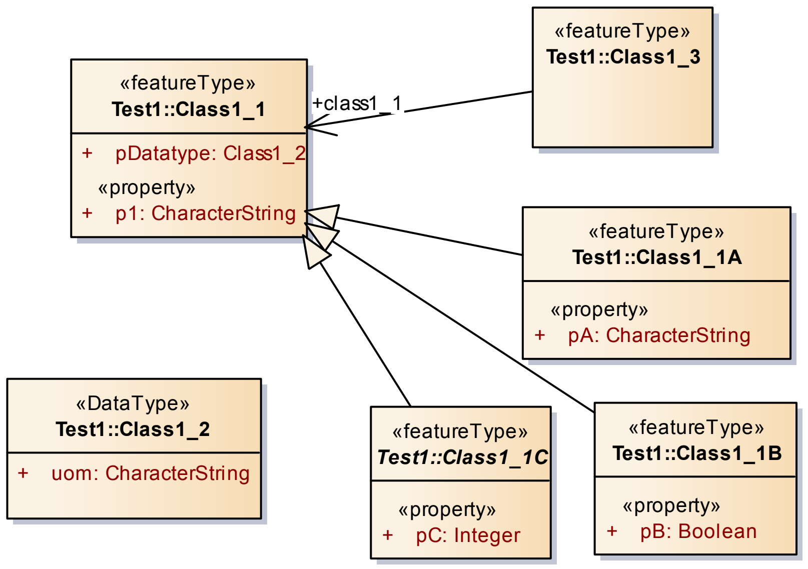


Figure 19 – Conceptual schema example for ‘of-type’ operator translation to Schematron

Given the following rules:

1. Each Class1_3 shall have class1_1 of-type 'Class1_1A' or 'Class1_1B'.
2. Each Class1_3 shall have class1_1 of-type 'Class1_1'.

The tool produces the following Schematron rule / assertions:

Listing 1 – Schematron for rules using ‘of-type’ operator

```
<rule context="core:Class1_3">
  <assert id="1" test="every $x1 in current() satisfies (for $c1 in
count(for $x2 in $x1/core:timeSlice/*/core:class1_1 return if
(//*[concat('#',@gml:id)=$x2/@xlink:href][name()='core:Class1_1A'] or
//*[concat('#',@gml:id)=$x2/@xlink:href][name()='core:Class1_1B']) then
1 else ()) return ($c1 &gt;= 1))">Each Class1_3 shall have class1_1 of-
type 'Class1_1A' or 'Class1_1B'.</assert>
  <assert id="2" test="every $x1 in current() satisfies (for $c1 in
count(for $x2 in $x1/core:timeSlice/*/core:class1_1 return if
(//*[concat('#',@gml:id)=$x2/@xlink:href][name()='core:Class1_1A' or
name()='core:Class1_1B' or name()='core:Class1_1']) then 1 else ())
return ($c1 &gt;= 1))">Each Class1_3 shall have class1_1 of-type
'Class1_1'.</assert>
</rule>
```

As we can see, the translation of the first rule performs a check against the QNames of Class1_1A and Class1_1B. The XPath created for the second rule also checks for the QName of Class1_1 itself, but not of its subtype Class1_1C because that is abstract.

8.2.6.4 Realization of null-checks

The null check for a given property is realized in two different ways, depending on how the property is encoded in XML:

1. Propert encoded as XML element: here the XPath expression applies a check on the xsi:nil XML attribute.
   - Example: […] for $x3 in $x1/core:timeSlice/*/core:pDatatype/@uom return if (not(not(string-length(normalize-space($x3))))) then 1 else () […]
2. Property encoded as XML attribute: in this case the XPath expression generated by the tool checks that the attribute has a non-empty string value, trimming leading and trailing whitespace.
   - Example: […] for $x2 in $x1/core:timeSlice/*/core:p1 return if (not($x2[@xsi:nil='true'])) then 1 else () […]

Copyright © 2015 Open Geospatial Consortium

9 Accomplishments

- The open source tool ShapeChange has been extended to load and parse SBVR constraints and to automatically derive Schematron code from them. The process supports SBVR constraints for both AIXM schemas as well as ISO 19109 application schemas.
- A model transformation has been realized, to support the AIXM extension mechanism.
- More than 60% of the AIXM business rules have been translated to Schematron rules, showing that the implementation of constraints expressed using SBVR can be automated to a significant extent. The translation rate can be increased even further through future work as described in section 1.3.
- The generated Schematron rules have successfully been integrated in a validation web service, and successfully tested there using a test dataset.

Annex A: Revision history

Date	Release	Editor	Primary clauses modified	Description
2015-05-13	0.1	Johannes Echterhoff	all	first version of complete ER
2015-Jun-11	Pub	Carl Reed	Numerous	Prepare for publication

Copyright © 2015 Open Geospatial Consortium

Bibliography

[1] (2013) Rob van Haarst: *SBVR Made Easy – Business Vocabulary and Rules as a Critical Asset*, Publisher: ConceptualHeaven