

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Z.151

(10/2012)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE
ASPECTS FOR TELECOMMUNICATION SYSTEMS

Formal description techniques (FDT) – User Requirements
Notation (URN)

User Requirements Notation (URN) – Language definition

Recommendation ITU-T Z.151



International
Telecommunication
Union

Documents
Document Sharing Hub

ITU-T Z-SERIES RECOMMENDATIONS

LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

FORMAL DESCRIPTION TECHNIQUES (FDT)	
Specification and Description Language (SDL)	Z.100–Z.109
Application of formal description techniques	Z.110–Z.119
Message Sequence Chart (MSC)	Z.120–Z.129
User Requirements Notation (URN)	Z.150–Z.159
Testing and Test Control Notation (TTCN)	Z.160–Z.179
PROGRAMMING LANGUAGES	
CHILL: The ITU-T high level language	Z.200–Z.209
MAN-MACHINE LANGUAGE	
General principles	Z.300–Z.309
Basic syntax and dialogue procedures	Z.310–Z.319
Extended MML for visual display terminals	Z.320–Z.329
Specification of the man-machine interface	Z.330–Z.349
Data-oriented human-machine interfaces	Z.350–Z.359
Human-machine interfaces for the management of telecommunications networks	Z.360–Z.379
QUALITY	
Quality of telecommunication software	Z.400–Z.409
Quality aspects of protocol-related Recommendations	Z.450–Z.459
METHODS	
Methods for validation and testing	Z.500–Z.519
MIDDLEWARE	
Processing environment architectures	Z.600–Z.609

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T Z.151

User Requirements Notation (URN) – Language definition

Summary

Recommendation ITU-T Z.151 defines the User Requirements Notation (URN) intended for the elicitation, analysis, specification and validation of requirements. URN combines modelling concepts and notations for goals (mainly for non-functional requirements and quality attributes) and scenarios (mainly for operational requirements, functional requirements and performance and architectural reasoning). The goal sub-notation is called goal-oriented requirements language (GRL) and the scenario sub-notation is called use case map (UCM).

History

Edition	Recommendation	Approval	Study Group
1.0	ITU-T Z.151	2008-11-13	17
1.1	ITU-T Z.151 (2008) Cor. 1	2012-04-29	17
2.0	ITU-T Z.151	2012-10-14	17

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at <http://www.itu.int/ITU-T/ipr/>.

© ITU 2013

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

Table of Contents

		Page
1	Scope	1
1.1	Goal modelling with URN.....	1
1.2	Scenario modelling with URN	2
1.3	Documentation structure	3
2	References.....	3
3	Definitions	4
3.1	Terms defined elsewhere	4
3.2	Terms defined in this Recommendation.....	4
4	Abbreviations and acronyms	5
5	Conventions	6
5.1	Grammars	6
5.2	Basic definitions	6
5.3	Presentation style.....	6
6	URN basic structural features.....	7
6.1	URN abstract grammar metaclasses.....	8
6.2	URN concrete grammar metaclasses.....	12
7	GRL features.....	14
7.1	GRL basic structural features	14
7.2	GRL actors.....	19
7.3	GRL intentional elements.....	21
7.4	GRL links	24
7.5	GRL strategies	32
7.6	GRL indicators	37
7.7	GRL contribution contexts	43
7.8	GRL concrete grammar metaclasses	46
8	UCM features	62
8.1	UCM basic structural features	63
8.2	UCM maps and path nodes.....	65
8.3	UCM stubs and plug-ins.....	92
8.4	UCM components.....	102
8.5	UCM scenario definitions.....	110
8.6	UCM performance annotations	120
8.7	UCM concrete grammar metaclasses	130
9	Data language	131
9.1	URN data model	131
9.2	URN data types.....	132
9.3	Grammar for expressions	134

	Page
9.4 Grammar for actions.....	136
9.5 Grammar for failures	137
10 URN interchange format	137
11 URN analysis.....	138
11.1 GRL model evaluation	138
11.2 UCM scenario path traversal	140
12 Compliance statement.....	145
13 Tool compliance	150
13.1 Definitions of valid tools	150
13.2 Conformance	150
Annex A – URN interchange format: XML schema	151
Appendix I – Summary of the URN	170
I.1 Summary of abstract metamodel	170
I.2 Summary of concrete metamodel	175
I.3 Summary of URN symbols	178
Appendix II – Examples of GRL model evaluation algorithms	179
II.1 Introduction	179
II.2 Example of quantitative evaluation algorithm	181
II.3 Example of qualitative evaluation algorithm	185
II.4 Example of hybrid evaluation algorithm	192
II.5 Calculating with exceeding expectations	193
Appendix III – Examples of UCM path traversal mechanisms	194
III.1 Introduction	194
III.2 Example of depth-first UCM path traversal mechanism	194
III.3 Example of breadth-first UCM path traversal mechanism.....	199
Bibliography.....	205

Introduction

Coverage

User Requirement Notation (URN) has concepts for the specification of goals, non-functional requirements, rationales, indicators, behaviour, scenarios and structuring. This Recommendation focuses on the definition of an abstract syntax, a concrete graphical syntax and an interchange format for URN. An assessment of conformity of the current URN representation to the language requirements for URN (Recommendation ITU-T Z.150) is also included.

Application

URN is applicable within standards bodies and industry. URN helps to describe and communicate requirements, and to develop reasoning about them. The main application areas include telecommunication systems, services and business processes, but URN is generally suitable for describing most types of reactive systems and information systems. The range of applications is from descriptions of business goals and requirements to high-level system design and architecture.

Status/Stability

This Recommendation contains the stable definition of URN. URN components for goal modelling and scenario modelling have been used for more than a decade. The main text is accompanied by the following:

- Annex A: URN Interchange Format: XML Schema
- Appendix I: Summary of the URN
- Appendix II: Examples of GRL Model Evaluation Algorithms
- Appendix III: Examples of UCM Path Traversal Mechanisms
- URN Change Request Form

Recommendation ITU-T Z.151

User Requirements Notation (URN) – Language definition

1 Scope

This Recommendation defines the User Requirements Notation (URN) intended for the elicitation, analysis, specification and validation of requirements. URN allows software and requirements engineers to discover and specify requirements for a proposed system or an evolving system, and analyse such requirements for correctness and completeness.

URN combines modelling concepts and notations for goals and intentions (mainly for non-functional requirements and quality attributes) and scenarios (mainly for operational requirements, functional requirements and performance and architectural reasoning). In particular, URN has concepts for the specification of goals, non-functional requirements, rationales, indicators, behaviour, scenarios and structuring.

This Recommendation focuses on the definition of an abstract syntax, a concrete graphical syntax, and an interchange format for URN. An assessment of conformity of the current URN representation to the language requirements for URN [ITU-T Z.150] is also included.

URN is applicable within standards bodies and industry. URN helps to describe and communicate requirements, and to develop reasoning about them. The main application areas include telecommunications systems, services and business processes, but URN is generally suitable for describing most types of reactive systems and information systems. The range of applications is from business descriptions of goals and requirements to high-level design.

URN is a notation that complies with [ITU-T Z.150]. It includes concepts and notations satisfying the language requirements of Z.150's URN-NFR (for non-functional requirements) and URN-FR (for functional requirements). URN integrates these concepts and notation into a single language.

1.1 Goal modelling with URN

The subset of the URN language that addresses ITU-T Z.150 URN-NFR language requirements is named goal-oriented requirement language (GRL), which is a language for supporting goal-oriented modelling and reasoning about requirements, especially non-functional requirements and quality attributes. It provides constructs for expressing various types of concepts that appear during the requirement process. GRL has its roots in two widespread goal-oriented modelling languages: *i** and the NFR framework. Major benefits of GRL over other popular notations include the integration of GRL with a scenario notation and a clear separation of GRL model elements from their graphical representation, enabling a scalable and consistent representation of multiple views/diagrams of the same goal model.

There are four main categories of concepts in GRL: actors, intentional elements, indicators and links. The intentional elements in GRL are goals, softgoals, tasks, resources and beliefs. They are intentional because they are used for models that allow answering questions such as why particular behaviours, and informational and structural aspects were chosen to be included in the system requirements, what alternatives were considered, what criteria were used to deliberate among alternative options, and what the reasons were for choosing one alternative over the other. Actors are holders of intentions; they are the active entities in the system or its environment (e.g., stakeholders or other systems) who want goals to be achieved, tasks to be performed, resources to be available and softgoals to be satisfied. Indicators make real-world measurements available for reasoning in the goal model, allowing for a more accurate assessment of the satisfaction of actors. Links are used to connect isolated elements in the requirement model. Different types of links

depict different structural and intentional relationships (including decompositions, contributions and dependencies).

This kind of modelling is different from the detailed specification of "what" is to be done. Here the modeller is primarily concerned with exposing "why" certain choices for behaviour and/or structure were made or constraints introduced. The modeller is not yet interested in the operational details of processes or system requirements, or component interactions. Omitting these kinds of details during early development and standardization phases allows taking a higher level (sometimes called a strategic stance) towards modelling the current or the future standard or software system and its embedding environment. Modelling and answering "why" questions leads one to consider the opportunities stakeholders seek out and/or vulnerabilities they try to avoid within their environment by utilizing capabilities of the software system and/or other stakeholders, by trying to rely upon and/or assign capabilities and by introducing constraints on how those capabilities ought to be performed.

GRL supports the analysis of strategies, which help reach the most appropriate trade-offs among (often conflicting) goals of stakeholders. A strategy consists of a set of intentional elements and indicators that are given initial satisfaction values. These satisfaction values capture contextual or future situations as well as choices among alternative means of reaching various goals. For indicators, these satisfaction values are based on real-world measurements. These satisfaction values are then propagated to the other intentional elements through their links, enabling a global assessment of the strategy being studied as well as the global satisfaction of the actors involved. A good strategy provides rationale and documentation for decisions leading to requirements, providing better context for standards/system developers and implementers while avoiding unnecessary re-evaluations of worse alternative strategies.

GRL also provides support for reasoning about scenarios by establishing correspondences between intentional GRL elements and non-intentional elements referring to scenario models of URN-FR. Modelling both goals and scenarios is complementary and aids the identification of further goals and additional scenarios (and scenario steps) important to stakeholders, thus contributing to the completeness and accuracy of requirements.

1.2 Scenario modelling with URN

The subset of the URN language that addresses Z.150 URN-FR language requirements is named use case map (UCM). UCM specifications employ scenario paths to illustrate causal relationships among responsibilities. Furthermore, UCMs provide an integrated view of behaviour and structure by allowing the superimposition of scenario paths on a structure of abstract components. The combination of behaviour and structure enables architectural reasoning after which it is possible to refine UCM specifications into more detailed scenario models such as MSCs or UML sequence diagrams, or into state machines in SDL-2010 or UML statechart diagrams and finally into concrete implementations. Validation, verification, performance analysis, interaction detection and test generation can be performed at all stages. Thus, the UCM notation enables a seamless transition from the informal to the formal by bridging the modelling gap between goal models and natural language requirements (e.g., use cases) and design in an explicit and visual way. The UCM notation allows the modeller to delay the specification of component states and messages and even, if desired, of concrete components to later, more appropriate, stages of the development process. The goal of the UCM notation is to provide the right degree of formality at the right time in the development process.

UCM specifications identify input sources and output sinks, as well as describe the required inputs and outputs of a scenario. UCM specifications also integrate many scenarios or related use cases in a map-like diagram. Scenarios can be structured and integrated incrementally. This enables reasoning about and detection of potential undesirable interactions of scenarios and components. Furthermore, the dynamic (runtime) refinement capabilities of the UCM notation allow for the

specification of (runtime) policies and for the specification of loosely coupled systems where functionality is decided at runtime through negotiation between components or compliance to high-level goals. UCM scenarios can be integrated together, yet individual scenarios are tractable through scenario definitions based on a simple data model. UCMs treat scenario paths as first class model entities and therefore build the foundation to more formally facilitate reusability of scenarios and behavioural patterns across a wide range of architectures.

The UCM notation is a specification language intended for modellers as well as non-specialists because of its visual, simple and intuitive nature but at the same time it aims to provide sufficient rigorousness for developers or tools and contracts.

Most of the characteristics of excellent requirements such as verifiable, complete, consistent, unambiguous, understandable, modifiable and traceable can be supported by UCMs. Others such as prioritized and annotated are easily incorporated.

1.3 Documentation structure

This Recommendation defines the User Requirements Notation in the following way:

- Clauses 2, 3 and 4 describe, respectively, references to related ITU-T Recommendations and other standards, definitions and acronyms used in this Recommendation.
- Clause 5 describes conventions used in this Recommendation, with a particular emphasis on metamodeling.
- Clause 6 specifies the abstract syntax of basic structural features of the URN language.
- Clause 7 specifies the abstract syntax, concrete syntax and semantics of GRL features.
- Clause 8 specifies the abstract syntax, concrete syntax and semantics of UCM features.
- Clause 9 specifies the data language used to formalize conditions and expressions.
- Clause 10 specifies an XML-based interchange format for URN models based on the concrete syntax metamodel. The XML schema definition is provided in Annex A.
- Clause 11 describes basic URN analysis techniques, namely GRL model evaluation and UCM scenario path traversal.
- Clause 12 presents how this Recommendation complies with [ITU-T Z.150].
- Clause 13 defines levels of compliances for tools.
- Annex A presents the XML schema definition of the URN interchange format.
- Appendix I provides a summary of the URN metamodel and graphical notation.
- Appendix II gives three examples of GRL model evaluation algorithms.
- Appendix III gives two examples of UCM path traversal mechanisms.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T T.55] Recommendation ITU-T T.55 (2008), *Use of the universal multiple-octet coded character set (UCS)*.

- [ITU-T Z.104] Recommendation ITU-T Z.104 (2011), *Specification and Description Language – Data and action language in SDL-2010*.
- [ITU-T Z.111] Recommendation ITU-T Z.111 (2008), *Notations and guidelines for the definition of ITU-T languages*.
- [ITU-T Z.150] Recommendation ITU-T Z.150 (2011), *User Requirements Notation (URN) – Language requirements and framework*.
- [W3C XSD1] W3C (2004), *XML Schema Part 1: Structures Second Edition*.
<<http://www.w3.org/TR/xmlschema-1>>
- [W3C XSD2] W3C (2004), *XML Schema Part 2: Datatypes Second Edition*.
<<http://www.w3.org/TR/xmlschema-2>>

3 Definitions

3.1 Terms defined elsewhere

The definitions given in [ITU-T Z.150] apply.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.2.1 abort scope: (UCM) the set of maps comprising the map where the abort start point is defined as well as all lower level maps as established by the hierarchy of stubs and plug-in maps.

3.2.2 actor: (GRL) element that represents an active entity (stakeholder or other) that has intentions and carries out actions to achieve its goals by exercising its know-how.

3.2.3 goal-oriented requirement language (GRL): The subset of the User Requirements Notation used to model and analyse non-functional requirements and quality attributes with goal graphs.

3.2.4 GRL link: (GRL) intentional relationship existing between intentional elements or actors.

3.2.5 indicator: (GRL) relates a measured, real-world, quantitative or qualitative value with a quantitative or qualitative satisfaction value and is used to measure the performance, any other relevant real-world property or satisfaction of an intentional element, actor or system.

3.2.6 in-path: (UCM) incoming path of a stub, in particular the last node connection before reaching the stub.

3.2.7 intentional element: (GRL) element that describes an intention. Used for models that allow answering questions such as why particular behaviours, informational and structural aspects were chosen to be included in the system requirement, what alternatives were considered, what criteria were used to deliberate among alternative options and what the reasons were for choosing one alternative over the other.

3.2.8 out-path: (UCM) outgoing path of a stub, in particular the first node connection after leaving the stub.

3.2.9 release path: (UCM) path with an end point or empty point that is connected to a timer.

3.2.10 scenario definition: (UCM) collection of initial values, initial conditions and desired conditions used to identify and test individual scenarios during the traversal of a UCM model.

3.2.11 strategy: (GRL) collection of satisfaction values associated with intentional elements used to provide an initial context for GRL model analysis.

3.2.12 traversal root map: (UCM) map that is at the highest level in the map hierarchy established by the traversal mechanism for a scenario definition.

3.2.13 trigger path: (UCM) path with an end point or empty point that is connected to a waiting place.

3.2.14 unconnected start point: (UCM) start point that is not a failure start point or abort start point and that is not directly connected to another end point or to another path.

3.2.15 use case map notation: The subset of the User Requirements Notation used to model and analyse operational requirements and functional requirements with use cases and scenarios.

3.2.16 visit: (UCM) visit of a synchronizing stub is characterized by how often an in-path of the stub has been traversed. If an in-path is traversed the first time, then it is the first visit of the stub. If the same in-path is traversed the n^{th} time, then it is the n^{th} visit of the stub. If another in-path of the stub is traversed for the first time, then it is the first visit of the stub. Plug-in maps that have been instantiated because of a visit are said to belong to the visit.

3.2.17 waiting path: (UCM) incoming path of a waiting place or timer that is not a release path or trigger path.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations:

COTS	Commercial-Off-The-Shelf
DSP	Digital Signal Processor
FAS	Failure start point or Abort Start point
FP	Failure Point
FR	Functional Requirement
GRL	Goal-oriented Requirement Language
MSC	Message Sequence Chart
NFR	Non-Functional Requirement
NTRP	Number of arrivals along the Trigger/Release Path
NWP	Number of arrivals along the Waiting Path
OCS	Originating Call Screening
PCC	Path Continuation Criteria
PT	Path Traversal
RP	Regular Path
SDL-2010	Specification and Description Language 2010
TRP	Trigger/Release Path
TTCN-3	Testing and Test Control Notation (TTCN-3)
UCM	Use Case Map
UCS	Universal Multiple-Octet Coded Character Set
UML	Unified Modelling Language
URN	User Requirements Notation
URN-FR	User Requirements Notation – Functional Requirements
URN-NFR	User Requirements Notation – Non-Functional Requirements

UTF-8	8-bit UCS/Unicode Transformation Format
WP	Waiting Path
XML	eXtensible Markup Language
XSD	XML Schema Definition

5 Conventions

The conventions of [ITU-T Z.111] apply to this Recommendation.

This Recommendation uses the universal multiple-octet coded character set (UCS) encoding of characters recommended in [ITU-T T.55].

5.1 Grammars

The conventions of [ITU-T Z.111] apply to this Recommendation.

5.2 Basic definitions

5.2.1 Validity

A specification is a valid User Requirements Notation specification only if it satisfies the syntactic rules and the static conditions defined in this Recommendation.

5.3 Presentation style

The conventions of [ITU-T Z.111] apply to this Recommendation.

5.3.1 Division of text

The conventions of [ITU-T Z.111] apply to this Recommendation.

5.3.2 Titled enumeration items

a) *Abstract grammar*

The abstract grammar is specified in the form defined in [ITU-T Z.111]. The metamodel presentation of abstract syntax in [ITU-T Z.111], clause 5.4.1.2 is used. Each metaclass in the metamodel, attributes, relationships to other metaclasses and constraints (static conditions expressed in natural language) are specified. As an extension to [ITU-T Z.111] conventions there are additional subheadings of the form i) **Attributes**, ii) **Relationships** and iii) **Constraints**.

b) *Concrete grammar*

The URN concrete grammar is presented as an extension to the abstract grammar metamodel combined with a description of the graphical symbols used. The concrete grammar includes all the metaclasses (with attributes, relationships and constraints) of the abstract grammar. The additional concepts (shown as grey metaclasses) that extend the abstract grammar metamodel are useful to support a graphical language but they have no semantic implication. Common additional concepts include layout information, line styles and informal descriptions. For example, in Figure 1, a colour attribute is added to an element of the abstract grammar. Composition with multiplicity 0..1 is used here to ensure that specifications without this layout information are still valid and that the additional concept will not interfere during analysis.

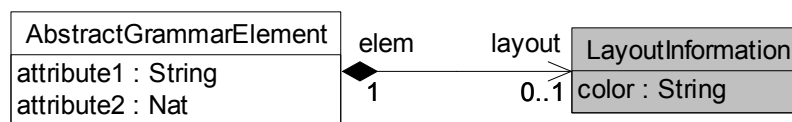


Figure 1 – Example of metaclasses from an abstract grammar (white) and a concrete grammar (grey)

As a convention and in order to simplify diagrams, metaclasses that are defined in a different section but that are referenced in a diagram will have their attributes hidden. For example, Figure 2 makes reference to an abstract grammar element metaclass defined elsewhere (e.g., in Figure 1), but its attributes are hidden. Note that this is different from a metaclass *without* attributes, in which case the attribute compartment at the bottom of the metaclass (bounded by an horizontal line in the middle of the rectangle) is empty. Similarly, only relevant associations are presented in diagrams. Appendix I provides a global overview of all metaclasses, their associations and their attributes.

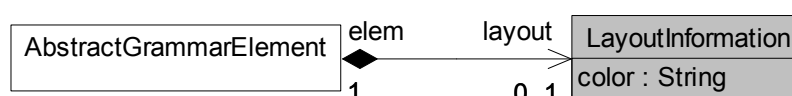


Figure 2 – Example of metaclasses where the left one (with attributes hidden) is detailed elsewhere

Not all URN metaclasses, attributes or relationships have a concrete graphical notation. It is then up to tools to provide ways of creating, accessing and modifying instances of these metaclasses (for instance, through a "property" window).

Many elements with a graphical representation also have model-specific coordinates and sizes. The following convention is used for layout coordinates information.

- Horizontal coordinate (x-axis): An integer value representing the number of point units from the origin (0). Positive values are at the right of the origin and negative values at the left of the origin.
- Vertical coordinate (y-axis): An integer value representing the number of point units from the origin (0). Positive values are below the origin and negative values above the origin.

c) *Semantics*

The semantics of the abstract grammar metaclass is expressed in natural language. The semantics of a concrete grammar metaclass is that of its abstract grammar metamodel elements (the additional grey metaclasses have no semantics).

d) *Model*

This clause, when present, describes shorthand or alternative concrete syntaxes.

e) *Example*

Where necessary, examples of use are included. These examples are only informative, not normative.

6 URN basic structural features

The URN basic structural features describe containers for URN, GRL and UCM specifications, as well as definitions of URN model elements, their links and metadata, concerns and conditions. The abstract syntax metaclasses are first presented in clause 6.1. Their concrete grammar references concrete syntax metaclasses regrouped in clause 6.2.

- Clause 6.1: URN abstract grammar metaclasses
- Clause 6.2: URN concrete grammar metaclasses

NOTE – Clause 9 defines the data model and data language for URN. In particular, it defines the *Integer* type used in some of the attributes of the abstract and concrete syntax metaclasses for URN. The attribute types Boolean, Nat and String used in these metaclasses are those defined in [ITU-T Z.111].

6.1 URN abstract grammar metaclasses

The topmost metaclass, *URNspec* (see Figure 3), contains directly or indirectly all the other elements of a URN model, including concerns (see Figure 4). In this Recommendation, the terms "URN model" and "URN specification" are used interchangeably.

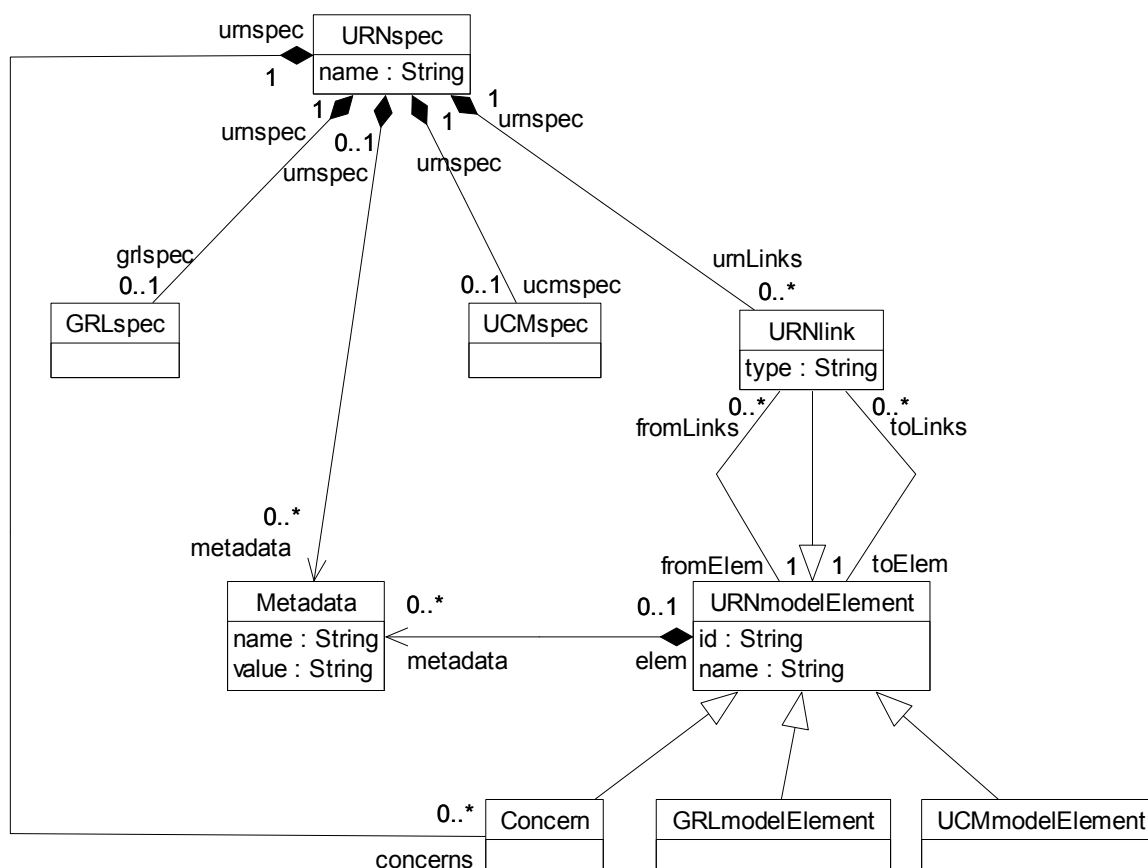


Figure 3 – Abstract grammar: URN specification, links, metadata and model elements

6.1.1 URNspec

URNspec is the root element of a URN model/specification. It names the specification and serves as a container for all the other specification elements (see Figure 3).

a) Abstract grammar

i) Attributes

- name (String): The name of the URN specification.

ii) Relationships

- Composition of *GRLspec* (0..1): The *URNspec* may contain one GRL specification (see clause 7.1.1).

- Composition of *UCMSpec* (0..1): The *URNspec* may contain one UCM specification (see clause 8.1.1).
- Composition of *URNlink* (0..*): The *URNspec* may contain URN links.
- Composition of *Metadata* (0..*): The *URNspec* may contain metadata information.
- Composition of *Concern* (0..*): The *URNspec* may contain concerns.

iii) Constraints

- There exists exactly one instance of *URNspec* in a URN specification.

b) Concrete grammar

URNspec has no concrete syntax. However, it may contain additional information in an instance of *ConcreteURNspec*.

i) Relationships

- Composition of *ConcreteURNspec* (0..1): The *URNspec* may contain one concrete URN specification (see Figure 5).

c) Semantics

None (*URNspec* is a structural concept only).

6.1.2 URNmodelElement

URN model elements have names and unique identifiers. They can also be linked to each other (see Figure 3).

a) Abstract grammar

i) Attributes

- *id* (String): The identifier of the URN model element.
- *name* (String): The name of the URN model element.

ii) Relationships

- Composition of *Metadata* (0..*): A *URNmodelElement* may contain metadata information.
- Association with *URNlink* (*fromLinks*, 0..*): A *URNmodelElement* may be the source of URN links.
- Association with *URNlink* (*toLinks*, 0..*): A *URNmodelElement* may be the target of URN links.
- Association with *Concern* (0..1): A *URNmodelElement* may belong to one concern.
- *URNmodelElement* is a superclass of *URNlink*, *Concern*, *GRLmodelElement* (see clause 7.1.2) and *UCMmodelElement* (see clause 8.1.2).

iii) Constraints

- *id* shall be unique within the URN specification.
- All instances of *URNmodelElement* shall appear in one of its subclasses (that is, metaclass *URNmodelElement* is abstract).

b) Concrete grammar

The concrete syntax for *URNmodelElement* is further defined in its subclasses. In addition, a *URNmodelElement* may contain an informal description in a *Description*.

i) Relationships

- Composition of *Description* (0..1): A *URNmodelElement* may contain one description (see Figure 6).

c) *Semantics*

A *URNmodelElement* is an uniquely identifiable model element that can contain metadata and be linked to other model elements. Its subclasses possibly have additional attributes and relationships.

6.1.3 URNlink

A URN link is a URN model element that connects a source URN model element to a target URN model element. URN links have a user-defined type (see Figure 3).

a) *Abstract grammar*

i) **Attributes**

- Inherits attributes from *URNmodelElement*.
- type (String): The user-defined type of the URN link.

ii) **Relationships**

- Inherits relationships from *URNmodelElement*.
- Contained by *URNspec* (1): A *URNlink* is contained in the URN specification.
- Association with *URNmodelElement* (fromElem, 1): A *URNlink* has one source URN model element.
- Association with *URNmodelElement* (toElem, 1): A *URNlink* has one target URN model element.

iii) **Constraints**

- Inherits constraints from *URNmodelElement*.

b) *Concrete grammar*

The presence of a link on a source or target model element is indicated with a triangle symbol (►) next to the name of the element, if that element's name is displayed in the concrete syntax.

i) **Relationships**

- Inherits relationships from *URNmodelElement*.

c) *Semantics*

URNlinks provide modellers with a way to create new relationships of various types between any pair of model elements in a URN specification. These links can be used for traceability, refinement, composition and other purposes, hence providing an extensible semantics to URN.

6.1.4 Metadata

Metadata is a name-value pair that can be used to attach information to a URN specification or its model elements. *Metadata* is contained by the *URNspec* or a *URNmodelElement* (see Figure 3).

a) *Abstract grammar*

i) **Attributes**

- name (String): The name of the URN metadata information instance.
- value (String): The value of the URN metadata information instance.

ii) **Relationships**

- None.

iii) Constraints

- Each *Metadata* instance is contained in exactly one instance of type *URNspec* or *URNmodelElement*.

b) Concrete grammar

None.

c) Semantics

Metadata instances provide modellers with a way to attach user-defined named values to most elements found in a URN specification, hence providing an extensible semantics to URN.

6.1.5 Concern

A *Concern* is a guarded grouping of URN model elements. Concerns are typically used to group related GRL and UCM diagrams into one unit of understanding (see Figures 3 and 4).

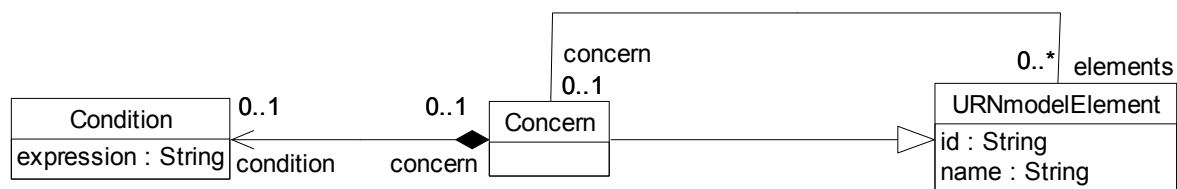


Figure 4 – Abstract grammar: URN concerns

a) Abstract grammar

i) Attributes

- Inherits attributes from *URNmodelElement*.

ii) Relationships

- Inherits relationships from *URNmodelElement*.
- Contained by *URNspec* (1): A *Concern* is contained in the URN specification.
- Composition of *Condition* (0..1): A *Concern* may contain one condition.
- Association with *URNmodelElement* (0..*): A *Concern* possibly groups URN model elements.

iii) Constraints

- Inherits constraints from *URNmodelElement*.

b) Concrete grammar

A *Concern* does not have a visual representation.

i) Relationships

- Inherits relationships from *URNmodelElement*.

c) Semantics

A *Concern* groups URN model elements together. This grouping can be guarded with a *Condition* for composition purposes.

6.1.6 Condition

Condition is a Boolean expression that serves as a guard, precondition, postcondition or failure condition. A *Condition* is contained by one of the following model elements: *Concern*, *StartPoint*, *EndPoint*, *PluginBinding*, *NodeConnection* or *ScenarioDef* (see Figures 4, 77 and 92).

a) *Abstract grammar*

i) **Attributes**

- expression (String): The Boolean expression of the condition.

ii) **Relationships**

- None.

iii) **Constraints**

- The expression of a *Condition* shall be a Boolean expression, as defined in clause 9.3.
- Each *Condition* instance is contained in exactly one instance of type *Concern*, *StartPoint*, *EndPoint*, *PluginBinding*, *NodeConnection* or *ScenarioDef*.

b) *Concrete grammar*

Condition has no concrete syntax, but the label of the *ConcreteCondition* contained by the condition is visualized for *NodeConnections*, *StartPoints* and *EndPoints* (see clauses 8.2.3, 8.2.6 and 8.2.8, respectively). Conditions for *Concerns*, *PluginBindings* or *ScenarioDefs* are not visualized.

i) **Relationships**

- Composition of *ConcreteCondition* (0..1): A *Condition* may have one concrete condition (see Figure 7).
- Composition of *Label* (0..1): A *Condition* may have one label (see Figure 52).

c) *Semantics*

The expression of a *Condition* contained by a *Concern* indicates whether the grouping of model elements identified by the concern is to be enabled in the URN specification (true) or disabled (false).

The expression of a *Condition* not contained by a *Concern* is evaluated at runtime when the model element to which the condition belongs is reached during the path traversal of the UCM specification. The evaluation results in either true or false.

The expression may make use of globally defined *Variables* and shall be well-formed according to the textual grammar detailed in clause 9.3. If the expression uses a variable name with a "_pre" suffix (e.g., in the postcondition of a scenario definition), then the initialized value of this variable, prior to traversal of the UCM specification, is used. This is mainly useful in postconditions that compare the runtime value of a variable with the initial value of that variable. For example, the expression `VariableX == VariableX_pre + 1` will be true if `VariableX` has been incremented by 1 since the beginning of the traversal.

If a *Condition* is not specified for a *Concern*, *StartPoint*, *EndPoint*, *PluginBinding*, *NodeConnection* or *ScenarioDef*, but the existence of the condition is required by, for example, model analysis, then the evaluation of the required condition is assumed to result in true.

6.2 URN concrete grammar metaclasses

The following concrete grammar metaclasses may be contained by some of the abstract grammar metaclasses. They have no semantics.

6.2.1 ConcreteURNspec

The *ConcreteURNspec* metaclass contains standard meta-information about the URN model (*URNspec*) itself (see Figure 5).

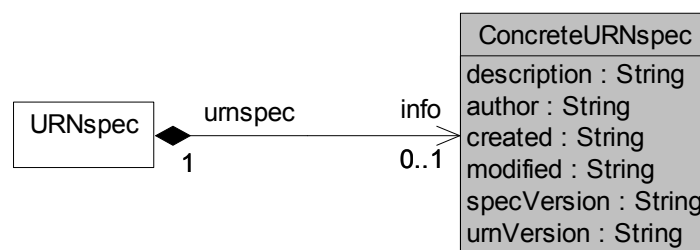


Figure 5 – Concrete grammar: ConcreteURNspec metaclass

a) *Abstract grammar*
None. This is a concrete syntax metaclass only.

b) *Concrete grammar*
There is no visual representation of this metaclass.

i) Attributes

- description (String): An informal description of the URN specification.
- author (String): The author of the URN specification.
- created (String): The date and time of creation of the URN specification. The suggested format is (in English) "Month day, year hours:minutes:seconds AmOrPm timezone". For example: "November 15, 2007 9:21:06 AM EST".
- modified (String): The date and time of the last modification to the URN specification. The suggested format is (in English) "Month day, year hours:minutes:seconds AmOrPm timezone". For example: "November 15, 2007 9:21:06 AM EST".
- specVersion (String): The version number of the URN specification. It is suggested to use an integer that starts at 1 when the specification is first created and that is incremented by one each time the specification is modified.
- urnVersion (String): The version number of the URN standard used. For example: "Z.151 (11/08)".

ii) Constraints

- The date modified is later than the date created.

6.2.2 Description

An informal *Description* can be attached to any *URNmodelElement* (see Figure 6).

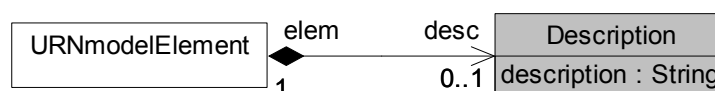


Figure 6 – Concrete grammar: Description metaclass

a) *Abstract grammar*
None. This is a concrete syntax metaclass only.

b) *Concrete grammar*
There is no visual representation of this metaclass.

i) Attributes

- description (String): An informal description of the URN model element.

6.2.3 ConcreteCondition

ConcreteCondition defines a label and a description for a *Condition* (see Figure 7).

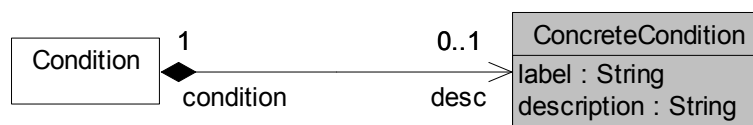


Figure 7 – Concrete grammar: ConcreteCondition metaclass

a) *Abstract grammar*

None. This is a concrete syntax metaclass only.

b) *Concrete grammar*

The label of *ConcreteCondition* is visualized for containing *Conditions* of *NodeConnections*, *StartPoints* and *EndPoints* (see clauses 8.2.3, 8.2.6 and 8.2.8, respectively).

i) **Attributes**

- label (String): The label for the condition is used for visualization purposes.
- description (String): An informal description of the condition.

c) *Semantics*

None.

7 GRL features

The goal-oriented requirement language provides a set of URN features that enable the description and analysis of goals/intentions of systems and stakeholders. The GRL features are grouped under six categories:

- GRL basic structural features: clause 7.1.
- GRL actors: clause 7.2.
- GRL intentional elements: clause 7.3.
- GRL links: clause 7.4.
- GRL strategies: clause 7.5.
- GRL indicators: clause 7.6.
- GRL contribution contexts: clause 7.7.
- GRL concrete grammar metaclasses: clause 7.8.

NOTE – Many of the concrete grammar metaclasses defined here are also used by UCM features.

7.1 GRL basic structural features

The GRL basic structural features describe containers for GRL specifications, as well as definitions of GRL model elements, including linkable elements. The abstract grammar metaclasses are presented in this clause, whereas their concrete grammar metaclasses are detailed in clause 7.8.

7.1.1 GRLspec

GRLspec serves as a container for the GRL specification elements (see Figure 8).

NOTE – The name *intElements* is used for backward compatibility with previous versions of this Recommendation. A more appropriate name would be *contElements*.

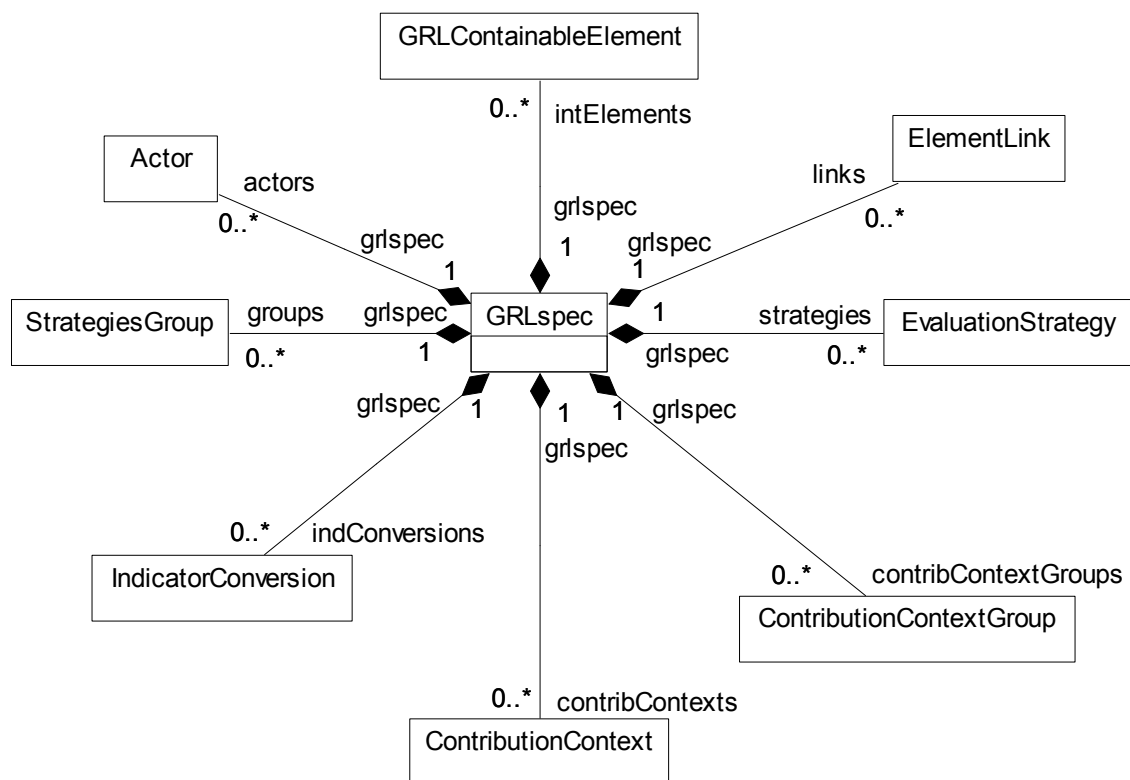


Figure 8 – Abstract grammar: GRL specification

a) *Abstract grammar*

i) Attributes

- None.

ii) Relationships

- Contained by *URNspec* (1): The *GRLspec* is contained in the URN specification (see Figure 3).
- Composition of *Actor* (0..*): The *GRLspec* may contain actor definitions.
- Composition of *GRLContainableElement* (0..*): The *GRLspec* may contain containable elements.
- Composition of *IndicatorConversion* (0..*): The *GRLspec* may contain conversions for indicators.
- Composition of *ElementLink* (0..*): The *GRLspec* may contain element links.
- Composition of *StrategiesGroup* (0..*): The *GRLspec* may contain strategy groups.
- Composition of *EvaluationStrategy* (0..*): The *GRLspec* may contain evaluation strategies.
- Composition of *ContributionContextGroup* (0..*): The *GRLspec* may contain contribution context groups.
- Composition of *ContributionContext* (0..*): The *GRLspec* may contain contribution contexts.

iii) Constraints

- None.

b) *Concrete grammar*

GRLspec has no concrete syntax. However, it may contain concrete GRL specification information and GRL graphs.

i) **Relationships**

- Composition of *ConcreteGRLspec* (0..1): The *GRLspec* may contain one concrete GRL specification (see Figure 30).
- Composition of *GRLGraph* (0..*): The *GRLspec* may contain GRL graphs (see Figure 30).

c) *Semantics*

None (*GRLspec* is a structural concept only).

7.1.2 GRLmodelElement

a) *Abstract grammar*

A *GRLmodelElement* is a URN model element specialized for GRL concepts (see Figure 9).

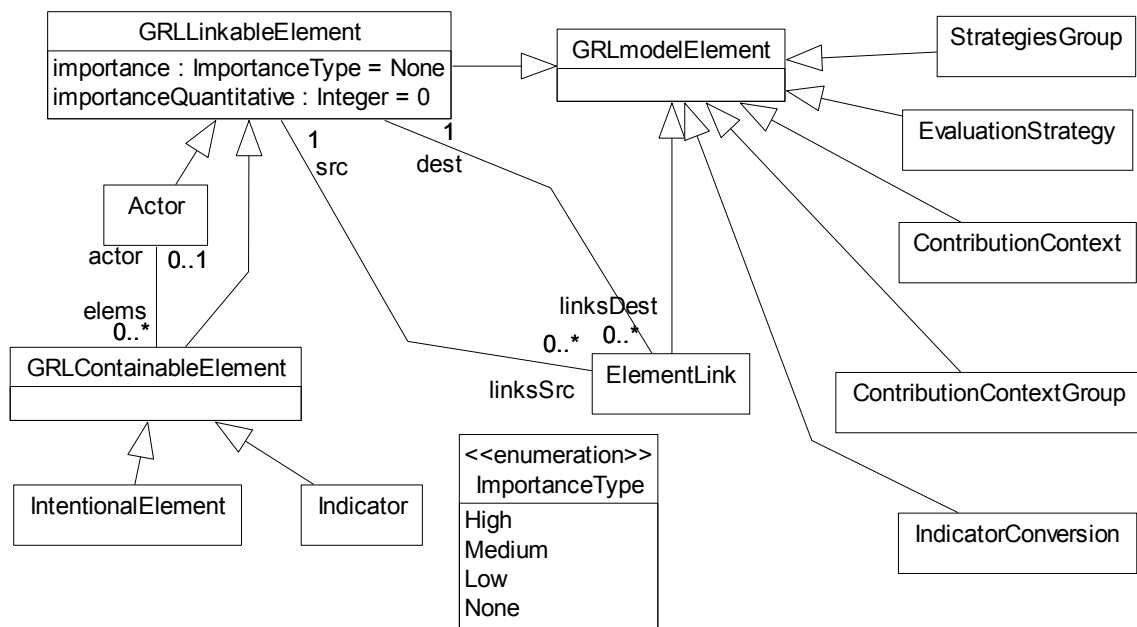


Figure 9 – Abstract grammar: GRL model, linkable and containable elements

i) **Attributes**

- Inherits attributes from *URNmodelElement* (see Figure 3).

ii) **Relationships**

- Inherits relationships from *URNmodelElement*.
- *GRLmodelElement* is a superclass of *GRLLinkableElement*, *ElementLink*, *StrategiesGroup*, *EvaluationStrategy*, *IndicatorConversion*, *ContributionContextGroup* and *ContributionContext*.

iii) **Constraints**

- Inherits constraints from *URNmodelElement*.
- All instances of *GRLmodelElement* shall appear in one of its subclasses (that is, metaclass *GRLmodelElement* is abstract).

b) *Concrete grammar*

The concrete syntax for *GRLmodelElement* is further defined in its subclasses.

i) **Relationships**

- Inherits relationships from *URNmodelElement* (see Figure 6).
- *GRLmodelElement* is a superclass of *GRLGraph*, *ActorRef*, *GRLNode* and *LinkRef* (see Figure 30).

c) *Semantics*

A *GRLmodelElement* is an uniquely identifiable GRL model element that can contain metadata and be linked to other URN model elements through *URNlinks*. Its subclasses may have additional attributes and relationships.

7.1.3 GRLLinkableElement

A *GRLLinkableElement* is a GRL model element that has an importance and can be linked to other GRL linkable elements through an *ElementLink*. *GRLLinkableElement* abstracts the commonalities of actor definitions, intentional elements and indicators (see Figure 9).

a) *Abstract grammar*

i) **Attributes**

- Inherits attributes from *GRLmodelElement*.
- importance (*ImportanceType*): Qualitative importance of the containable element to its containing actor definition in the case of intentional elements and indicators and to the overall GRL model in the case of actors, if any. Default value is None.
- importanceQuantitative (*Integer*): Quantitative importance of the containable element to its containing actor definition in the case of intentional elements and indicators and to the overall GRL model in the case of actors, if any. Default value is 0 (see clause 9.2.2).

ii) **Relationships**

- Inherits relationships from *GRLmodelElement*.
- Association with *ElementLink* (linksSrc, 0..*): A *GRLLinkableElement* may be the source of GRL element links.
- Association with *ElementLink* (linksDest, 0..*): A *GRLLinkableElement* may be the destination of GRL element links.
- *GRLLinkableElement* is a superclass of *GRLContainableElement* and *Actor*.
- Uses *ImportanceType* enumeration.

iii) **Constraints**

- Inherits constraints from *GRLmodelElement*.
- All instances of *GRLLinkableElement* shall appear in one of its subclasses (that is, metaclass *GRLLinkableElement* is abstract).
- importanceQuantitative ≥ 0 and importanceQuantitative ≤ 100 .
- The name of a *GRLLinkableElement* cannot be the empty String "".

b) *Concrete grammar*

The concrete syntax for *GRLLinkableElement* is further defined in its subclasses. The line and fill colours of a linkable element are specified in its concrete style (*ConcreteStyle*) and are hence shared by all the linkable element's references.

i) Relationships

- Inherits relationships from *GRLmodelElement*.
- Composition of *ConcreteStyle* (0..1): A *GRLLinkableElement* may contain one concrete style (see Figure 30).

c) Semantics

A *GRLLinkableElement* is a GRL model element that can be linked to other actor definitions, intentional elements and indicators. The semantics of importance and importanceQualitative are described by the subclasses of *GRLLinkableElement* (see clauses 7.1.5 and 7.2.1).

7.1.4 ImportanceType

The qualitative importance of an actor to the overall GRL model and of an intentional element or indicator to its actor definition can be High, Medium, Low or None (see Figure 9).

a) Abstract grammar

i) Attributes

- None (enumeration metaclass).

ii) Relationships

- Used by *GRLLinkableElement*.

iii) Constraints

- None.

b) Concrete grammar

None (enumeration metaclass). However, it influences the presentation of intentional elements and indicators contained by actor definitions (see clause 7.8.5) as well as the presentation of actors (see clauses 7.8.3 and 7.8.6).

c) Semantics

High is used for specifying the highest importance, Low for some non-null importance, Medium for a level in between high and low, and finally None for no importance. The satisfaction level of an intentional element or indicator with a None importance will have no impact on the qualitative evaluation of the global satisfaction of the associated actor definition. The satisfaction level of an actor with a None importance will have no impact on the qualitative evaluation of the global satisfaction of the overall GRL model.

7.1.5 GRLContainableElement

A *GRLContainableElement* is a GRL model element that can be contained in an actor. *GRLContainableElement* abstracts the commonalities of intentional elements and indicators (see Figure 9). A *GRLContainableElement* may be given *Evaluation* values (see Figure 24).

a) Abstract grammar

i) Attributes

- Inherits attributes from *GRLLinkableElement*.

ii) Relationships

- Inherits relationships from *GRLLinkableElement*.
- Association with *Actor* (0..1): A *GRLContainableElement* may be contained in one actor definition.
- *GRLContainableElement* is a superclass of *IntentionalElement* and *Indicator*.

iii) Constraints

- Inherits constraints from *GRLLinkableElement*.
- All instances of *GRLContainableElement* shall appear in one of its subclasses (that is, metaclass *GRLContainableElement* is abstract).
- If importance of a *GRLContainableElement* is greater than 0, then the *GRLContainableElement* shall be contained in an actor definition.
- If importanceQuantitative is not equal to None, then the *GRLContainableElement* shall be contained in an actor definition.

b) Concrete grammar

The concrete syntax for *GRLContainableElement* is further defined in its subclasses.

i) Relationships

- Inherits relationships from *GRLLinkableElement*.
- Association with *IntentionalElementRef* (0..*): An *GRLContainableElement* may be referenced by intentional (and indicator) element references (see Figure 30).

c) Semantics

A *GRLContainableElement* is a GRL model element that can be contained in an actor definition.

A value of 0 for importanceQuantitative means that the intentional element or indicator is not important to the actor definition, whereas a value of 100 means that it is highly important. A value of None for importance means that the intentional element or indicator is not important to the actor definition, whereas a value of High means that it is highly important. Often, only top-level intentional elements or indicators in a GRL actor will have a non-null importance factor, which summarizes the importance of decomposing or contributing elements.

The two importance attributes are only taken into consideration during actor satisfaction analysis, when the intentional element or indicator is included by an actor definition (they have no meaning otherwise). Only the relevant importance attribute is considered depending on the type of analysis (qualitative or quantitative).

It is not required for importance and importanceQuantitative to be consistent, as modellers may want to use only one type of analysis (qualitative or quantitative). However, it is recommended to keep them consistent if the modellers intend to switch between different types of analysis.

7.2 GRL actors

Figure 10 shows the metaclasses for GRL actors, intentional elements and their links. It is referenced by this clause as well as by clauses 7.3 and 7.4.

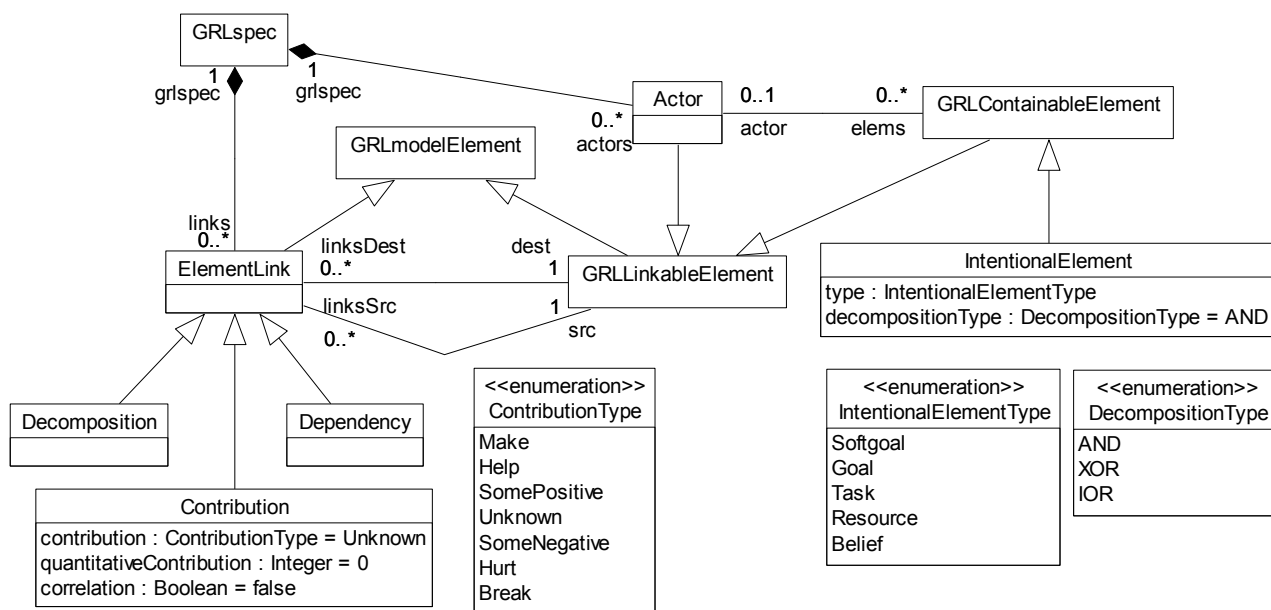


Figure 10 – Abstract grammar: GRL actors, intentional elements and links

7.2.1 Actor

An *Actor* (also referred to as actor definition) is a GRL linkable element that represents an entity that has intentions and carries out actions to achieve its goals by exercising its know-how. Actor definitions are often used to represent stakeholders as well as systems. Actor definitions may contain containable elements (see Figure 10).

One could start modelling the domain using only actors and dependencies between actors, without containable elements inside the actors. One can then add intentional elements and indicators inside the actors to specify why actors depend on each other and how dependencies between actors are fulfilled.

a) Abstract grammar

i) Attributes

- Inherits attributes from *GRLLinkableElement*.

ii) Relationships

- Inherits relationships from *GRLLinkableElement*.
- Contained by *GRLspec* (1): An *Actor* definition is contained in the GRL specification (see Figure 8).
- Association with *GRLContainableElement* (0..*): An *Actor* definition may contain intentional elements and indicators.

iii) Constraints

- Inherits constraints from *GRLLinkableElement*.
- Any two *Actor* definitions cannot share the same name inside a URN specification.

b) Concrete grammar

An actor definition does not have a visual representation, but actor references (*ActorRef*) and collapsed actor references (*CollapsedActorRef*) in GRL diagrams do have a graphical representation. The colour of an actor definition's circle line and the fill colour are defined in the concrete style (*ConcreteStyle*) (from superclass *GRLLinkableElement*) and are hence shared by all the actor's references.

i) Relationships

- Inherits relationships from *GRLLinkableElement*.
- Association with *ActorRef* (0..*): An *Actor* definition may be referenced by actor references (see Figure 30).
- Association with *CollapsedActorRef* (0..*): An *Actor* definition may be referenced by collapsed actor references (see Figure 30).

c) Semantics

An *Actor* definition is a GRL linkable element that may contain intentional elements and indicators describing its intentions, capabilities and related measures. An actor definition may also depend on another actor definition to satisfy some intentional element or indicator. How well an actor definition is satisfied depends on the satisfaction level and importance of the intentional elements and indicators it contains.

A value of 0 for *importanceQuantitative* means that the actor is not important to the overall GRL model, whereas a value of 100 means that the actor is highly important. A value of *None* for *importance* means that the actor is not important to the overall GRL model, whereas a value of *High* means that the actor is highly important.

The two importance attributes are taken into consideration during overall GRL model satisfaction analysis. Only the relevant importance attribute is considered depending on the type of analysis (qualitative or quantitative).

It is not required for *importance* and *importanceQuantitative* to be consistent, as modellers may want to use only one type of analysis (qualitative or quantitative). However, it is recommended to keep them consistent if the modellers intend to switch between different types of analysis.

d) Model

None.

e) Examples

Figure 11 is a GRL diagram that shows a "Telecom Provider" as a collapsed actor (left) and as an actor with boundary (right). The collapsed actor has an importance value of 50 whereas the actor with boundary has an importance value of 0 (hence its importance is not shown). See clauses 7.8.3 and 7.8.6 for the details of the concrete syntax.

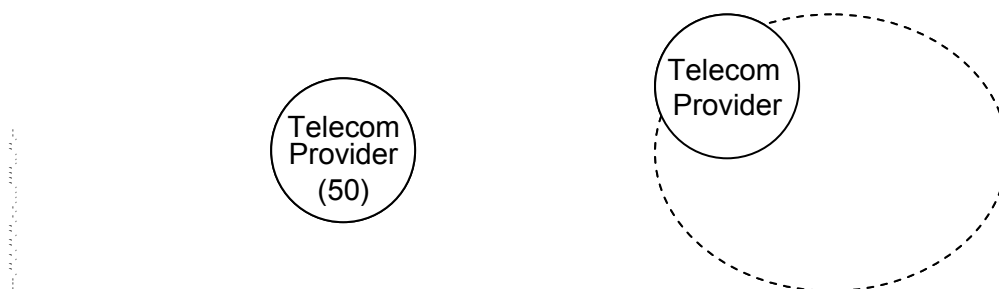


Figure 11 – Example: GRL collapsed actor (left) and actor with boundary (right)

7.3 GRL intentional elements

7.3.1 IntentionalElement

An *IntentionalElement* is a GRL containable element used for models that allow answering questions such as why particular behaviours, informational and structural aspects were chosen to be included in the system requirement, what alternatives were considered, what criteria were used to deliberate among alternative options and what the reasons were for choosing one alternative over

another. Intentional elements may be included in actor definitions and they can be linked to each other in different ways. There are different types of intentional elements specified. Intentional elements can be decomposed and they can be given a quantitative or qualitative importance level when included in an actor definition (see Figure 10).

a) *Abstract grammar*

i) **Attributes**

- Inherits attributes from *GRLContainableElement*.
- *type* (*IntentionalElementType*): The type of intentional element.
- *decompositionType* (*DecompositionType*): The type of decomposition when this intentional element is the source of decomposition link, if any. Default value is AND.

ii) **Relationships**

- Inherits relationships from *GRLContainableElement*.
- Uses *IntentionalElementType* enumeration.
- Uses *DecompositionType* enumeration.

iii) **Constraints**

- Inherits constraints from *GRLContainableElement*.
- If an *IntentionalElement* is associated with an *Actor* definition, then there is only one *IntentionalElement* with this name associated with the *Actor* definition.
- If an *IntentionalElement* is not associated with any *Actor* definition, then there is only one *IntentionalElement* with this name that is not associated with any *Actor* definition.

b) *Concrete grammar*

An intentional element does not have a visual representation, but intentional element references (*IntentionalElementRef*) in GRL diagrams do have a graphical representation.

i) **Relationships**

- Inherits relationships from *GRLContainableElement*.

c) *Semantics*

An *IntentionalElement* describes an intention or a capability. An intentional element contained in an *Actor* definition is held by this actor definition and therefore describes part of its intentions or capabilities.

d) *Model*

None.

e) *Examples*

Figure 12 is a GRL diagram that shows five intentional elements, one for each type.

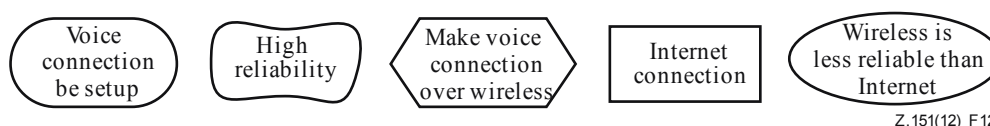


Figure 12 – Example: GRL intentional elements

- "Voice Connection Be Setup" is defined as a (hard) goal because this is something that can be achieved entirely.
- "High Reliability" is defined as a softgoal because this is something that can never be entirely achieved (but that can be sufficiently achieved).

- "Make Voice Connection Over Wireless" is defined as a task because this is a particular way of setting up a connection.
- "Internet Connection" is defined as a resource because this is a physical entity that can be available or not.
- "Wireless is less reliable than Internet" is defined as a belief because this provides a rationale for some of the design decisions.

See clauses 7.1.4, 7.3.2, and 7.8.5 for the details of the concrete syntax.

The GRL diagram in Figure 13 shows a goal and a task contained by a "Telecom Provider" actor. The goal has an importance value of 50 whereas the task has an importance value of 0 (hence its importance is not shown).

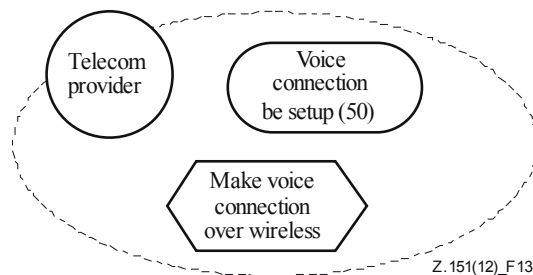


Figure 13 – Example: GRL actor that contains a goal and a task

7.3.2 IntentionalElementType

An intentional element can be a Goal, Softgoal, Task, Resource or Belief (see Figure 10).

a) Abstract grammar

i) Attributes

- None (enumeration metaclass).

ii) Relationships

- Used by *IntentionalElement*.

iii) Constraints

- None.

b) Concrete grammar

The symbols in Figure 14 are used to denote the various types of GRL intentional elements. See usage in *IntentionalElementRef*, clause 7.8.5.

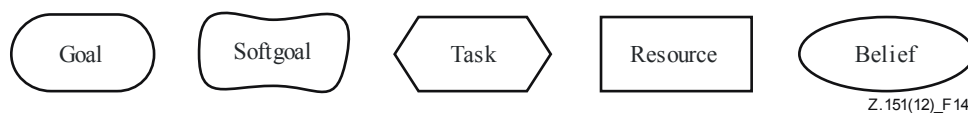


Figure 14 – Symbol: GRL intentional element types

c) Semantics

- A (hard) Goal is a condition or state of affairs in the world that the stakeholders would like to achieve. How the goal is to be achieved is not specified, allowing alternatives to be considered. A goal can be either a business goal or a system goal. A business goal expresses goals regarding the business or state of the business affairs the individual or organization wishes to achieve. A system goal expresses goals the target system should achieve and generally describes the functional requirements of the target information system.

- A **Softgoal** is a condition or state of affairs in the world that the actor would like to achieve, but unlike in the concept of (hard) goal, there are no clear-cut criteria for whether the condition is achieved, and it is up to subjective judgement and interpretation of the modeller to judge whether a particular state of affairs in fact achieves sufficiently the stated softgoal. Softgoals are often used to describe qualities and non-functional aspects such as security, robustness, performance, usability, etc.
- A **Task** specifies a particular way of doing something. When a task is part of the decomposition of a (higher-level) task, this restricts the higher-level task to that particular course of action. Tasks can also be seen as the solutions in the target system, which will address (or operationalize) goals and softgoals. These solutions provide operations, processes, data representations, structuring, constraints and agents in the target system to meet the needs stated in the goals and softgoals.
- A **Resource** is a physical or informational entity, for which the main concern is whether it is available.
- A **Belief** is used to represent design rationale. Beliefs make it possible for domain characteristics to be considered and properly reflected in the decision-making process, hence facilitating later review, justification and change of the system, as well as enhancing traceability.

7.4 GRL links

7.4.1 ElementLink

An *ElementLink* connects two GRL linkable elements and represents the intentional relationship existing between them. *ElementLink* abstracts the commonalities of decomposition, contribution and dependency links (see Figure 10).

a) Abstract grammar

i) Attributes

- Inherits attributes from *GRLmodelElement* (see Figure 9).

ii) Relationships

- Inherits relationships from *GRLmodelElement*.
- Contained by *GRLspec* (1): An *ElementLink* is contained in the GRL specification (see Figure 8).
- Association with *GRLLinkableElement* (src, 1): An *ElementLink* has one source GRL linkable element.
- Association with *GRLLinkableElement* (dest, 1): An *ElementLink* has one destination GRL linkable element.
- *ElementLink* is a superclass of *Contribution*, *Dependency* and *Decomposition*.

iii) Constraints

- Inherits constraints from *GRLmodelElement*.
- All instances of *ElementLink* shall appear in one of its subclasses (that is, metaclass *ElementLink* is abstract).
- The source and destination GRL linkable elements shall be different.

b) Concrete grammar

The concrete syntax for *ElementLink* is further defined in its subclasses.

i) Relationships

- Inherits relationships from *GRLmodelElement* (see Figure 30).

- Association with *LinkRef* (0..*): An *ElementLink* may have link references (see Figure 30).

c) *Semantics*

An *ElementLink* is a directed link that connects a source actor definition, intentional element or indicator to a different destination actor definition, intentional element or indicator. The semantics of the link is provided by the subclass used.

7.4.2 Contribution

A *Contribution* link describes how a source intentional element or source indicator contributes to the satisfaction of a destination intentional element. A contribution is an effect that is a primary desire during modelling, whereas a correlation expresses knowledge about interactions between intentional elements in different categories. A correlation link is the same as a contribution link except that the correlation is not an explicit desire, but is a side-effect and that correlations are only used with intentional elements and not with indicators (see Figure 10).

a) *Abstract grammar*

i) **Attributes**

- Inherits attributes from *ElementLink*.
- contribution (*ContributionType*): The qualitative level of contribution. Default value is Unknown.
- quantitativeContribution (*Integer*): The quantitative level of contribution. Default value is 0 (see clause 9.2.2).
- correlation (Boolean): Indicates whether the link is a regular contribution (false) or a correlation (true). Default value is false.

ii) **Relationships**

- Inherits relationships from *ElementLink*.
- Uses *ContributionType* enumeration.

iii) **Constraints**

- Inherits constraints from *ElementLink*.
- Actor definitions can neither be the source nor the destination of a contribution.
- The destination linkable element (dest) shall not be an intentional element of type Resource or Belief.
- The destination linkable element (dest) shall not be an indicator.
- If the source linkable element (src) is an indicator, then correlation is false.
- $\text{quantitativeContribution} \geq -100$ and $\text{quantitativeContribution} \leq 100$.

b) *Concrete grammar*

A *Contribution* does not have a visual representation, but link references (*LinkRef*) in GRL diagrams do provide a graphical representation.

i) **Relationships**

- Inherits relationships from *ElementLink*.

c) *Semantics*

A *Contribution* defines the level of impact that the satisfaction of a source intentional element or indicator has on the satisfaction of a destination intentional element. If the impact is qualitative (positive or negative, sufficient or insufficient; see the contributions in clause 7.4.3), then contribution will be used in goal model evaluations. The impact can be

also quantitative (value in $[-100, 100]$) in which case `quantitativeContribution` will be used in goal model evaluations. A correlation link (`correlation` is true) has the same impact on an evaluation as regular contribution links, but it emphasizes side-effects between intentional elements in different categories or actor definitions. Correlations are not used with indicators.

Only the relevant contribution attribute is considered depending on the type of analysis (qualitative or quantitative). It is not required for `contribution` and `quantitativeContribution` to be consistent, as modellers may want to use only one type of analysis (qualitative or quantitative). However, it is recommended to keep them consistent if the modellers intend to switch between different types of analysis.

NOTE – Semantic variation: Modellers are allowed to impose additional stylistic constraints on the well-formedness of contributions. For example, as it is possible for contribution links to be qualitative and partial, while goals, tasks and resources represent clear-cut concepts, a constraint is able specify that intentional elements only contribute to softgoals, though the destination intentional element has to be a softgoal.

d) *Model*

None.

e) *Example*

Figure 15 is a GRL diagram that shows three contributions and two correlations linking five intentional elements.

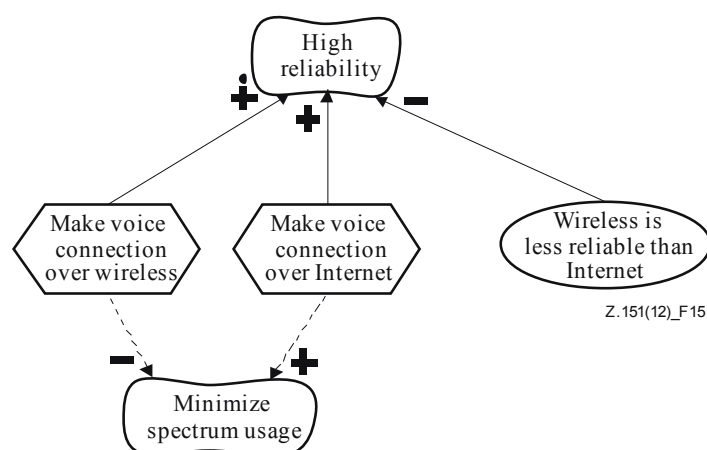


Figure 15 – Example: GRL contributions and correlations

- "Make Voice Connection Over Wireless" has a positive and sufficient contribution on "High Reliability".
- "Make Voice Connection Over Internet" has some positive contribution on "High Reliability".
- "Wireless is less reliable than Internet" has some negative contribution on "High Reliability".
- "Make Voice Connection Over Wireless" has some negative correlation (side-effect) on "Minimize Spectrum Usage".
- "Make Voice Connection Over Internet" has some positive correlation (side-effect) on "Minimize Spectrum Usage".

See clauses 7.4.3 and 7.8.7 for the details of the concrete syntax.

7.4.3 ContributionType

A qualitative contribution level in a *Contribution* link can take one of the following values: Make, Help, SomePositive, Unknown, SomeNegative, Hurt, Break (see Figure 10).

a) *Abstract grammar*

i) **Attributes**

- None (enumeration metaclass).

ii) **Relationships**

- Used by *Contribution* and by *ContributionChange* (see Figure 29).

iii) **Constraints**

- None.

b) *Concrete grammar*

Figure 16 lists the icons used to annotate GRL contribution links (including correlation links) according to the value of their (qualitative) contribution. See usage in *LinkRef*, clause 7.8.7.



Figure 16 – Symbol: GRL contribution types

c) *Semantics*

The qualitative contribution of a source intentional element or indicator to a destination intentional element can be one of the following values based on the degree (positive or negative) and sufficiency of the contribution to the satisfaction of the destination intentional element:

- **Make:** The contribution is positive and sufficient.
- **Help:** The contribution is positive but not sufficient.
- **SomePositive:** The contribution is positive, but the extent of the contribution is unknown.
- **Unknown:** There is some contribution, but the extent and the degree (positive or negative) of the contribution is unknown.
- **SomeNegative:** The contribution is negative, but the extent of the contribution is unknown.
- **Break:** The contribution of the contributing element is negative and sufficient.
- **Hurt:** The contribution is negative but not sufficient.

d) *Model*

An alternative presentation of the Unknown contribution is to simply omit the Unknown icon on the contribution link. This makes GRL diagrams less cluttered, without loss of information.

7.4.4 Dependency

A *Dependency* describes how a source actor definition (the *depender*) depends on a destination actor definition (the *dependee*) for an intentional element or indicator (the *dependum*). Often, the modeller will use two consecutive dependency links (from depender to dependum, and from

dependum to dependee) to express detailed dependencies, but dependencies can be used in more generic situations as well (see Figure 10).

The dependum specifies *what* the dependency is about, i.e., the intentional element or indicator around which a dependency relationship centres. With an intentional element or indicator as a source of the dependency, the depender may specify *why* it depends on the dependee for the dependum. With an intentional element or indicator as a target of the dependency, the dependee may specify *how* it is required to provide or satisfy the dependum.

a) *Abstract grammar*

i) **Attributes**

- Inherits attributes from *ElementLink*.

ii) **Relationships**

- Inherits relationships from *ElementLink*.

iii) **Constraints**

- Inherits constraints from *ElementLink*.
- Intentional elements of type *Belief* can neither be the source nor the destination of a dependency.
- At least one of the GRL linkable elements linked by a dependency link shall be an actor definition or an intentional element contained in an actor definition or an indicator contained in an actor definition.
- If the source and destination linked by a dependency link are intentional elements or indicators, then these intentional elements and indicators shall not be contained in the same actor definition.

b) *Concrete grammar*

A *Dependency* does not have a visual representation, but link references (*LinkRef*) in GRL diagrams do provide a graphical representation for dependencies.

i) **Relationships**

- Inherits relationships from *ElementLink*.

c) *Semantics*

Dependencies enable reasoning about how actor definitions depend on each other to achieve their goals. The satisfaction level of the depender may be limited by the ability of the dependee to provide the dependum to the depender.

Dependency links can be used in a number of configurations including but not limited to the ones described below. According to the required level of detail, intentional elements or indicators inside actor definitions can be used as source and/or destination of a dependency link. Assume *Depender* and *Dependee* are different instances of *Actor* definition, *D1* and *D2* are different instances of *Dependency*, and *Why*, *How* and *What* are different instances of *GRLContainableElement*. *Why* is inside *Depender*, *How* is inside *Dependee* and *What* is not inside any actor definition. A named arrow ($\text{---}D\text{---}$) indicates the presence of a dependency link *D* between the source and target GRL linkable elements involved.

i) *Depender* $\text{---}D1\text{---}$ *What* $\text{---}D2\text{---}$ *Dependee*

- *Depender* depends on *Dependee* for *What*. *What* represents the dependum.

ii) *Depender* $\text{---}D1\text{---}$ *How*

- *Depender* depends on *Dependee* for *How*. The dependum is unknown.

- iii) *Why* —D1→ *What* —D2→ *Dependee*
 - *Why* in *Depender* depends on *Dependee* for *What*. *What* represents the dependum.
- iv) *Why* —D1→ *How*
 - *Why* in *Depender* depends on *Dependee* for *How*. The dependum is unknown.
- v) *Depender* —D1→ *Dependee*
 - *Depender* depends on *Dependee*. The dependum is unknown.
- vi) *Why* —D1→ *What* —D2→ *How*
 - *Why* in *Depender* depends on *How* in *Dependee* for *What*. *What* represents the dependum.

d) *Model*

None.

e) *Examples*

The following examples are GRL diagrams illustrating the six configurations discussed in the semantics clause. The same numbering scheme is used. Explanations follow each diagram. The types of intentional elements used here are simply examples and do not preclude other usages in dependencies.

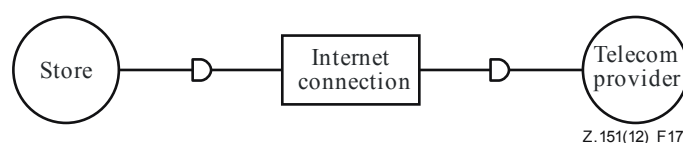


Figure 17 – Example: GRL dependencies (configuration 1)

- i) *The Store depends on the Telecom Provider to provide an Internet Connection* (Figure 17). This is a configuration that focuses solely on strategic dependencies between actors. *Why* and *how* the dependum is provided are unknown.

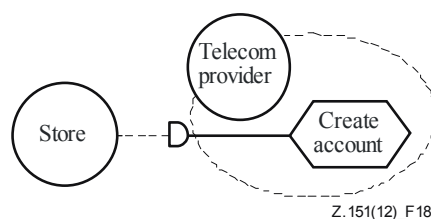


Figure 18 – Example: GRL dependencies (configuration 2)

- ii) *The Store depends on the Telecom Provider to create an account* (Figure 18). The dependum and *why* it is required are unknown.

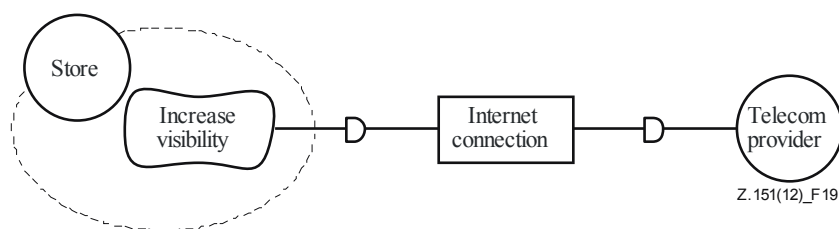


Figure 19 – Example: GRL dependencies (configuration 3)

- iii) *To increase its visibility, the Store depends on the Telecom Provider to provide an Internet Connection* (Figure 19). *How* the dependum is provided is unknown.

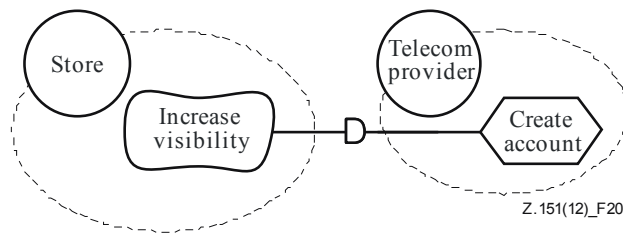


Figure 20 – Example: GRL dependencies (configuration 4)

- iv) To increase its visibility, the Store depends on the Telecom Provider to create an account (Figure 20). The dependum is unknown.

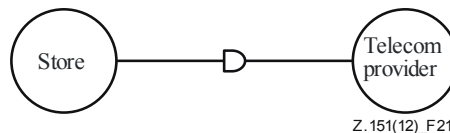


Figure 21 – Example: GRL dependencies (configuration 5)

- v) The Store depends on the Telecom Provider (Figure 21). This is a configuration that is typical of preliminary goal models that require further refinement. A dependency is identified, but what, why and how are still unknown.

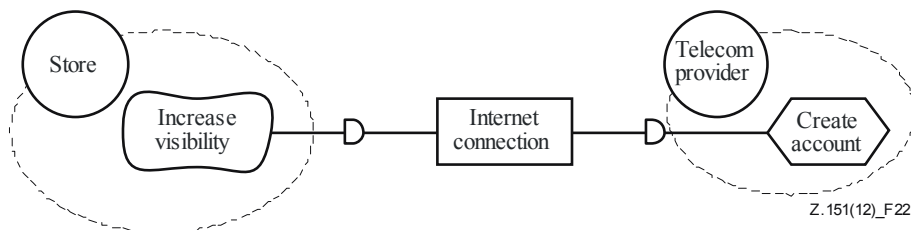


Figure 22 – Example: GRL dependencies (configuration 6)

- vi) To increase its visibility, the Store depends on the Telecom Provider to provide an Internet Connection by creating an account (Figure 22). This is a configuration that details the dependum (the Internet connection) together with why it is required and how it is provided.

7.4.5 Decomposition

Decomposition links provide the ability to define what source intentional elements need to be satisfied or available in order for a target intentional element to be satisfied. The type of decomposition (AND, XOR, IOR) is specified by the *decompositionType* attribute of the target intentional element. Therefore, an intentional element can be decomposed using one decomposition type only (see Figure 10).

a) Abstract grammar

i) Attributes

- Inherits attributes from *ElementLink*.

ii) Relationships

- Inherits relationships from *ElementLink*.

iii) Constraints

- Inherits constraints from *ElementLink*.
- Actor definitions can neither be the source nor the destination of a decomposition.

- Intentional elements of type *Belief* can neither be the source nor the destination of a decomposition.
- *Indicators* cannot be the destination of a decomposition.

b) *Concrete grammar*

A *Decomposition* does not have a visual representation, but link references (*LinkRef*) in GRL diagrams do provide a graphical representation.

i) **Relationships**

- Inherits relationships from *ElementLink*.

c) *Semantics*

Decomposition links connect the essential parts of an intentional element, which include subtasks that shall be performed, subgoals that shall be achieved, resources that shall be accessible and softgoals that shall be satisfied. There is no ordering between the decomposing elements.

A *Decomposition* link enables the hierarchical decomposition (AND) of a target intentional element by a source element. A target intentional element can be decomposed into many source intentional elements using as many decomposition links. All of the source intentional elements are necessary for the target intentional element to be satisfied.

A *Decomposition* link also enables the description of alternative means of satisfying a target intentional element (XOR for mutually exclusive alternatives, or IOR for alternatives that are not mutually exclusive). One of the source intentional elements is sufficient for the target intentional element to be satisfied.

NOTE – **Semantic variation:** Modellers are allowed to impose additional stylistic constraints on the well-formedness of decomposition links. For example, tasks could be limited to AND decomposition, so that if the target of a decomposition link is an intentional element of type *Task*, then its decomposition type has to be AND.

d) *Model*

None.

e) *Examples*

Figure 23 shows two representations of XOR decompositions. On the left, the goal is decomposed into two mutually exclusive alternatives presented as tasks. On the right, the same decomposition is presented using means-end relationships (with the same meaning). See clauses 7.4.6 and 7.8.7 for the details of the concrete syntax.

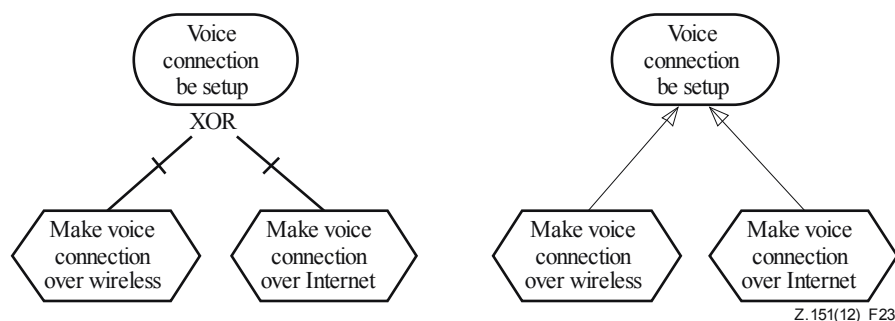


Figure 23 – Example: GRL XOR decomposition: normal (left) and means-end (right) presentations

7.4.6 DecompositionType

a) *Abstract grammar*

An intentional element can be decomposed in one of three ways according to its *decompositionType* attribute: AND, XOR or IOR (see Figure 10).

i) **Attributes**

- None (enumeration metaclass).

ii) **Relationships**

- Used by *IntentionalElement*.

iii) **Constraints**

None.

b) *Concrete grammar*

There is no specific icon for decomposition types. The name of the decomposition type itself (AND, XOR or IOR) is used. See usage in *LinkRef*, clause 7.8.7.

c) *Semantics*

- AND decomposition: The satisfaction of each of the sub-intentional elements is necessary to achieve the target.
- XOR decomposition: The satisfaction of one and only one of the sub-intentional elements is necessary to achieve the target.
- IOR decomposition: The satisfaction of one of the sub-intentional elements is sufficient to achieve the target, but many sub-intentional elements can be satisfied.

7.5 GRL strategies

GRL strategies are sets of initial evaluation values given to some containable elements (i.e., intentional elements and indicators) in a GRL model. For indicators, the initial evaluation values are based on real-world measures. These evaluation values, which can be quantitative or qualitative, are satisfaction levels that can then be propagated to the other intentional elements and indicators in the GRL model through the various decomposition, contribution and dependency links connecting them. Evaluations are used to assess how well goals in a model are achieved in a given context, which enables the selection of alternatives that represent appropriate trade-offs among the often-conflicting goals of the stakeholders/actors involved. A good strategy provides rationale and documentation for decisions leading to requirements, providing better context for standards/system developers and implementers while avoiding unnecessary re-evaluations of worse alternative strategies.

NOTE 1 – The name *intElement* is used for backward compatibility with previous versions of this Recommendation. A more appropriate name would be *contElement*.

NOTE 2 – The name *group* is used for backward compatibility with previous versions of this Recommendation. A more appropriate name would be *groups*.

a) *Abstract grammar*

i) **Attributes**

- Inherits attributes from *GRLmodelElement*.

ii) **Relationships**

- Inherits relationships from *GRLmodelElement*.
- Contained by *GRLspec* (1): An *EvaluationStrategy* is contained in the GRL specification (see Figure 8).
- Composition of *Evaluation* (0..*): An *EvaluationStrategy* may contain evaluations.
- Association with *StrategiesGroup* (1..*): An *EvaluationStrategy* is referenced by at least one group of strategies.
- Association with *EvaluationStrategy* (parentStrategies, 0..*): An *EvaluationStrategy* may be included by evaluation strategies.
- Association with *EvaluationStrategy* (includedStrategies, 0..*) {ordered}: An *EvaluationStrategy* may have an ordered collection of included evaluation strategies.

iii) **Constraints**

- Inherits constraints from *GRLmodelElement*.
- The strategy containment hierarchy established by the includedStrategies relationship does not contain any cycles (i.e., an *EvaluationStrategy* shall not appear more than once on a path from a top node to a leaf node in the containment hierarchy).
- Let LE be the list of *Evaluations* of an *EvaluationStrategy*. Let LCE be the list of *GRLContainableElements* associated with at least one *Evaluation* in LE. LCE shall not contain the same *GRLContainableElement* more than once.
- Let LE be the list of *Evaluations* of an *EvaluationStrategy* and the list of *Evaluations* of all includedStrategies of the same *EvaluationStrategy*. Let LEI be the subset of LE that contains only those *Evaluations* that are associated with a *GRLContainableElement* of type *Indicator*. Let LI be the list of *Indicators* associated with at least one *Evaluation* in LEI. For each *Indicator* in LI, there exists at least one *Evaluation* in LEI that is associated with the indicator and that is also associated with an *IndicatorConversion*. For each *Indicator* in LI, there exists at least one *Evaluation* in LEI that is associated with the indicator and that also contains an *IndicatorEvaluation*.

b) *Concrete grammar*

An *EvaluationStrategy* does not have a visual representation.

i) **Relationships**

- Inherits relationships from *GRLmodelElement*.

c) *Semantics*

The *Evaluations* contained in an *EvaluationStrategy* represent an initial context for the evaluation of a GRL model. Using a model evaluation algorithm (see clause 11.1), the initial values specified in the evaluations are propagated to the intentional elements and indicators that do not have any initial evaluation value, through the element links that connect them.

If an evaluation strategy contains another evaluation strategy, the evaluation algorithm considers the union of the *Evaluations*. For conflicting *Evaluations* (i.e., evaluations of the same containable element) however, the *Evaluation* of the containing evaluation strategy overrides the *Evaluation* of the contained evaluation strategy. If more than one evaluation strategy is included, then the last evaluation strategy in the ordered list of included evaluation strategies takes precedence over the previous ones (i.e., the evaluations are

applied beginning with the first included evaluation strategy in the ordered list; then each other evaluation strategy is applied up to the last included evaluation strategy, and finally the evaluations of the including evaluation strategy are applied).

7.5.3 Evaluation

An *Evaluation* provides initial quantitative and qualitative evaluation values to a containable element (see Figure 24).

a) Abstract grammar

i) Attributes

- *evaluation* (*Integer*): Initial quantitative satisfaction value (also called evaluation value) of the associated intentional element. Default value is 0 (see clause 9.2.2).
- *qualitativeEvaluation* (*QualitativeLabel*): Initial qualitative satisfaction value (also called evaluation value) of the associated intentional element. Default value is None.
- *exceeds* (*Boolean*): Indicates whether the quantitative and qualitative evaluations exceed expectations (true) or not (false). Default value is false.

ii) Relationships

- Contained by *EvaluationStrategy* (1): An *Evaluation* is contained in one evaluation strategy.
- Composition of *IndicatorEvaluation* (0..1): An *Evaluation* may contain one set of real-world values (see Figure 26).
- Association with *GRLContainableElement* (1): An *Evaluation* provides initial evaluation values to one containable element.
- Association with *IndicatorConversion* (0..1): An *Evaluation* may have one conversion method to convert real-world values into GRL evaluation values (see Figure 26).
- Uses *QualitativeLabel* enumeration.

iii) Constraints

- $\text{evaluation} \geq -100$ and $\text{evaluation} \leq 100$.
- If *exceeds* is true, then *evaluation* is 100 and *qualitativeEvaluation* is Satisfied.
- If the *intElement* of an *Evaluation* is of type *Indicator*, then the conversion shall be specified for the *Evaluation*.
- If the *intElement* of an *Evaluation* is of type *Indicator*, then the unit of the *Indicator* of the *Evaluation* is the same as the unit of the *IndicatorConversion* of the *Evaluation*.

b) Concrete grammar

An *Evaluation* does not have a visual representation. However, it may impact the presentation of intentional (and indicator) element references (see clause 7.8.5).

c) Semantics

An *Evaluation* defines the initial level of satisfaction of an intentional element and indicator. In the case of indicators, the initial level of satisfaction is based on a real-world measure (*IndicatorEvaluation*) and a conversion method (*IndicatorConversion*). If the level of satisfaction is qualitative (see the qualitative label types in clause 7.5.4), then *qualitativeEvaluation* will be used in goal model evaluations. If the level of satisfaction is quantitative (integer value between –100 for sufficiently denied and +100 for sufficiently satisfied, inclusively), then *evaluation* will be used in goal model evaluations. An evaluation value of 0 means that the intentional element is neither satisfied nor denied.

The **exceeds** attribute captures the fact that the initial level of satisfaction goes beyond expectations. This attribute is mostly used with indicators to indicate that the measured real-world value actually exceeds the agreed upon value which, when converted into a GRL evaluation value, results in full satisfaction.

Only the relevant evaluation attribute is considered depending on the type of analysis (qualitative or quantitative). It is not required for **Evaluation** and **qualitativeEvaluation** to be consistent as modellers may want to use only one type of analysis (qualitative or quantitative). However, it is recommended to keep them consistent if the modellers intend to switch between different types of analysis.

7.5.4 QualitativeLabel

A *QualitativeLabel* represents the qualitative satisfaction level of an intentional element, indicator or actor. It is also used by conversions from qualitative real-world values to GRL evaluation values. It can be one of the following values: Denied, WeaklyDenied, WeaklySatisfied, Satisfied, Conflict, Unknown and None (see Figure 24).

a) Abstract grammar

i) Attributes

- None (enumeration metaclass).

ii) Relationships

- Used by *Evaluation* and *QualToQMapping* (see Figure 26).

iii) Constraints

None.

b) Concrete grammar

Figure 25 lists the icons that are used to annotate GRL intentional elements, indicators and actors according to their (qualitative) satisfaction level for a given strategy evaluation. See usage in *ActorRef*, clause 7.8.3, *IntentionalElementRef*, clause 7.8.5, *CollapsedActorRef*, clause 7.8.6.



Figure 25 – Symbol: GRL qualitative labels

c) Semantics

The qualitative satisfaction level of an intentional element or indicator can be one of the following values based on the degree (positive or negative) and magnitude of the satisfaction:

- Denied: The intentional element or indicator is sufficiently dissatisfied.
- WeaklyDenied: The intentional element or indicator is partially dissatisfied.
- WeaklySatisfied: The intentional element or indicator is partially satisfied.
- Satisfied: The intentional element or indicator is sufficiently satisfied.
- Conflict: There are arguments strongly in favour and strongly against the satisfaction of the intentional element or indicator.
- Unknown: The satisfaction level of the intentional element or indicator is unknown.
- None: The intentional element or indicator is neither satisfied nor dissatisfied.

d) *Model*

An alternative presentation of the None satisfaction level is to simply omit the None icon on the intentional element or indicator. This makes GRL diagrams less crowded, without loss of information.

7.6 GRL indicators

GRL indicators represent real-world measurements in the GRL model. Two types of conversions translate a quantitative or qualitative real-world measurement into a quantitative or qualitative GRL evaluation value which can then be taken into consideration during GRL model analysis.

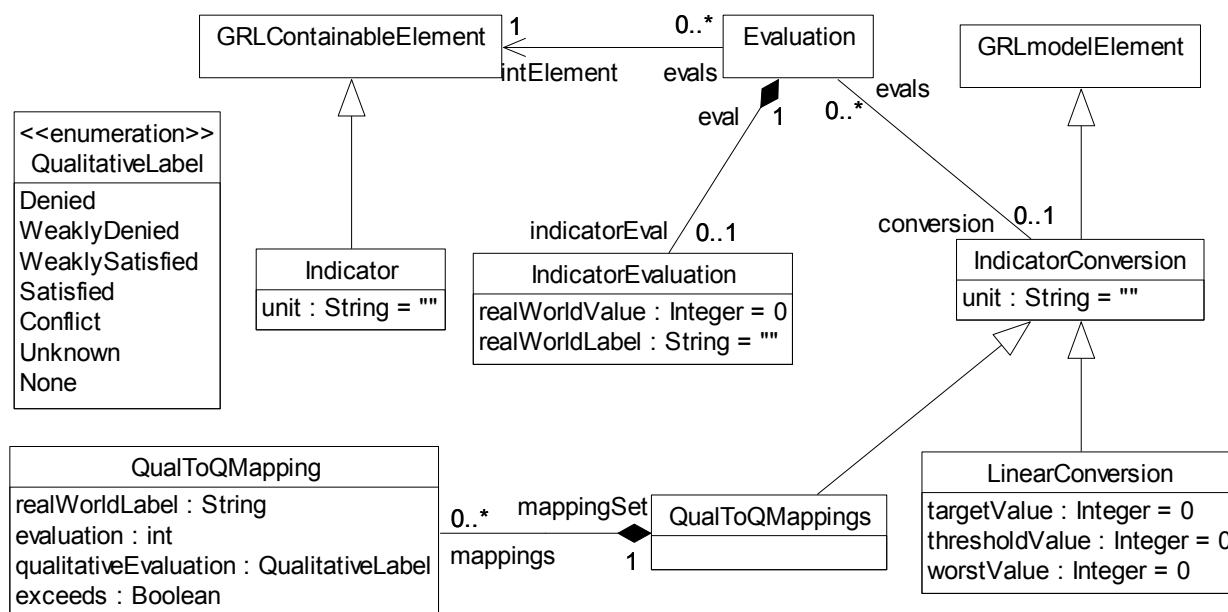


Figure 26 – Abstract grammar: GRL indicators

7.6.1 Indicator

An *Indicator* is a GRL containable element used for models that base GRL model analysis on real-world measurements. An indicator substantiates qualitative or quantitative evaluation values in the GRL model with the help of real-world data, allowing GRL model analysis to be more accurate and realistic than with satisfaction values only. Indicators may be included in actor definitions, they can be linked to each other in different ways and they can be given a quantitative or qualitative importance level when included in an actor definition (see Figure 26).

a) *Abstract grammar*

i) **Attributes**

- Inherits attributes from *GRLContainableElement*.
- unit (String): Indicates the measurement unit of the real-world values. Default value is the empty String "".

ii) **Relationships**

- Inherits relationships from *GRLContainableElement*.

iii) **Constraints**

- Inherits constraints from *GRLContainableElement*.
- If an *Indicator* is associated with an *Actor* definition, then there is only one *Indicator* with this name associated with the *Actor* definition.

- If an *Indicator* is not associated with any *Actor* definition, then there is only one *Indicator* with this name that is not associated with any *Actor* definition.

b) *Concrete grammar*

An indicator does not have a visual representation, but intentional element (or indicator) references (*IntentionalElementRef*) in GRL diagrams do have a graphical representation.

i) **Relationships**

- Inherits relationships from *GRLContainableElement*.

c) *Semantics*

An *Indicator* describes a qualitative or quantitative real-world measurement (*IndicatorEvaluation* contained by *Evaluation* associated with superclass *GRLContainableElement* of the indicator). This measurement is converted into a qualitative or quantitative GRL evaluation value (*Evaluation*) as defined by a conversion method (*IndicatorConversion* associated with the same *Evaluation*). An *Indicator* may have several *IndicatorEvaluations*, i.e., one for each *EvaluationStrategy*.

An indicator contained in an *Actor* definition belongs to this actor definition and therefore describes an actor-specific measurement.

The unit attribute of *Indicator* defines the unit of the real-world measurements of the indicator as defined in the previous paragraph (i.e., *realWorldValue* or *realWorldLabel* of *IndicatorEvaluation*). If decimal points are desired for the *realWorldValue*, then the unit is adjusted as the *realWorldValue* is always an integer value. For example, to set the *realWorldValue* to \$1.15, the *realWorldValue* is set to 115 and the unit of the *Indicator* is set to "cents". Similarly, to set the *realWorldValue* to 3.5%, the *realWorldValue* is set to 35 and the unit is set to "1/10 of a percent".

The unit of *Indicator* and unit of all its *IndicatorConversions* applied to the real-world measurements of the indicator (i.e., the *realWorldValues* or *realWorldLabels* of all *IndicatorEvaluations*) shall be the same.

d) *Model*

None.

e) *Examples*

Figure 27 is a GRL diagram with one indicator which captures the failure rate of voice connections over the Internet.

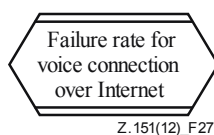


Figure 27 – Example: GRL indicator

See clauses 7.1.4 and 7.8.5 for the details of the concrete syntax.

The GRL diagram in Figure 28 shows a task and an indicator contained by a "Telecom Provider" actor. The indicator has an importance value of 40 while the task has an importance value of 100. The *Contribution* link between the indicator and the task has a quantitativeContribution of 100. The indicator also has a *Dependency* link with a task in a "Technician" actor.

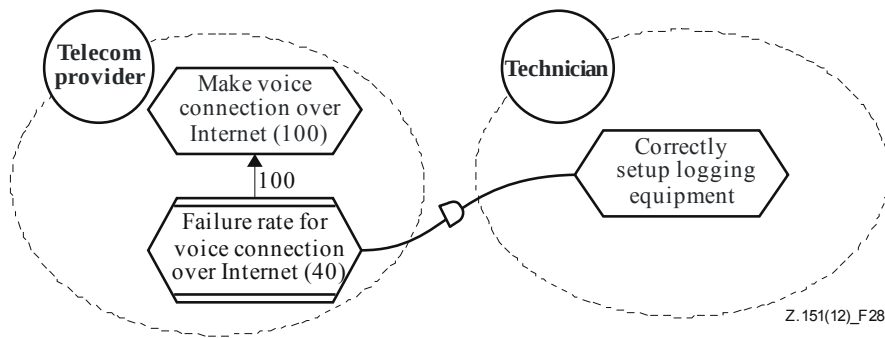


Figure 28 – Example: GRL model with an indicator

7.6.2 IndicatorEvaluation

An *IndicatorEvaluation* provides initial quantitative and qualitative real-world values to a containable element of type *Indicator* (see Figure 26).

a) Abstract grammar

i) Attributes

- **realWorldValue** (*Integer*): Initial quantitative real-world value of the associated indicator. Default value is 0 (see clause 9.2.2).
- **realWorldLabel** (*String*): Initial qualitative real-world value of the associated indicator. Default value is the empty String "".

ii) Relationships

- Contained by *Evaluation* (1): An *IndicatorEvaluation* is contained in one evaluation.

iii) Constraints

- The *GRLContainableElement* of the *Evaluation* containing an *IndicatorEvaluation* shall be of type *Indicator*.

b) Concrete grammar

An *IndicatorEvaluation* does not have a visual representation.

c) Semantics

An *IndicatorEvaluation* defines the initial real-world values of an indicator. If the real-world value is qualitative, then **realWorldLabel** will be used when converting the real-world value into GRL evaluations. If the level of satisfaction is quantitative, then **realWorldValue** will be used by the conversion. The conversion method is defined by the *IndicatorConversion* of the *Evaluation* containing the *IndicatorEvaluation*.

Only the relevant real-world value is considered depending on the type of conversion. It is not required for **realWorldValue** and **realWorldLabel** to be consistent as modellers may want to use only one type of conversion. However, it is recommended to keep them consistent if the modellers intend to switch between different types of conversions.

7.6.3 IndicatorConversion

An *IndicatorConversion* defines which quantitative and qualitative real-world values to convert into GRL evaluation values (see Figure 26).

a) Abstract grammar

i) Attributes

- Inherits attributes from *GRLmodelElement* (see Figure 9).

- **unit (String):** Indicates the measurement unit of the real-world values. Default value is the empty String "".

ii) Relationships

- Inherits relationships from *GRLmodelElement*.
- Contained by *GRLspec* (1): An *IndicatorConversion* is contained in the GRL specification (see Figure 8).
- Association with *Evaluation* (0..*): An *IndicatorConversion* may convert the initial real-world values associated with many *Evaluations*.
- *IndicatorConversion* is a superclass of *LinearConversion* and *QualToQMappings*.

iii) Constraints

- Inherits constraints from *GRLmodelElement*.
- All instances of *IndicatorConversion* shall appear in one of its subclasses (that is, metaclass *IndicatorConversion* is abstract).

b) Concrete grammar

An *IndicatorConversion* does not have a visual representation.

i) Relationships

- Inherits attributes from *GRLmodelElement*.

c) Semantics

The subclasses of *IndicatorConversion* specify how the initial real-world values of an indicator (*IndicatorEvaluation* of the associated *Evaluation*) are converted into GRL evaluation values (*Evaluation*). These subclasses may have additional attributes and relationships as required for the conversion.

An *IndicatorConversion* may only be chosen for an *Indicator*, if the unit of both is the same. For more details on units and decimal points for real-world values, see *Indicator*, clause 7.6.1.

7.6.4 LinearConversion

A *LinearConversion* converts quantitative real-world values into quantitative and qualitative GRL evaluation values based on linear extrapolation (see Figure 26).

a) Abstract grammar

i) Attributes

- Inherits attributes from *IndicatorConversion*.
- **targetValue (Integer):** Indicates the best result that can be measured for this quantitative real-world value. Default value is 0 (see clause 9.2.2).
- **thresholdValue (Integer):** Indicates the measurement result where the quantitative real-world value is neither negative nor positive but neutral. Default value is 0 (see clause 9.2.2).
- **worstValue (Integer):** Indicates the worst result that can be measured for this quantitative real-world value. Default value is 0 (see clause 9.2.2).

ii) Relationships

- Inherits relationships from *IndicatorConversion*.

iii) Constraints

- Inherits constraints from *IndicatorConversion*.

- The three value attributes of a *LinearConversion* are either sorted in ascending or descending order (i.e., either (i) the *targetValue* is greater than or equal to the *thresholdValue* and the *thresholdValue* is greater than or equal to the *worstValue* or (ii) the *targetValue* is less than or equal to the *thresholdValue* and the *thresholdValue* is less than or equal to the *worstValue*).

b) *Concrete grammar*

A *LinearConversion* does not have a visual representation.

i) **Relationships**

- Inherits relationships from *IndicatorConversion*.

c) *Semantics*

A *LinearConversion* specifies three real-world values, i.e., the *targetValue*, the *thresholdValue* and the *worstValue*, and then maps the *targetValue* to the GRL evaluation value 100, the *thresholdValue* to the GRL evaluation value 0 and the *worstValue* to the GRL evaluation value –100. An *IndicatorEvaluation* defines a quantitative real-world value *R* (*realWorldValue*) that is to be converted by the *LinearConversion* associated with the *IndicatorEvaluation* through an *Evaluation*. A real-world value *R* that falls between the *targetValue* and the *thresholdValue* is extrapolated to the [0, 100] GRL evaluation value range according to the formula: $(R - \text{thresholdValue}) / (\text{targetValue} - \text{thresholdValue}) \times 100$. A real-world value *R* that falls between the *thresholdValue* and the *worstValue* is extrapolated to the [–100, 0] GRL evaluation value range according to the formula: $(R - \text{thresholdValue}) / (\text{worstValue} - \text{thresholdValue}) \times (-100)$.

If the *targetValue* is the same as the *thresholdValue* or the *worstValue* is the same as the *thresholdValue*, then the result of the conversion is the GRL evaluation value of 0. If *R* is outside the [*targetValue*, *worstValue*] real-world value range, then the result of the conversion is the GRL evaluation value of 100, if *R* is beyond the *targetValue*. In this case, the *exceeds* value is set to true. In all other cases, the *exceeds* value is set to false. If *R* is beyond the *worstValue*, then the result of the conversion is the GRL evaluation value of –100.

NOTE – **Semantic variation:** The GRL qualitativeEvaluation value is calculated based on the resulting GRL evaluation value and a mapping from quantitative GRL evaluation values to qualitative GRL evaluation values. One possible mapping is shown in Table 1, but modellers are allowed to define any other mapping deemed appropriate.

Table 1 – GRL mapping from quantitative to qualitative evaluation values

<i>Quantitative value</i>	<i>Qualitative value</i>
–100	<i>Denied</i>
(–100, 0)	<i>WeaklyDenied</i>
0	<i>None</i>
(0, 100)	<i>WeaklySatisfied</i>
100	<i>Satisfied</i>

d) *Model*

None.

e) *Examples*

Table 2 gives an example for a linear conversion based on the GRL model in Figure 28, assuming that the unit of measurement for the indicator in Figure 28 is percentage. The linear conversion specifies the *targetValue* as 0%, the *thresholdValue* as 5% and the *worstValue* as 100%, and then maps these values to the GRL evaluation values 100, 0 and –100, respectively. Three sample real-world values and their corresponding GRL evaluation values are also given. The first two use the first formula, while the last one uses the second formula specified in this clause.

Table 2 – GRL linear conversion

	<i>Target</i>	<i>Threshold</i>	<i>Worst</i>	<i>Example 1</i>	<i>Example 2</i>	<i>Example 3</i>
<i>Real-world value</i>	0%	5%	100%	1%	3.5%	45%
<i>GRL evaluation value</i>	100	0	–100	$(1-5)/(0-5) \times 100 = 80$	$(3.5-5)/(0-5) \times 100 = 30$	$(45-5)/(100-5) \times (-100) = -42$

7.6.5 QualToQMappings

A *QualToQMappings* converts qualitative real-world values into quantitative and qualitative GRL evaluation values based on explicit mappings (see Figure 26).

a) *Abstract grammar*

i) **Attributes**

- Inherits attributes from *IndicatorConversion*.

ii) **Relationships**

- Inherits relationships from *IndicatorConversion*.
- Composition of *QualToQMapping* (0..*): A *QualToQMappings* conversion may contain many mappings.

iii) **Constraints**

- Inherits constraints from *IndicatorConversion*.

b) *Concrete grammar*

A *QualToQMappings* conversion does not have a visual representation.

i) **Relationships**

- Inherits relationships from *IndicatorConversion*.

c) *Semantics*

None (*QualToQMappings* is a structural concept only).

7.6.6 QualToQMapping

A *QualToQMapping* converts a single qualitative real-world value into quantitative and qualitative GRL evaluation values based on explicit mappings (see Figure 26).

a) *Abstract grammar*

i) **Attributes**

- *realWorldLabel* (String): Indicates a qualitative, measurable, real-world value.
- *evaluation* (Integer): Indicates the mapped initial quantitative satisfaction value (also called evaluation value) (see clause 9.2.2).