estec

European Space Research
and Technology Centre
Keplerlaan 1
2201 AZ Noordwijk
The Netherlands
T +31 (0)71 565 6565
F +31 (0)71 565 6040
www.esa.int

DOCUMENT

Fact based conceptual modelling in support to requirement engineering An example

Prepared by Serge Valera
Reference ESA/ESTEC/TEC-SWM/10.xxx
Issue 0
Revision 1
Date of Issue 10/10/2014
Status Draft
Document Type TN
Distribution

APPROVAL

Title	
Issue 0	**Revision** 1
Author	**Date** 10/10/2014
Approved by	**Date**

CHANGE LOG

Reason for change	Issue	Revision	Date

CHANGE RECORD

Issue 0	Revision 1		
Reason for change	**Date**	**Pages**	**Paragraph(s)**

1 INTRODUCTION

The purpose of this technical note is to show how fact based conceptual modelling can support the requirement engineering process.

The example used for that demonstration is taken from the Appendix A of the "space system data repository" ECSS technical memorandum i.e. ECSS-E-TM-10-23A.

This appendix provides some definitions of what requirements could be. The subset of these definitions used for the demonstration is listed in chapter 2. *It is noted that the requirement data model of the ECSS-E-TM-10-23A has not been consolidated by the technical memorandum authors nor validated by ECSS.*

In chapter 3, each chapter 2 definition is challenged.

For each chapter 2 definition or group of definitions, a formal (i.e. logic based) representation is proposed. These formal representations are just representing the understanding of their author and still need to be validated by ECSS, i.e. the entity responsible of "standardizing" (for ECSS) what requirements are.

The need to have a standardized definition of what requirements are is recognised by the ECSS. Would ECSS decide to put some efforts in the standardizing of the requirement model, this technical note will be given as input to that standardization work.

In chapter 3, two formal representation's views (a graphical view and a controlled natural language view) are given. These 2 views are produced by the NORMA software tool that complies with the Object Role Modelling (ORM) FBM dialect (refer to www.orm.net, see for example:

- http://orm.net/pdf/ORM2_TechReport1.pdf
- http://orm.net/pdf/ORM2_TechReport2.pdf

The fact based conceptual modelling objective is:

- to capture the "universe of discourse", meaning *all* the WHAT without any assumption, without any implicit understanding;
- to validate the resulting conceptual model (the requirements' specification) before any development.

This objective is time consuming and requires much more requirements' engineering efforts to be performed before contracting any development than commonly done. That requirements' engineering effort increase is largely compensated by the quality of the resulting requirements' specifications that provide smooth later development with a greater chance to produce solutions that fully satisfy the stakeholders' needs.

Requirements' engineering with fact based conceptual modelling addresses:

- the vocabulary of the universe of discourse (the UoD specific terms and related definitions),
- the UoD types' rules (e.g. Each satellite shall be designed for launch compatibility with at least two launchers), and
- the UoD instances' rules (e.g. The EarthCare satellite shall be designed for launch compatibility with the Soyuz launcher. The EarthCare satellite shall be designed for launch compatibility with the Zenit launcher).

Depending on what is to be developed, the requirements' specification includes types' rules related requirements or instances' rules related requirements or both.

Formally capturing the WHAT can enable "automating" (part of) the product developments.

Taking as an example the ECSS-E-TM-10-23A "requirement data model", let's assume that someone wishes to develop a database system that provides means to manage (create and maintain) products related requirements and to exchange that products related requirements with others. one can transform conceptual models into e.g.:

- A relational model that can be used to structure the internal repository of a relational database management system (RDBMS) software application (used to store all requirements), see the Appendix A-2 automatically generated relational model.
- An exchange model that can be used to exchange data between the RDBMS software application and other systems (using e.g. XSD to exchange requirements), see the Appendix A-3 automatically generated hierarchical XSD model.

2 THE ECSS-E-TM-10-23A APPENDIX A USE CASE

A.1 Requirement data model

a. Within the context of a system element:
   1. a requirement is uniquely identified by a name.
   2. a requirement is uniquely identified by an identifier.

b. A requirement is defined by one and only statement.

c. A requirement is defined by zero or more alternative representations.

NOTE An alternative representation could be a translation of the statement into another language such as Russian or Japanese, or a formal mathematical representation of the statement such as a MathML computer-interpretable mathematical expression of a quantitative requirement, e.g. $m_{equipment} \leq 12\text{kg}$

d. A requirement is associated with zero or more annotations.

e. A requirement is rationalised by zero or more justifications.

f. A requirement is classified by zero or more requirement categories.

g. The possible requirement categories include the following:
   1. Functional
   2. Mission
   3. Interface
   4. Environmental
   5. Physical
   6. Operational
   7. Human factor
   8. Logistics support
   9. Product assurance
   10. Configuration
   11. Design
   12. Verification
   13. Performance
   14. Implementation
   15. RAMS

h. Additional requirement categories can be added at mission or at customer level.

i. A requirement is associated with at most one criticality category justified by zero or more criticality assessment reports.

j. The criticality categories associated with requirements are mission-specific.

k. A requirement is verified in accordance with one or more methods.

l. The possible verification methods for a requirement include at least the following values:
   1. analysis;
   2. inspection;

3. review of design;
4. test.

m. A requirement is verified according to zero or one logic.

n. The verification logic of a requirement is one of the following values:

1. open, i.e. the verification level and approach are not defined;
2. defined, i.e. the verification level and approach are defined;
3. agreed, i.e. the verification level and approach are agreed by the supplier and customer.

o. A requirement is verified by zero or more observations.

p. An observation is time tagged by a stamp.

q. An observation provides a status.

r. An observation is detailed by zero or one report (i.e. a free text).

s. The possible verification statuses are:

1. draft, i.e. the verification is made but its associated report is not verified;
2. agreed, i.e. the verification is made and the associated report has been verified by the engineering responsible;
3. approved, i.e. the verification state has been approved by the verification board;
4. verified, i.e. the verification states have been accepted by the customer.

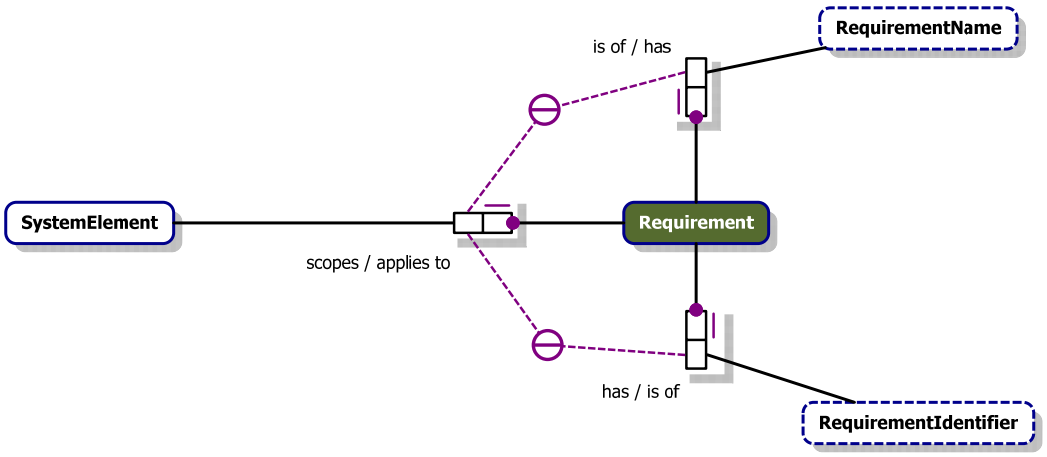
t. A requirement is derived from at most one higher-level requirement.

u. The higher-level requirement may either be related to the same system element or to another system element.

3 TOWARD FORMALIZING THE ECSS REQUIREMENT MODEL

3.1

a. Within the context of a system element:
   1. a requirement is uniquely identified by a name.
   2. a requirement is uniquely identified by an identifier.



Requirement applies to system element.

1. Each requirement applies to exactly one system element.
2. It is possible that some system element scopes more than one requirement.

Requirement has requirement name.

3. Each requirement has exactly one requirement name.
4. It is possible that some requirement name is of more than one requirement.
5. *[external uniqueness constraint]* For each system element and requirement name, at most one requirement applies to that system element and has that requirement name.

Requirement has requirement identifier.

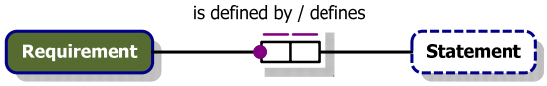
6. Each requirement has exactly one requirement identifier.
7. It is possible that some requirement identifier is of more than one requirement.
8. *[external uniqueness constraint]* For each system element and requirement identifier, at most one requirement applies to that system element and has that requirement identifier.

Rules 1 to 8 can be derived from a.1 and a.2.

The stakeholders should check one by one each rule and confirm that this is exactly what they have in mind.

3.2

b. A requirement is defined by one and only statement.



Requirement is defined by statement.

9. Each requirement is defined by exactly one statement.
10. Each statement defines at most one requirement.

Requirement b is wrongly written. It should read "one and only one" or "exactly one".

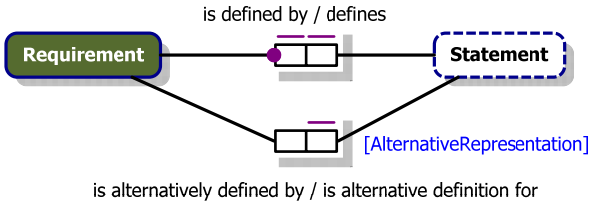
Rule 9 is equivalent to b.

Rule 10 was not specified. *It should be confirmed by the stakeholders.*

3.3

c. A requirement is defined by zero or more alternative representations.

NOTE An alternative representation could be a translation of the statement into another language such as Russian or Japanese, or a formal mathematical representation of the statement such as a MathML computer-interpretable mathematical expression of a quantitative requirement, e.g. $m_{equipment} \leq 12\text{kg}$



Requirement is alternatively defined by statement.

11. Each statement is alternative definition for at most one requirement.
12. It is possible that some requirement is alternatively defined by more than one statement.

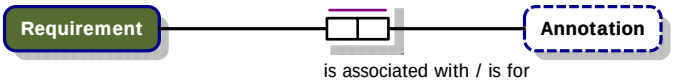
The wording "alternative representations" and the reference to languages within the note make thinking that b. and c. addresses a generic concept of "statement", a statement having different representations according to the community and language that is addressed. This generic concept needs to be further developed.

Rule 11 was not specified. *It should be confirmed by the stakeholders.*

Rule 12 is equivalent to c.

3.4

d.	A requirement is associated with zero or more annotations.



Requirement is associated with annotation.

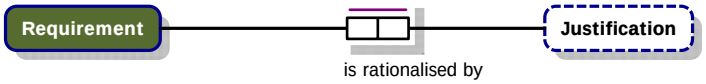
13. It is possible that some requirement is associated with more than one annotation and that some annotation is for more than one requirement.
14. In each population of requirement is associated with annotation, each requirement, annotation combination occurs at most once.

Requirement d. is included within the Rule 13. The rule 13 is more precise since it also introduces the fact that the same annotation can be used to annotate several requirements. That needs to be confirmed by the stakeholders.

Rule 14 explicitly specifies something that should be implicitly understood by everyone when reading d.

3.5

e.	A requirement is rationalised by zero or more justifications.



Requirement is rationalised by justification.

15. It is possible that some requirement is rationalized by more than one justification and that for some justification, more than one requirement is rationalized by that justification.
16. In each population of requirement is rationalized by justification, each requirement, justification combination occurs at most once.

Requirement e. is included within the Rule 15. The rule 15 is more precise since it also introduces the fact that the same justification can be used to rationalize several requirements. That needs to be confirmed by the stakeholders.

Rule 14 explicitly specifies something that should be implicitly understood by everyone when reading e.

3.6

f. A requirement is classified by zero or more requirement categories.

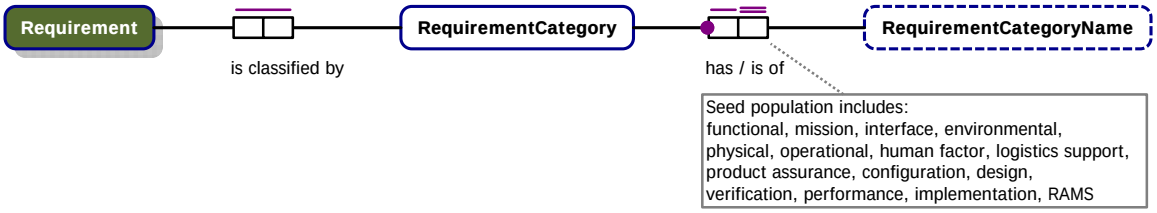
g. The possible requirement categories include the following:

Functional – Mission – Interface – Environmental – Physical

Operational – Human factor – Logistics support – Product assurance

Configuration – Design – Verification – Performance – Implementation – RAMS

h. Additional requirement categories can be added at mission or at customer level.



Requirement is classified by requirement category.

17. It is possible that some requirement is classified by more than one requirement category and that for some requirement category, more than one requirement is classified by that requirement category.
18. In each population of requirement is classified by requirement category, each requirement, requirement category combination occurs at most once.

Requirement category has requirement category name.

19. Each requirement category has exactly one requirement category name.
20. Each requirement category name is of at most one requirement category.

Requirement f. is included within the Rule 17. The rule 17 also explicitly specifies that a requirement category can be used to classify more than one requirement. This add-on is implicitly understood by all stakeholders.

Rule 18 explicitly specifies something that is implicitly understood by all stakeholders.

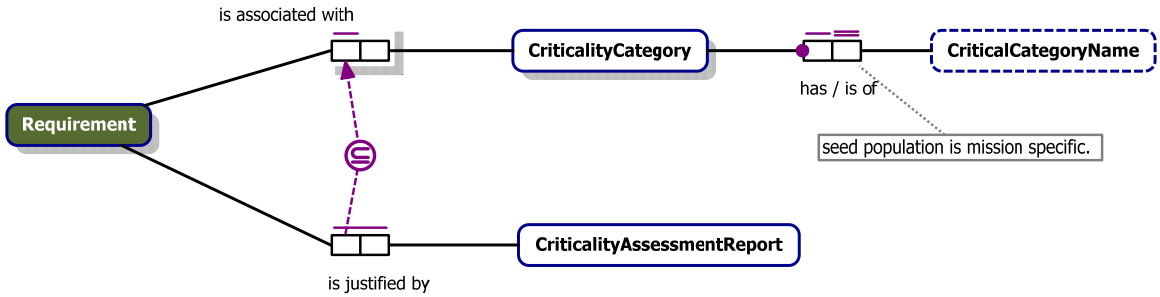
Rules 19 and 20 address g. and h.

The FBM tool used to produce this document does not provide currently the adequate functionality to model seed populations.

The wording "Seed population includes" implies that the seed population can be extended.

3.7

i.	A requirement is associated with at most one criticality category justified by zero or more criticality assessment reports.
j.	The criticality categories associated with requirements are mission-specific.



Requirement is associated with criticality category.

21. Each requirement is associated with at most one criticality category.
22. It is possible that more than one requirement is associated with the same criticality category.

Criticality category has critical category name.

23. Each criticality category has exactly one critical category name.
24. Each critical category name is of at most one criticality category.

Requirement is justified by criticality assessment report.

25. It is possible that some requirement is justified by more than one criticality assessment report and that for some criticality assessment report, more than one requirement is justified by that criticality assessment report.
26. In each population of requirement is justified by criticality assessment report, each requirement, criticality assessment report combination occurs at most once.
27. *[external subset constraint]* If some requirement is justified by some criticality assessment report then that requirement is associated with some criticality category.

Requirement i. is not elementary and should be expressed in 2 requirements, one requiring the criticality category association (see rule 21), the other one the criticality assessment reports (see rules 25 and 27).

Rules 22, 23, 24 and 25 explicitly specify a stakeholder implicit understanding. These 2 rules also address j. *As said, the FBM tool used to produce this document does not provide currently the adequate functionality to model seed populations. The wording "Seed population is mission specific" implies that the seed population needs to be specified.*

The second part of rule 25 needs to be confirmed by the stakeholders.

3.8

k. A requirement is verified in accordance with one or more methods.

l. The possible verification methods for a requirement include at least the following values:

1. analysis;
2. inspection;
3. review of design;
4. test..

Requirement is verified in accordance with verification method.

28. It is obligatory that each requirement is verified in accordance with some verification method.
29. It is possible that some requirement is verified in accordance with more than one verification method and that some verification method is required for more than one requirement.
30. In each population of requirement is verified in accordance with verification method, each requirement, verification method combination occurs at most once.

Verification method has verification method name.

31. Each verification method has exactly one verification method name.
32. Each verification method name is of at most one verification method.

Rule 28 is lighter than requirement k. It acknowledges the fact that a requirement can be specified before its verification methods. This is expressed by the "it is obligatory that…" part of the rule 28. *see also the verification logic that further constraints rule 28 for establishing when a verification logic can be said to be defined.*

The first part of rule 29 is already covered by rule 28.

The second part of rule 29, rule 30 are implicitly understood.

The FBM tool used to produce this document does not provide currently the adequate functionality to model seed populations. The wording "Seed population includes" implies that the seed population can be extended.

3.9

m. A requirement is verified according to zero or one logic.

n. The verification logic of a requirement is one of the following values:

1. open, i.e. the verification level and approach are not defined;
2. defined, i.e. the verification level and approach are defined;
3. agreed, i.e. the verification level and approach are agreed by the supplier and customer..

The requirements m and n are confusing. They are also inconsistent.

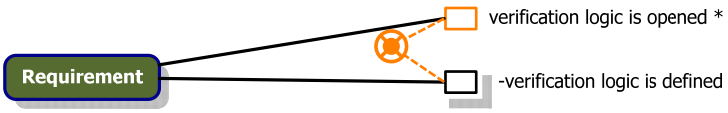
- Is the "no logic" verification stated in requirement m equivalent to the "open" verification logic of requirement n.1?
- what is meant by verification level?
- what is meant by verification approach?

In the following, the following assumptions are taken:

- the "verification level" means one of the "verification method" introduced in requirement k.
- "open verification logic" is the same than having "no logic" defined.

These assumptions need to be validated with the stakeholders.

The concept of verification approach needs to be further developed with the support of the stakeholders.



Requirement verification logic is defined.

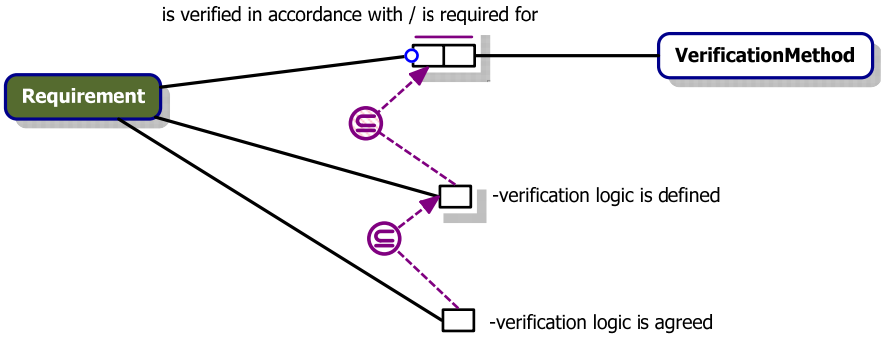
33. In each population of requirement verification logic is defined, each requirement occurs at most once.

Rule 34is derived (unary fact type : requirement verification logic is open), i.e. a consequence of not having defined the verification logic of a requirement.

34. Requirement verification logic is opened if and only if it is not true that (that requirement verification logic is defined).

Rule 35 is also derived i.e. a consequence of the rule 34.

35. *[exclusive-or constraint]* For each requirement, exactly one of the following holds:
    that requirement verification logic is opened;
    that requirement verification logic is defined.



Requirement is verified in accordance with verification method. *Refer to rules 28, 29 and 30*

Requirement verification logic is defined.

36. In each population of requirement verification logic is defined, each requirement occurs at most once.
37. *[external subset constraint]* If some requirement verification logic is defined then that requirement is verified in accordance with some verification method.

Requirement verification logic is agreed.

38. In each population of requirement verification logic is agreed, each requirement occurs at most once.
39. *[external subset constraint]* If some requirement verification logic is agreed then that requirement verification logic is defined.

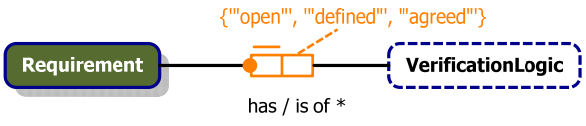
Rule 36 addresses n.2.

Rule 37 ensures that some verification methods are specified for a requirement before claiming that its verification logic is defined.

Rule 38 addresses n.3

Rule 39 explicitly addresses the need for a verification logic to be defined prior to be agreed, i.e. implicitly understood by all stakeholders.

Requirement n. introduces the concept of "verification logic" that has 3 possible values.
To map this stakeholder's view, the relation between a requirement and its verification logic can be derived from rules 33 to 39 as follows.



Requirement has verification logic

40. Each requirement has exactly one verification logic.
41. It is possible that some verification logic is of more than one requirement.

42. *Requirement has verification logic if and only if
    that requirement verification logic is opened where verification logic = "open"
    or that requirement verification logic is defined where verification logic = "defined"
    or that requirement verification logic is agreed where verification logic = "agreed".
43. The possible values of verification logic in requirement has verification logic are '"open"', '"defined"', '"agreed"'.

Rules 40 and 41 addresses requirement m taking into account the assumption given above.

Rule 43 addresses requirement n.

The verification logic property of a requirement (i.e. to be "open", "defined", "agreed") is obtained by the rule 42 that express how to derive the verification logic from the rules 33, 36 and 38.

3.10

o. A requirement is verified by zero or more observations.

p. An observation is time tagged by a stamp.

q. An observation provides a status.

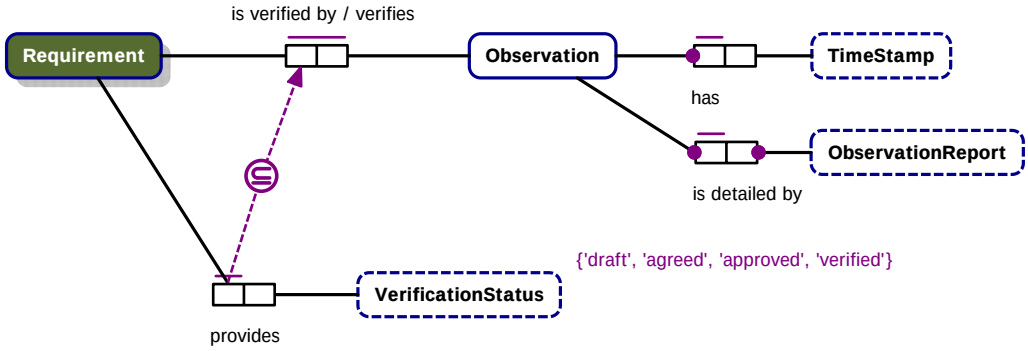
r. An observation is detailed by zero or one report (i.e. a free text).

s. The possible verification statuses are:

1. draft, i.e. the verification is made but its associated report is not verified;
2. agreed, i.e. the verification is made and the associated report has been verified by the engineering responsible;
3. approved, i.e. the verification state has been approved by the verification board;
4. verified, i.e. the verification states have been accepted by the customer.

The requirements q and s are confusing, i.e.:

- Is the status mentioned in q. corresponding to the verification status of s?
  Below, the assumption is taken that q is not a valid requirement, so it needs to be deleted.
- Are the possible verification statuses addressing an observation or a requirement?
  Below, the assumption is taken that the verification status addresses a requirement and not an observation.



Requirement is verified by observation.

44. It is possible that some requirement is verified by more than one observation and that some observation verifies more than one requirement.
45. In each population of requirement is verified by observation, each requirement, observation combination occurs at most once.

Observation has time stamp.

46. Each observation has exactly one time stamp.
47. It is possible that more than one observation has the same time stamp.

Observation is detailed by observation report.

48. Each observation is detailed by exactly one observation report.
49. For each observation report, some observation is detailed by that observation report.
50. It is possible that more than one observation is detailed by the same observation report.

Requirement provides verification status.

51. Each requirement provides at most one verification status.
52. It is possible that more than one requirement provides the same verification status.
53. The possible values of verification status are 'draft', 'agreed', 'approved', 'verified'.
54. If some requirement provides some verification status then that requirement is verified by some observation.

Rule 44 addresses requirement o.

Rule 46 addresses requirement p.

Requirement q. is assumed to be wrong and as such not modelled, i.e. waiting for clarification from the stakeholders.

Rule 48 addresses requirement r. Rule 48 is stronger than requirement r since an observation cannot just be expressed by a timestamp, i.e. without the observation report, there is no observation.

Rules 51 to 54 addresses requirement s.

It is noted that, similar to rules 40 to 43, the rule 51 to 54 could have been modelled as derivation. Currently this is not done since there is a need for some clarifications with the stakeholders. Once the clarifications will be given, the model would need to be updated and what is currently asserted with rule 51 to 54 could be derived.

The key issue in this "by step" modelling approach is that the customers can validate rules 51 to 54 independently of whether that knowledge is asserted or derived.

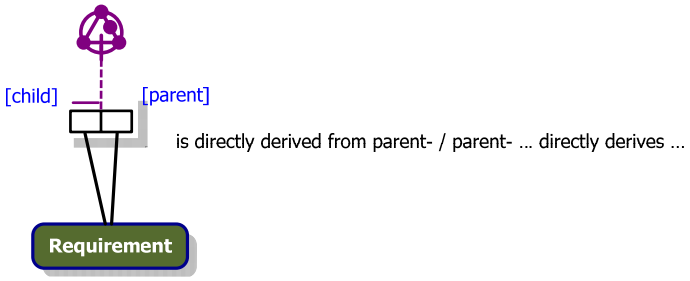
3.11

t.	A requirement is derived from at most one higher-level requirement.

Requirement t is not really correct. For example, a requirement A can be directly derived from a higher-level requirement B that itself is directly derived from another higher-level requirement C. As such requirement A can be said to be derived from more than one higher-level requirement.

There is a need to introduce the concept of "directly derived" as a mean to express that a requirement at level n is derived from at most one requirement at level n-1.

In the following this modelling issue is followed.



Requirement is directly derived from parent requirement.

55. Each requirement is directly derived from at most one parent requirement.
56. It is possible that some parent requirement directly derives more than one requirement.
57. No requirement may cycle back to itself via one or more traversals through requirement is directly derived from parent requirement.
58. If $requirement_1$ is directly derived from some parent $requirement_2$ then it is not true that $requirement_1$ is indirectly related to $requirement_2$ by repeatedly applying this fact type.

Rule 55 addresses requirement t

Rule 56 explicitly specifies something that is implicitly understood by all stakeholders.

Rules 57 and 58 ensure that the requirement organizations in levels reflect hierarchies of requirements, i.e. tree structures, i.e. no cycle, no transitivity.

3.12

u. The higher-level requirement may either be related to the same system element or to another system element.

Requirement u is not a good requirement since it constraints nothing.

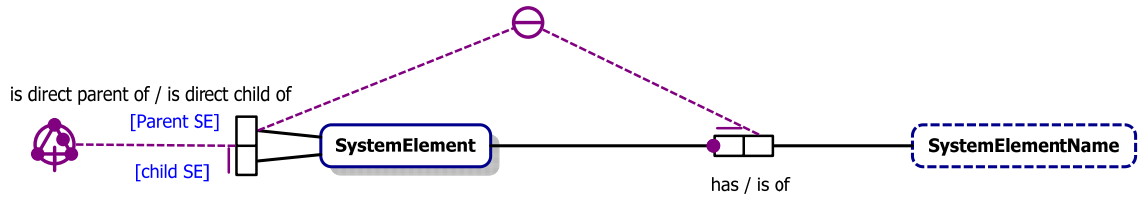
However, it addresses the fact that the stakeholders need to specify requirements for more than one system element.

Focusing more on this issue (taking into account the ECSS-E-TM-10-23), one can assume that referring to "system element" the stakeholders have in mind that a system element represents e.g. a product within a product tree.

This concept is not developed in the Annex A.

A proposal is made below to organize the system elements as nodes of product tree structures.

This is just a proposal that needs to be validated by the stakeholders.



System element is direct parent of system element.

59. Each system element is direct child of at most one system element.
60. It is possible that some system element is direct parent of more than one system element.
61. No system element may cycle back to itself via one or more traversals through system element is direct parent of system element.
62. If system element$_2$ is direct child of some system element$_1$ then it is not true that system element$_2$ is indirectly related to system element$_1$ by repeatedly applying this fact type.

System element has system element name.

63. Each system element has exactly one system element name.
64. It is possible that some system element name is of more than one system element.
65. For each system element$_1$ and system element name, at most one system element is direct child of that system element$_1$ and has that system element name.

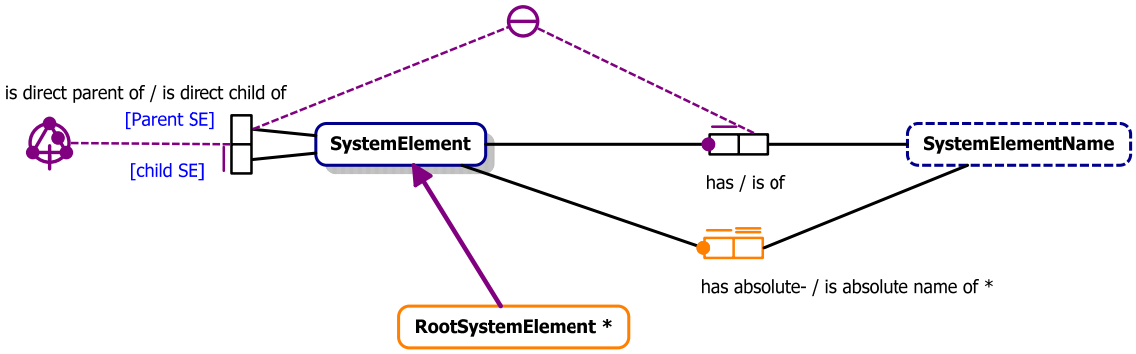
Rules 59 to 62 specify that the system elements are hierarchically organized (in tree structures)

Rules 63 to 65 specify that each system element is identified by a name that is relative to its parent system element (e.g. such as the file names within a directory)

The above model seems incomplete:

- What are the "root" system elements?
- What is the absolute name of a system element?

This information can be obtained by derivation as describe below.



Root system element

66. Each root system element is by definition some system element$_1$ where that system element$_1$ is direct child of no system element.

System element$_1$ has absolute system element name$_1$

67. Each system element has exactly one absolute system element name.
68. Each system element name is absolute name of at most one system element.
69. System element$_1$ has absolute system element name$_1$ if and only if that system element$_1$
    (is some root system element and has that system element name$_1$)
    or (is direct child of some system element$_2$ that has some absolute system element name$_2$ and that system element$_1$ has some system element name$_3$)
    where system element name$_1$ = concat(system element name$_3$, " of ", system element name$_2$).

Annex A
Logical Modelling views

A.1 Introduction

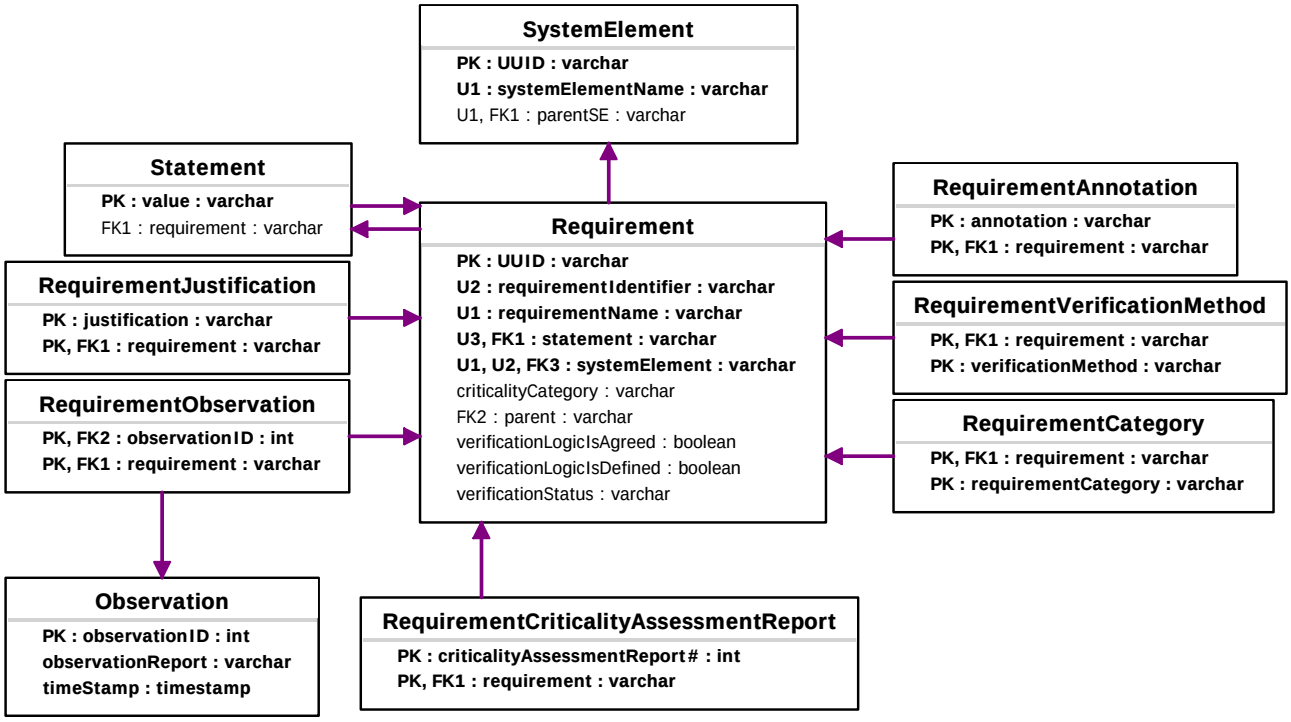
Fact based conceptual models can be transformed into logical and physical models.

In this annex, 2 of such transformations are shown, i.e.

- A relational model that can be used to structure the internal repository of a relational database management system (RDBMS) software application (used to store all requirements), see the Appendix A-2 automatically generated relational model.
- An exchange model that can be used to exchange data between the RDBMS software application and other systems (using e.g. XSD to exchange requirements), see the Appendix A-3 automatically generated hierarchical XSD model.

These 2 models are not valid since the conceptual model presented in this document has not been validated.

A.2 A possible relational model

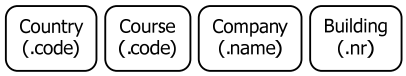
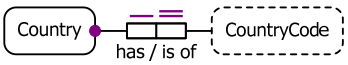
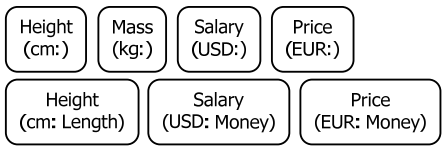
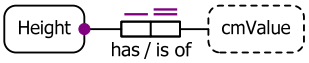
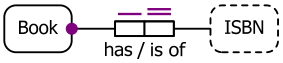
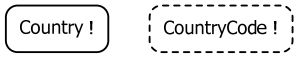
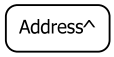
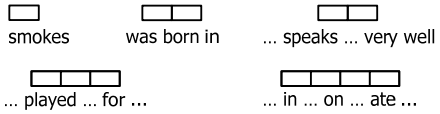
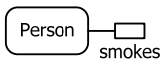
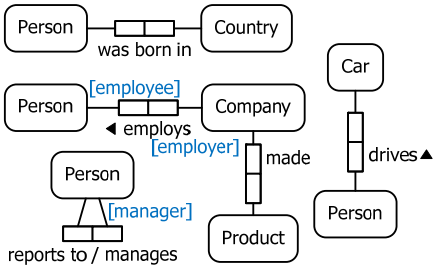


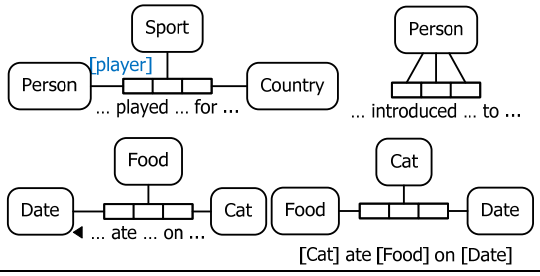
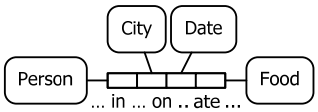
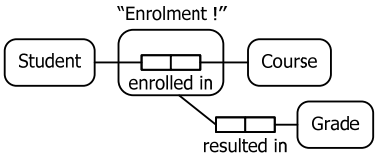
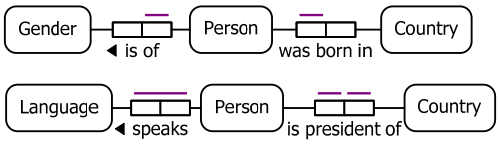
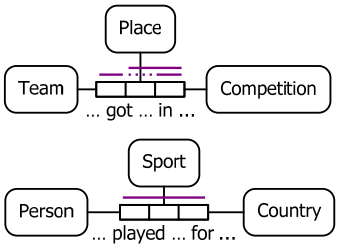
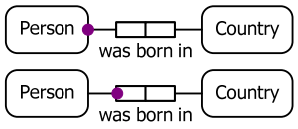
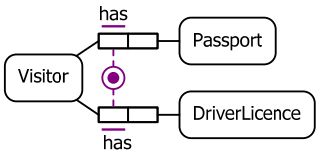
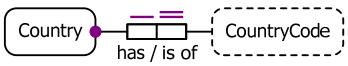
A.3 A possible hierarchical XSD model

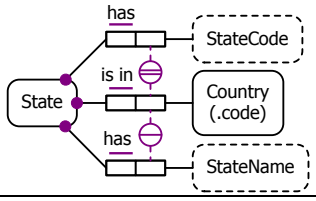
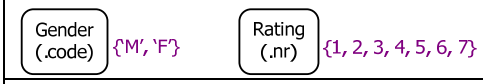
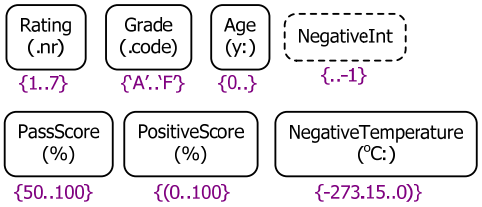
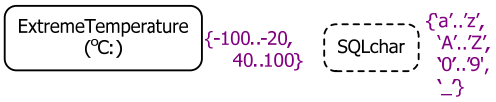
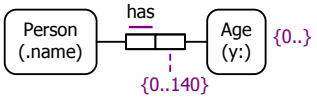
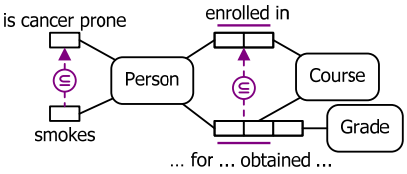
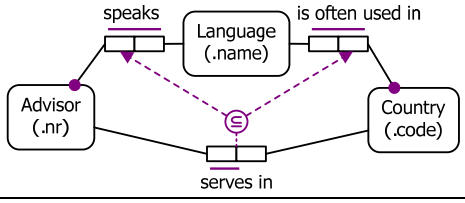
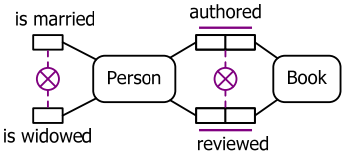
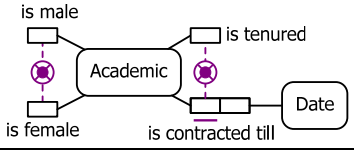
ECSS-E-TM-10-23A Requirement model - A FBM use case.xsd

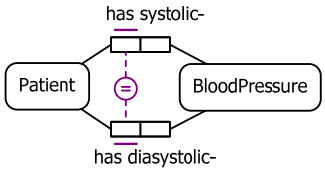
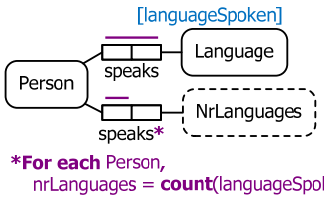
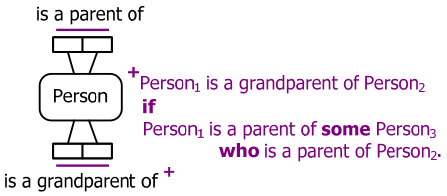
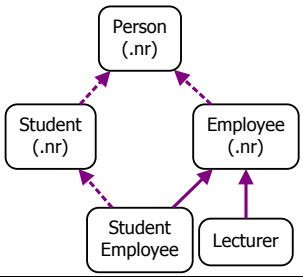
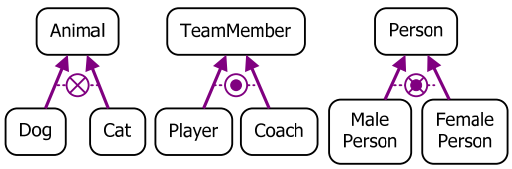
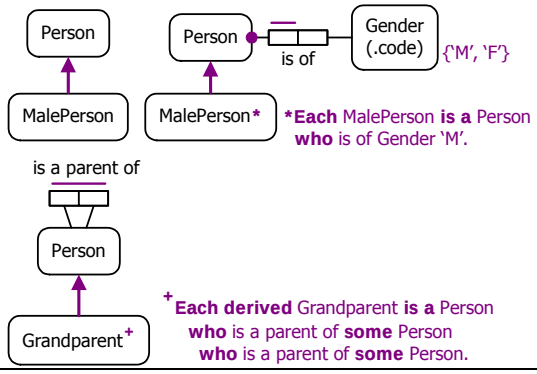
ORM 2 Graphical Notation

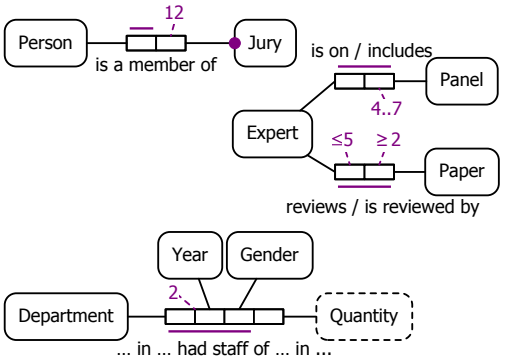
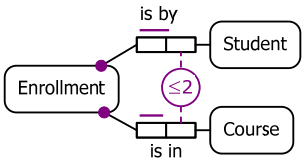
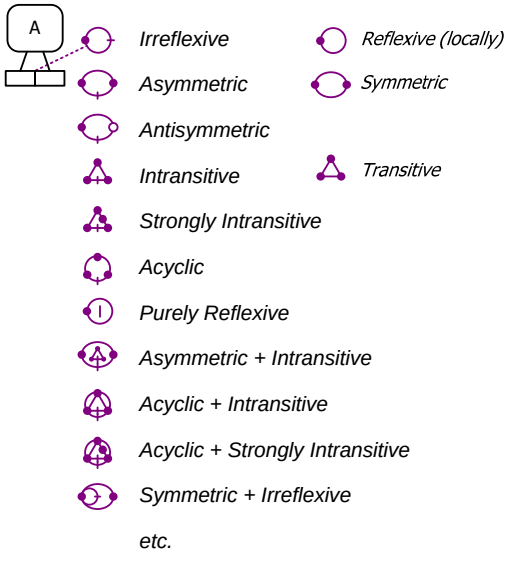
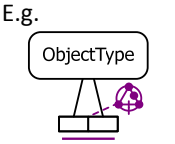
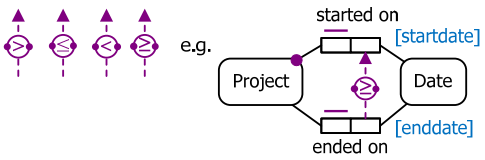
Terry Halpin

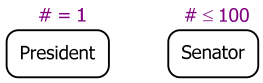
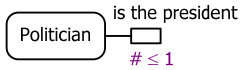
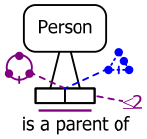
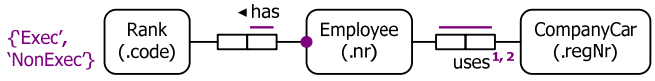
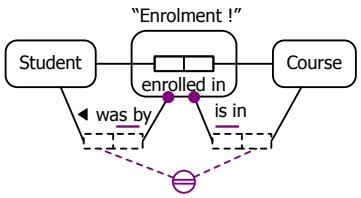
Construct	*Examples*	*Description/Notes*
Entity Type		Named soft rectangle, named hard rectangle, or named ellipse. The soft rectangle shape is the default.
Value Type		Named, dashed, soft rectangle (or hard rectangle or ellipse).
Entity type with popular reference mode		Abbreviation for injective reference relationship to value type, e.g. 
Entity type with unit-based reference mode		Abbreviation for reference type, e.g.  Optionally, unit type may be displayed.
Entity type with general reference mode		Abbreviation for reference type, e.g. 
Independent Object Type		Instances of the type may exist, without playing any elementary fact roles
External Object Type		This notation is tentative (yet to be finalized)
Predicate (unary, binary, ternary, etc.)		Ordered set of 1 or more role boxes with at least one predicate reading in mixfix notation. If shown, object placeholders are denoted by "...". If placeholders are not shown, unaries are in prefix and binaries are in infix notation.
Duplicate type or predicate shape		If an object type or predicate shape is displayed more than once (on the same page or different pages) it is shadowed.
Unary fact type		The smokes role may be played by instances of the Person object type
Binary fact type		By default, predicate readings (binary or longer) are read left-to-right or top-to-bottom. An arrow-tip is used to display a different reading direction. Role names may be displayed in square brackets beside their role. Forward and inverse readings for binaries may be shown together, separated by "/".

Construct	*Examples*	*Description/Notes*
Ternary fact type		Role names may be added in square brackets. Arrow-tips are used to reverse the default left-right or top-down reading order. Reading orders other than forward and reverse are shown using named placeholders.
Quaternary fact type		The above notes for the ternary case apply here also. Fact types of higher arity (number of roles) are also permitted.
Objectification (a.k.a. nesting)		The enrolment fact type is objectified as an entity type whose instances can play roles. In this example, the objectification type is independent, so we can know about an enrolment before the grade is obtained.
Internal uniqueness constraint (UC) on unaries		These are equivalent (by default, predicates are assumed to be populated with sets, so no whole fact may be duplicated).
Internal UC on binaries		The examples show the 4 possible patterns: 1:*n* (one-to-many); *n:*1 (many-to-one); *m:n* (many-to-many); 1:1 (one-to-one0
Internal UC on ternaries. For *n*-aries (n > 1) each UC must span at least *n*-1 roles		The first example has two, 2-role UCs: the top UC forbids ties; the other UC ensures that each team gets only place per competition (a dotted line excludes its role from the UC). The second example has a spanning UC (many-to-many-to-many).
Simple mandatory role constraint		The example constraint means that each person was born in some country. The mandatory role dot may be placed at either end of the role connector.
Inclusive-or constraint (disjunctive mandatory role)		The constraint is displayed as a circled dot connected to the constrained roles. The example constraint means that each visitor referenced in the model must have a passport or a driver licence (or both).
Preferred internal UC		A double bar on a UC indicates it underlies the preferred reference scheme.

Construct	*Examples*	*Description/Notes*
External UC (double-bar indicates preferred identifier)		Here, each state is primarily identified by combining its country and state code. Each combination of country and state name also applies to only one state.
Object Type Value Constraint		*Enumerations*
		*Ranges* are inclusive of end values by default. Round brackets are used to exclude an end value. Square brackets may be added to explicitly declare inclusion, e.g. the constraint on PositiveScore may also be specified as {(0..100]}.
		Multiple combinations are allowed.
Role value constraint		As for object type value constraints, but connected to the constrained role. Here, an age of a person must be at most 140 years.
Subset constraint		The arrow points from the subset end to the superset end (e.g. if a person smokes then that person is cancer prone). The role sequences at both ends must be compatible. A connection to the junction of 2 roles constrains that role pair.
Join subset constraint		The constrained role pair at the superset end is projected from a role path that involves a conceptual join on Language. The constraint declares that if an advisor serves in a country then that advisor must speak a language that is often used in that country.
Exclusion constraint		These constraints mean that no person is both married and widowed, and no person reviewed and authored the same book. Exclusion may apply between 2 or more compatible role sequences, possibly involving joins.
Exclusive-or constraint		An exclusive-or constraint is simply the conjunction of an inclusive-or constraint and an exclusion constraint. Also known as an xor constraint.

Construct	*Examples*	*Description/Notes*
Equality constraint		This constraint means that a patient's systolic BP is recorded if and only if his/her diastolic BP is recorded. An equality constraint may apply between 2 or more compatible role sequences, possibly involving joins.
Derived fact type, and derivation rule		A fact type is either asserted, derived, or semiderived. A derived fact type is marked with an asterisk "*". A derivation rule is supplied. A double asterisk "**" indicates derived and stored (eager evaluation).
Semiderived fact type, and derivation rule		A fact type is semiderived if some of its instances may be derived, and some of its instances may be simply asserted. It is marked by "+" (half an asterisk). "++"indicates semiderived and stored (eager evaluation for derived instances).
Subtyping		All subtypes are proper subtypes. An arrow runs from subtype to supertype. A solid arrow indicates a path to the subtype's preferred identifier (e.g. here, student employees are primarily identified by their employee number). A dashed arrow indicates the supertype has a different preferred identifier.
Subtyping constraints		A circled "X" indicates the subtypes are mutually exclusive. A circled dot indicates the supertype equals the union of the subtypes. The combination (xor constraint) indicates the subtypes partition the supertype (exclusive and exhaustive).
Subtype derivation status		A subtype may be • asserted, • derived (denoted by "*"), • or semiderived (denoted by "+"). If the subtype is asserted, it has no mark appended and has no derivation rule. If the subtype derived or semiderived, a derivation rule is supplied.

Construct	*Examples*	*Description/Notes*
Internal frequency constraint		This constrains the number of times an occurring instance of a role or role sequence may appear in each population. Here: each jury has exactly 12 members; each panel that includes an expert includes at least 4 and at most 7 experts; each expert reviews at most 5 papers; each paper that is reviewed is reviewed by at least 2 experts; and each department and year that has staff numbers recorded in the quaternary appears there twice (once for each gender).
External frequency constraint		The example constraint has the following meaning. In this context, each combination of student and course relates to at most two enrolments (i.e. a student may enroll at most twice in the same course)
Ring constraints	 E.g. 	A ring predicate *R* is locally reflexive if and only if, for all *x* and *y*, *xRy* implies *xRx*. E.g. "knows" is locally but not globally reflexive. Reflexive, symmetric and transitive properties may also be enforced using semiderivation rather than by constraining asserted fact types. The example constrains the subtyping relationship in ORM to be both acyclic (no cycles can be formed by a chain of subtyping connections) and strongly intransitive (no object type *A* can be both a direct subtype of another type *B* and an indirect subtype of *B,* where indirect subtyping means there is a chain of two or more subtyping relationships that lead from *A* to *B*). Ring constraints may be combined only if they are compatible, and one is not implied by the other. ORM tools ensure that only legal combinations are allowed.
Value-comparison constraints		The example constraint verbalizes as: **For each** Project, **existing** enddate >= startdate.

Construct	*Examples*	*Description/Notes*
Object cardinality constraint		The example constraints ensure there is exactly one president and at most 100 senators (at any given time),
Role cardinality constraint		The example constraint ensures that at most one politician plays the role of president (at any given time).
Deontic constraints	*Uniqueness* *Mandatory* *Subset, Equality, Exclusion* *Frequency* ≤2 *Irreflexive* *Acyclic* *Asymmetric* *Asym-Intrans* *Intransitive* *Acyclic-Intrans* *Antisymmetric* *Symmetric* *Strongly Intransitive* *etc.* e.g. 	Unlike alethic constraints, deontic constraint shapes are colored blue rather than violet. Most include “o” for “obligatory”. Deontic ring constraints instead use dashed lines. In the parenthood example, the alethic frequency constraint ensures that each person has at most two parents, the alethic ring constraint ensures that parenthood is acyclic, and the deontic ring constraint makes it obligatory for parenthood to be strongly intransitive.
Textual constraints	 [1] **Each** Employee **who** has Rank 'NonExec' uses **at most one** CompanyCar. [2] **Each** Employee **who** has Rank 'Exec' uses **some** CompanyCar.	First-order constraints with no graphic notation may be expressed textually in the FORML 2 language. These examples use footnoting to capture a restricted uniqueness constraint and a restricted mandatory role constraint.
Objectification display options: link fact types, and compact display.	 	Internally, link fact types connect objectified associations to their component object types. By default, display of link fact types is suppressed. If displayed, link predicate shapes use dashed lines instead of solid lines. Objectification object types may also be displayed without their defining components, using an object type shape containing a small predicate shape, as shown in this Enrolment example.