BRITISH STANDARD

BS ISO/IEC 15909-2:2009

Software and system engineering — High-level Petri nets

Part 2: Transfer Format

ICS 35.080

NO COPYING WITHOUT BSI PERMISSION EXCEPT AS PERMITTED BY COPYRIGHT LAW

Document Sharing Hub
British Standards

Full document details: BS ISO/IEC 15909-2:2009

National foreword

This British Standard is the UK implementation of ISO/IEC 15909-2:2009.

The UK participation in its preparation was entrusted to Technical Committee IST/15, Software and systems engineering.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 January 2010.

© BSI 2010

ISBN 978 0 580 59389 5

Amendments/corrigenda issued since publication

Date	Comments

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 15909-2:2009

INTERNATIONAL STANDARD

ISO/IEC 15909-2

First edition
2009-11-15

Software and system engineering — High-level Petri nets —

Part 2: Transfer Format

Ingénierie du logiciel et du système — Réseaux de Petri de haut niveau —

Partie 2: Format de transfert

Reference number
ISO/IEC 15909-2:2009(E)

Documents
Document Sharing Hub
© ISO/IEC 2009

Full document details: BS ISO/IEC 15909-2:2009

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2009

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

© ISO/IEC 2009 – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 15909-2:2009

Contents

1 Scope ... 1

2 Conformance ... 1
2.1 PNML Documents ... 1
2.2 PNML Place/Transition Net Documents ... 1
2.3 Textually conformant PNML High-level Petri Net Documents ... 1
2.4 Structurally conformant PNML High-level Petri Net Documents ... 2
2.5 Place/Transition Net Document in High-level Notation ... 2
2.6 Symmetric Net Documents ... 2

3 Normative References ... 2

4 Terms and Definitions ... 3
4.1 Terms ... 3
4.2 Abbreviations ... 5

5 Concepts ... 6
5.1 General Principles ... 6
5.2 PNML Core Model ... 7
5.2.1 Petri Net Documents, Petri Nets, and Objects ... 8
5.2.2 Pages and Reference Nodes ... 8
5.2.3 Labels ... 8
5.2.4 Graphical Information ... 9
5.2.5 Tool Specific Information ... 11
5.2.6 Data Types ... 11
5.3 Petri Net Type Meta Models and their Built-in Sorts ... 12
5.3.1 Place/Transition Nets ... 12
5.3.2 High-Level Core Structure ... 13
5.3.3 Dots ... 16
5.3.4 Multisets ... 16
5.3.5 Booleans ... 18
5.3.6 Finite Enumerations ... 19
5.3.7 Cyclic Enumerations ... 19
5.3.8 Finite Integer Ranges ... 20
5.3.9 Partitions ... 20
5.3.10 Symmetric Nets ... 21
5.3.11 High-Level Petri Net Graphs ... 21
5.3.12 Place/Transition Nets as High-level Net Graphs ... 24

6 Mapping between Part 1 and Part 2 ... 27
6.1 Graphics and Structuring ... 27
6.2 Annotations of HLPNGs ... 27

7 PNML Syntax ... 29
7.1 PNML Documents ... 29
7.1.1 PNML Elements ... 29
7.1.2 Labels ... 29
7.1.3 Graphics ... 30
7.1.4 Mapping of XMLSchemaDataTypes concepts ... 31
7.1.5 Example ... 31
7.2 Mapping Petri Net Type Definitions to XML Syntax ... 33
7.3 Mapping for *High-Level Nets* ... 33
7.3.1 Mapping High-Level Nets meta model elements to PNML syntax ... 33

Annex A (normative) RELAX NG Grammar for the PNML Core Model ... 37

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

Annex B (normative) **RELAX NG Grammars for special types** 49
B.1 Place/Transition Nets 49
B.1.1 The labels 49
B.1.2 Token Graphics 50
B.1.3 The Grammar 51
B.2 High-level Petri Nets 52
B.2.1 Core structure of HLPNGs 52
B.2.2 Dots 61
B.2.3 Multisets 62
B.2.4 Booleans 64
B.2.5 Finite Enumerations 66
B.2.6 Cyclic Enumerations 68
B.2.7 Finite Integer Ranges 69
B.2.8 Partitions 71
B.2.9 Integers 73
B.2.10 Strings 77
B.2.11 Lists 80
B.2.12 Arbitrary Declarations 82
B.2.13 P/T Nets as restricted HLPNGs 84
B.2.14 Symmetric Nets 85
B.2.15 HLPNGs 86

Annex C (informative) **PNML Example of a High-level Net** 88

Annex D (informative) **The PNML Framework: Easing the implementation of PNML** 95
D.1 Introduction 95
D.2 Methodology 95
D.2.1 Overview 95
D.2.2 Generating the API from a model 96
D.3 Features of the PNML Framework 96
D.4 Application to a Petri nets CASE tool 97
D.4.1 Exporting Petri nets models to PNML 98
D.5 Importing Petri nets models from PNML 98
D.6 Time and effort required to develop and integrate a PNML plugin 99
D.7 Conclusion 99
D.7.1 Synthesis 99
D.7.2 The release 100

Bibliography 101

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved
Full document details: BS ISO/IEC 15909-2:2009

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

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

ISO/IEC 15909-2 was prepared by Joint Technical Committee ISO/IEC JTC1, Information Technology , Subcommittee SC 7, Software and Systems Engineering.

ISO/IEC 15909 consists of the following parts, under the general title Software and Systems Engineering – High-level Petri Nets:

— Part 1: Concepts, definitions and graphical notation

— Part 2: Transfer format

— Part 3: Extensions

Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

Introduction

This document is part 2 of ISO/IEC 15909. ISO/IEC 15909 is an International Standard concerned with defining a modelling language and its transfer format, known as *High-level Petri Nets*. Part 1 of ISO/IEC 15909 provides the mathematical definition of *High-level Petri Nets*, called the semantic model, the graphical form of the technique, known as *High-level Petri Net Graphs* (HLPNGs), and its mapping to the semantic model. Part 1 also introduces some common notational conventions for HLPNGs.

This part of ISO/IEC 15909 defines a transfer format for *High-level Petri Nets* in order to support the exchange of *High-level Petri Nets* among different tools. This format is called the *Petri Net Markup Language* (*PNML*). Since there are many different versions of *Petri nets* in addition to *High-level Petri Nets*, this part of ISO/IEC 15909 defines the *core concepts* of *Petri nets* along with an XML syntax, which can be used for exchanging any kind of *Petri net*. Based on this *PNML Core Model*, this part of ISO/IEC 15909 also defines the transfer syntax for the three versions of *Petri nets* that are defined in part 1 of ISO/IEC 15909: *Place/Transition Nets*, *Symmetric Nets*[1], and *High-level Petri Nets*, where *Place/Transition Nets* and *Symmetric Nets* can be considered to be restricted versions of *High-level Petri Nets*. For *Place/Transition Nets*, this part of ISO/IEC 15909 introduces two different transfer formats: one is a format specifically tuned to *Place/Transition Nets*, the other is a format that represents *Place/Transition Nets* as a restricted version of *High-level Petri Nets* as defined in part 1 of ISO/IEC 15909.

The basic level of conformance to this part of ISO/IEC 15909 is to the *PNML Core Model*. The other levels are according to the particular type of the *Petri net*; for *High-level Petri Nets* there are two levels of conformance: *textual conformance* ignores the exact syntax and structure of the *labels*; *structural conformance* requires that *labels* are given in the exact syntax as defined here. Since *Symmetric Nets* are designed for analysability, *textual conformance* does not make any sense for *Symmetric Nets*; therefore, there is only *structural conformance* for *Symmetric Nets*.

Note that this part of ISO/IEC 15909 introduces some concepts that are not defined in part 1 of ISO/IEC 15909. These concepts are not related to the mathematical concepts of *Petri nets* and their semantics. They concern the graphical representation of nets and the structuring of large *Petri net* models. These concepts need to be defined, along with a transfer format for *Petri nets*, in order to ensure that the graphical appearance of a *Petri net* in different tools is similar.

This part of ISO/IEC 15909 is structured as follows: Clause 1 describes the scope, the areas of application and the intended audience of this part of ISO/IEC 15909. Clause 2 defines conformance. Clause 3 gives references that are essential for the correct interpretation of this International Standard. Clause 4 defines all terms relevant to this International Standard and includes a list of abbreviations. Clause 5 introduces the concepts of PNML using UML meta models. Clause 5.2 defines the *PNML Core Model*, which is the structure common to all versions of *Petri nets*. Clause 5.3 defines the particular concepts of the different *Petri net types*. Clause 6 provides the mapping of the syntactical concepts defined in this part of ISO/IEC 15909 to the concepts defined in part 1 of ISO/IEC 15909. Clause 7 defines how the concepts of PNML as defined in Clause 5 are mapped to XML syntax.

Annex A defines the exact XML syntax for the *PNML Core Model* in terms of a RELAX NG grammar. Annex B defines the exact XML syntax for the different types of *Petri nets*. Annex C provides a small example for the syntax of a symmetric net. Annex D discusses a framework for implementing this International Standard and an API for accessing *Petri nets*, which is based on the UML models for the PNML meta models.

[1] Symmetric nets have been first introduced as well-formed nets and are currently under standardization as an Amendment to part 1 of ISO/IEC 15909.

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

Software and system engineering — High-level Petri nets —

Part 2:
Transfer Format

1 Scope

This part of ISO/IEC 15909 defines an XML-based transfer format for *Petri nets*, which are defined conceptually and mathematically in part 1 of ISO/IEC 15909. This transfer format enables the exchange of *Petri nets* among different *Petri net* tools and among different parties. Moreover, this part of ISO/IEC 15909 defines some concepts and XML-based syntax for defining the detailed graphical appearance of *Petri nets*.

The focus of this part of ISO/IEC 15909 is on the transfer format for *Place/Transition Nets*, *High-level Petri Nets* and *Symmetric Nets*. The presentation, however, is structured in such a way that it is open for future extensions, so that other versions of *Petri nets* can be added later. The exact definition of this extension mechanism, called *Petri net type definition*, is not defined in this part of ISO/IEC 15909; it will be defined in part 3 of ISO/IEC 15909.

The transfer format will be used to transfer specifications of systems developed in *High-level Petri Nets* between tools to facilitate the development of systems in teams.

This part of ISO/IEC 15909 is written as a reference for developers of *Petri net* tools. Moreover, it will be useful for researchers who define new versions and variants of *Petri nets*.

2 Conformance

There are different levels of conformance to this part of ISO/IEC 15909. All conformance levels impose additional conditions on valid XML documents.

2.1 PNML Documents

An XML document is conformant to the *PNML Core Model* if it meets the definitions of Clause 5.2 (concepts) and Clause 7.1 (their mapping to XML syntax) – such a document is called a *PNML Document* or a *Petri Net Document*. A Petri net tool is conformant to the *PNML Core Model*, if it can import all *PNML Documents*, and if it can export all *Petri nets* to a *PNML Document*.

The other levels of conformance concern the different *Petri Net Types*.

2.2 PNML Place/Transition Net Documents

A *PNML Document* is a conformant *Place/Transition Net* if it meets the additional restrictions of Clause 5.3.1 (concepts of P/T-nets) and Clause 7.2 (their mapping to XML syntax) – such a document is called a *PNML Place/Transition Net Document*. A Petri net tool is conformant to the *PNML Place/Transition Net* definition if it can import all *PNML Place/Transition Net Documents*, and if it can export all *Place/Transition Nets* to *PNML Place/Transition Net Documents*. Note that this transfer format is tuned to *Place/Transition Nets*. There is another format, which considers *Place/Transition Nets* as a restricted form of *High-level Petri Nets* (see Clause 2.5).

2.3 Textually conformant PNML High-level Petri Net Documents

For *High-level Petri Nets*, there are two different levels of conformance. The first level requires the existence of textual labels as defined in Clause 5.3.2 and 5.3.11 (concepts) and Clause 7.3 (mapping to XML syntax). But,

Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

it does not require the existence of the structural parts of the *annotations*; it only requires that the textual parts of the *annotations* exist, but the text is not required to be in a specific syntax and, therefore, the meaning of it cannot be transferred to other tools. Such a *PNML Document* is called a *textually conformant PNML High-level Petri Net Document*. A Petri net tool is conformant to the *textual PNML High-level Petri Net* definition if it can import all *textually conformant PNML High-level Petri Net Documents*, and if it can export all *High-level Petri Nets* to a *textually conformant PNML High-level Petri Net Document*.

2.4 Structurally conformant PNML High-level Petri Net Documents

The second level of *High-level Petri Net* conformance requires that all *annotations* obey the rules defined in Clause 5.3.2 and 5.3.11 (concepts) and Clause 7.3 (mapping to XML syntax). Such a *PNML Document* is called a *structurally conformant PNML High-level Petri Net Document*. A Petri net tool is conformant to the *structural PNML High-level Petri Net* definition if it can import all *structurally conformant PNML High-level Petri Net Documents*, and if it can export all *High-level Petri Nets* to a *structurally conformant PNML High-level Petri Net Document*.

2.5 Place/Transition Net Document in High-level Notation

A *structurally conformant PNML High-level Petri Net Document* that uses only the single sort *dot* and only the *arc annotations* of *Place/Transition Nets* is a *conformant Place/Transition Net Document in High-level Notation*.

2.6 Symmetric Net Documents

Finally, there is conformance to *Symmetric Nets*, which is a restricted version of *High-level Petri Nets*. A *Symmetric Net Document* is a *structurally conformant PNML High-level Petri Net Document* if it contains only the concepts defined in Clause 5.3.10. A Petri net tool is conformant to the *Symmetric Net* definition if it can import all *Symmetric Net Documents*, and if it can export all *Symmetric Nets* to *Symmetric Net Documents*.

3 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 15444, *Information technology – JPEG 2000 image coding system*

ISO/IEC 15909-1, *Software and system engineering – High-level Petri nets – Part 1: Concepts, definitions and graphical notation*

ISO/IEC 15948, *Information technology – Computer graphics and image processing – Portable Network Graphics (PNG): Functional specification*

ISO/IEC 19757-2:2003, *Information technology – Document Schema Definition Language (DSDL) – Part 2: Regular-grammar-based validation - RELAX NG*

CSS, *Cascading Style Sheets, level 2 revision 1, CSS 2.1 Specification. W3C Candidate Recommendation, 25 February 2004*

OCL 2.0, *Object Constraint Language, OMG Available Specification, Version2.0. OMG formal/06-05-01, May 2006*

UML 2.1, *OMG Unified Modeling Language (OMG UML): Superstructure, V2.1.2 OMG Available Specification, November 2007*

XML 1.1, *Extensible Markup Language (XML) 1.1 (Second Edition) W3C Recommendation, 29 September 2006*

XML Schema Datatypes: *XML Schema Part 2: Datatypes (Second Edition); W3C Recommendation, 28 October 2004*

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved
Full document details: BS ISO/IEC 15909-2:2009

4 Terms and Definitions

For the purposes of this document, the terms and definitions given in part 1 of ISO/IEC 15909 and the following apply.

4.1 Terms

4.1.1
annotation
label (4.1.5) represented as text near to the **object** (4.1.7) it is associated with

4.1.2
attribute
label (4.1.5) that governs the form or shape of the **object** (4.1.7) it is associated with, which, in contrast to an **annotation** (4.1.1), is typically not shown as text

4.1.3
graphical information
information defining the graphical appearance of **objects** (4.1.7) and **labels** (4.1.5) of a **net graph**, which can be the position, size, line colour, fill colour, font or the line width

4.1.4
global label
label (4.1.5) associated with the **net graph** itself, rather than with an **object** (4.1.7) of a **net graph**

4.1.5
label
information associated with the **net graph** or one of its **object** (4.1.7)

4.1.6
meta model
model defining the concepts and their relations for some modelling notation, which, in the context of this document, are mainly UML class diagrams

4.1.7
object (of a net graph)
arc, **node**, **reference node** (4.1.14), or **page** (4.1.8) of a **net graph**

4.1.8
page
structuring mechanism used to split a large **net graph** into smaller parts, which are also the units of the net to be printed

4.1.9
PNML Core Model
meta model (4.1.6) defining the basic concepts and structure of **net graph** models that are common to all versions of **Petri nets**

Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

4.1.10
PNML Document (Petri Net Document)
XML document that contains one or more **net graphs**

4.1.11
PNML High-level Net Document
PNML Document (4.1.10) that contains one or more **net graphs**, where all **net graphs** conform to **High-level Petri Nets**

4.1.12
PNML Place/Transition Net Document
PNML Document (4.1.10) that contains one or more **net graphs**, where all **net graphs** conform to **Place/Transition Nets**

4.1.13
PNML Symmetric Net Document
PNML Document (4.1.10) that contains one or more **net graphs**, where all **net graphs** conform to **Symmetric Nets**

4.1.14
reference node
node of a **Petri net** that is a representative of another **node**, possibly defined on another **page** (4.1.8) of the **net graph**

4.1.15
reference place
reference node (4.1.14) that represents a **place** and refers to either another **reference place** (4.1.15) or to a **place**

4.1.16
reference transition
reference node (4.1.14) that represents a **transition** and refers to either another **reference transition** (4.1.15) or to a **transition**

4.1.17
source node
node associated with the start of an **arc**

4.1.18
target node
node associated with the end of an **arc**

4.1.19
tool specific information
information associated with an **object** (4.1.7) of a **net graph** or with the **net graph** itself that is specific to a particular tool and is not meant to be used by other tools

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved
Full document details: BS ISO/IEC 15909-2:2009

4.2 Abbreviations

4.2.1
CSS
Cascading Stylesheets

4.2.2
CSS2
Cascading Stylesheets level 2

4.2.3
OCL
Object Constraint Language

4.2.4
P/T Net
Place/Transition Net

4.2.5
PNML
Petri Net Markup Language

4.2.6
RELAX NG
Regular Language description for XML/ New Generation

4.2.7
SN
Symmetric Net

4.2.8
UML
Unified Modeling Language

4.2.9
XML
eXtensible Markup Language

Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

5 Concepts

A *Petri net* can be considered to be a labelled graph, where the particular *labels* depend on the particular type of *Petri net*. The concepts for *High-level Petri Net Graphs* are defined in part 1 of ISO/IEC 15909 and are represented in terms of a meta model in UML notation in this clause. The mapping of these concepts to XML syntax is defined in Clause 7.

Some concepts, however, have not been defined in part 1 because the main concern of part 1 was on the semantics of *High-level Petri Nets*. Part 2 of ISO/IEC 15909 defines concepts for defining the graphical appearance of *Petri nets* in order to enable similar graphical representation in different tools. Moreover, part 2 defines the concept of *pages* in order to be able to split the graphical representation of a *Petri net* into smaller units for printing and viewing them.

Note that the concepts and definitions presented in part 1 of ISO/IEC 15909 are semantic in nature, whereas part 2 is syntactic in nature. For example, part 1 refers to *types*, which are sets, but does not mandate any concrete syntax for transferring such sets. Likewise part 1 uses *functions*, but does not mandate a syntax for transferring them. In order to transfer *High-level Petri Nets*, this part of ISO/IEC 15909 needs to introduce a syntax for *types* and *functions*. To this end, it uses the concepts of *sorts* and *operators*. These are syntactic symbols for denoting *types* and *functions*, and are assigned a fixed interpretation, which is defined along with the *sorts* and *operators*. Therefore, this part of ISO/IEC 15909 often refers to *sorts* and *operators*, where part 1 refers to their semantical counterpart, *types* and *functions*. By the fixed interpretation of *sorts* and *operators*, the respective *types* and *functions* are also defined.

5.1 General Principles

This part of ISO/IEC 15909 defines a transfer format for *Place/Transition Nets*, *High-level Petri Net Graphs*, and *Symmetric Nets* as defined in part 1 of ISO/IEC 15909. This transfer format has been designed to be extensible and to be open for future variants of *Petri nets* and possibly for other use, such as the transfer of results associated with the analysis of Petri nets, e. g., reachability graphs. In order to obtain this flexibility, the transfer format considers a *Petri net* as a labelled directed graph, where all type specific information of the net is represented in *labels*. A *label* may be associated with a *node*, an *arc*, a *page* or the *net* itself. This basic structure is captured in the PNML *Core Model*, which is defined in Clause 5.2.

The PNML *Core Model* is presented using UML class diagrams. Note that these UML diagrams do not define the concrete XML syntax for PNML *Documents*; rather, they define the abstract syntax only. The mapping of the PNML *Core Model* elements (i. e., the abstract syntax) to the concrete XML syntax will be defined in Clause 7; Annex A defines the exact XML syntax in terms of a RELAX NG grammar.

The PNML *Core Model* imposes no restrictions on *labels*. Therefore, the PNML *Core Model* can represent any kind of *Petri net*. Due to this generality of the PNML *Core Model*, there can be PNML *Documents* that do not correspond to a *Petri net* at all. For example, there could be *labels* from two different and even incompatible versions of *Petri nets* within the same PNML *Document*. For a concrete version of *Petri nets*, the legal *labels* are defined by extending the PNML *Core Model* with another meta model that exactly defines the legal *labels* of this type.

Technically, the PNML *Core Model* is a UML package, and there are additional UML packages for the different *Petri net types* that extend the PNML *Core Model* package. This part of ISO/IEC 15909 defines a package for *Place/Transition Nets*, a package for *Symmetric Nets*, and a package for *High-level Petri Net Graphs*, where the package for *High-level Petri Net Graphs* extends the package for *Symmetric Nets*. Therefore, every *Symmetric Net* is also a *High-level Petri Net Graph*. The representation of these concepts in XML syntax is defined in Clause 7.2; Annex B defines the exact XML syntax of the concepts for the different types in terms of RELAX NG grammars.

Figure 1 shows the different packages and how they are related. The package PNML *Core Model* defines the basic structure of *Petri nets* or *net graphs*; this structure is extended by the package for each type. The PNML *Core Model* is defined in Clause 5.2, the package PT-Net is defined in Clause 5.3.1, the package SymmetricNet is defined in Clause 5.3.10 and the package HLPNG is defined in Clause 5.3.11. Clause 7 shows how the concepts defined in these packages are represented in concrete XML syntax.

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved
Full document details: BS ISO/IEC 15909-2:2009

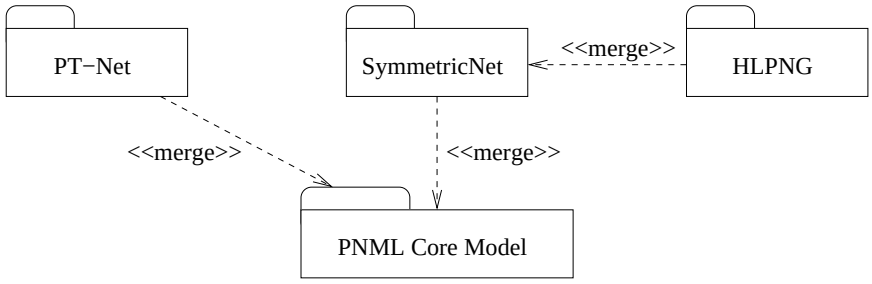

Figure 1: Overview of the UML packages of PNML

5.2 PNML Core Model

Figures 2 and 3 show the *PNML Core Model* as a UML class diagram. The diagram of Fig. 2 focuses on the conceptual parts, whereas the diagram of Fig. 3 focuses on the parts concerning the graphical representation. Note that the data type *String* is imported from a separate package *XMLSchemaDataTypes*, which is discussed in Clause 5.2.6.

The concepts defined in the *PNML Core Model* are discussed in the following clauses.

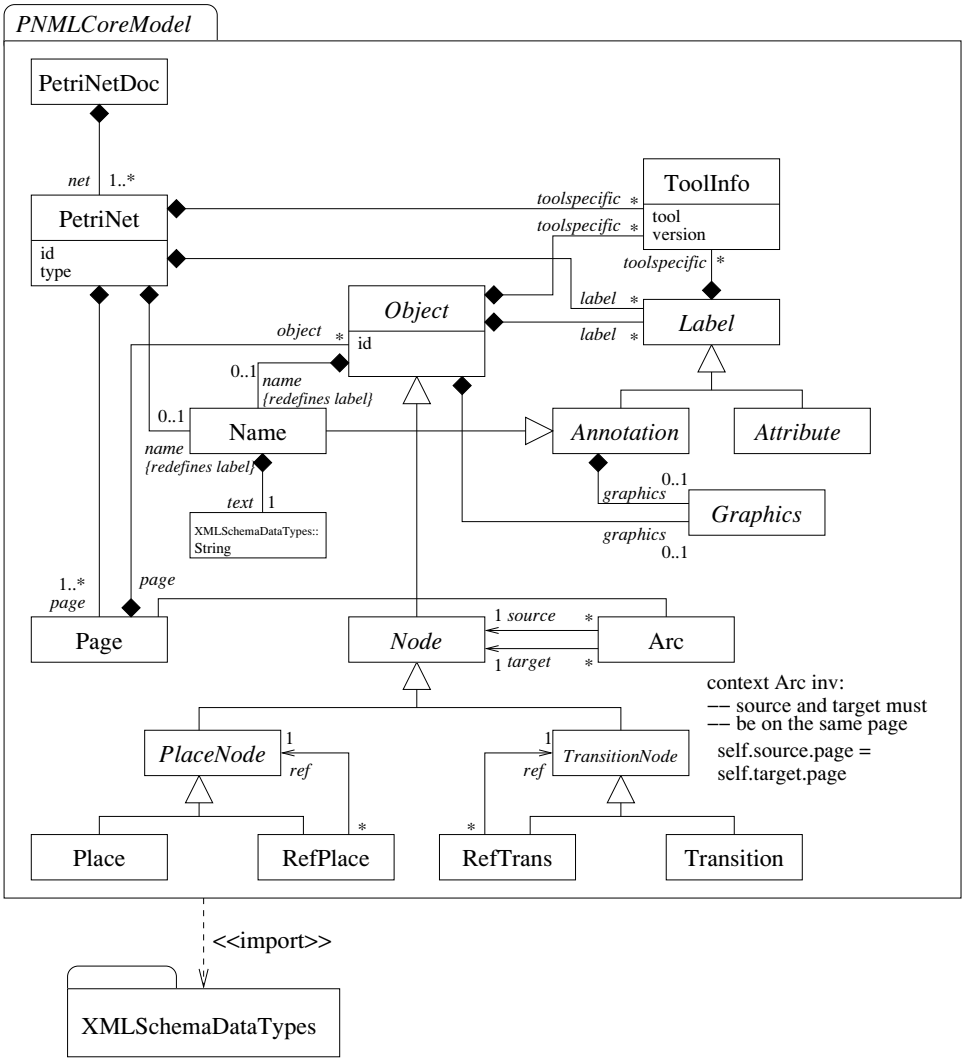

Figure 2: The *PNML Core Model* package: concepts

Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

5.2.1 Petri Net Documents, Petri Nets, and Objects

A document that meets the requirements of the *PNML* Core Model is called a *Petri Net Document* (PetriNetDoc) or a *PNML Document*. It contains one or more *Petri Nets* (PetriNet). Each *Petri Net* has a unique identifier and a *type*. The *type* is a name referring to the package with its definition; an example for such a package name is `http://www.pnml.org/version-2009/grammar/ptnet` for the package defining *P/T Nets*. The URL for the definition of PNML itself is `http://www.pnml.org/version-2009/grammar/pnml`.

A *Petri net* consists of one or more top-level *pages* that in turn consist of several *objects*. These *objects*, basically, represent the graph structure of the *Petri net*. Each *object* within a *Petri net document* has a unique *identifier*, which can be used for referring to this *object*. Moreover, each *object* may be equipped with graphical information defining its position, size, colour, shape and other attributes on its graphical appearance. The precise graphical information that can be provided for an *object* depends on the particular type of *object* (see Clause 5.2.4 and Fig. 3).

The main *objects* of a *Petri net* are *places*, *transitions* and *arcs*. For convenience, a *place* or a *transition* is generalized to a *node*. For reasons explained in Clause 5.2.2 below, this generalization is via *place nodes* and via *transition nodes*. *Nodes* of a *Petri net* can be connected by *arcs*.

Note that it is legal to have an *arc* from a *place* to a *place* or from a *transition* to a *transition* according to the *PNML Core Model*. The reason is that there are versions of *Petri nets* that support such *arcs*. If a *Petri net type* does not support such *arcs*, this restriction will be defined in the particular package defining this type.

5.2.2 Pages and Reference Nodes

Three other kinds of *objects* are used for structuring a *Petri net*: *pages*, *reference places*, and *reference transitions*. As mentioned above, a *page* may contain other *objects*; since a *page* is an *object* itself, a *page* may even contain other *pages*, which defines a hierarchy of subpages.

This part of ISO/IEC 15909 requires that an *arc* must connect *nodes* on the same *page* only. The reason for this requirement is that *arcs* connecting *nodes* on different *pages* cannot be drawn graphically on a single *page*. In the *PNML Core Model* of Fig. 2, this requirement is captured by the OCL expression (OCL 2.0, OMG) next to the class for *arcs*.

In order to connect *nodes* on different *pages* by an *arc*, a representative of one of the two *nodes* is drawn on the same *page* as the other *node*. Then, this representative may be connected with the other *node* by an *arc*. This representative is called a *reference node*, because it has a reference to the *node* it represents. Note that a *reference place* must refer to a *place* or a *reference place*, and a *reference transition* must refer to a *reference transition* or a *transition*. Moreover, cyclic references among *reference nodes* are not allowed. Though this requirement cannot be expressed in UML or OCL notation, this requirement is mandatory.

The concepts of *pages* and *reference nodes* are not defined in part 1 of ISO/IEC 15909. The reason for introducing them in part 2 is that these concepts are needed for graphical and structuring purposes. Semantically, they do not carry any meaning. The meaning of these concepts is defined by merging each *reference node* with the *node* it ultimately refers to and by simply ignoring the *pages*. This procedure is called *flattening* of the page structure.

5.2.3 Labels

In order to assign further meaning to an *object*, each *object* may have *labels*. Typically, a *label* represents the name of a *node*, the initial marking of a *place*, the *transition condition*, or some *arc annotation*. In addition, the *Petri net* itself or its *pages* may have some *labels*, which are called *global labels*. For example, the package HLPNG defines *declarations* as *global labels* of a *High-level Petri Net*, which are used for defining *variables*, and user-defined *sorts* and *operators*.

This part of ISO/IEC 15909 distinguishes two kinds of *labels*: *annotations* and *attributes*. An *annotation* comprises information that is typically displayed as text next to the corresponding *object*. Examples of *annotations* are names, initial markings, arc annotations, transition conditions, and timing or stochastic information.

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved
Full document details: BS ISO/IEC 15909-2:2009

In contrast, an *attribute* is, typically, not displayed as text next to the corresponding *object*. Rather, an *attribute* has an effect on the shape or colour of the corresponding *object*. For example, an *attribute* such as arc type could have domain { `normal`, `read`[2], `inhibitor`, `reset` }. This part of ISO/IEC 15909, however, does not mandate the effect that an *attribute* has on the graphical appearance of a net *object*.

Note that the classes for *label*, *annotation* and *attribute* are abstract in the PNML *Core Model*, which means that the PNML *Core Model* does not define concrete *labels*, *annotations*, and *attributes*. The only concrete *label* defined in the PNML *Core Model* is the *name*, which is a *label* that can be used for any *object* within any *Petri net type*. Note that, this way, any *object* – including *nodes*, *pages*, even the *net* itself as well as *arcs* – can have a *name*. The value of a *name* is a *String*, which is imported from the separate package *XMLSchemaDataTypes* (see Clause 5.2.6 for more information). The other concrete *labels* will be defined in the packages for the concrete *Petri net types* (see Clause 5.3).

In order to support the exchange of information among tools that have different textual representation for the same concepts (i. e. if they have different concrete syntax), there are two ways for representing the information within an annotation: *textually* in some concrete syntax and *structurally* as an abstract syntax tree (see Clauses 5.3.11 and 7.1.2 for details).

Note that *reference nodes* may have *labels*, but these *labels* do not have any effect on the semantics of the net and can be completely ignored semantically. Since the labels of a *reference node* do not have an effect on the semantics, it is easy to obtain a semantically equivalent *Petri net* without *pages* by merging every *reference node* to the *node* it directly or indirectly refers to. This is called *flattening* of the *Petri net*. But, the labels of a *reference node* can have an effect on the graphical appearance or can give some additional information to the user.

5.2.4 Graphical Information

In addition to the *Petri net* concepts, information concerning the graphical appearance can be associated with each *object* and each *annotation*. For a *node*, this information includes its position; for an *arc*, it includes a list of positions that define intermediate points of the *arc*; for an *object's annotation*, it includes its relative position with respect to the corresponding *object*; and for an *annotation* of a *page*, the position is absolute. There can be further information concerning the size, colour and shape of nodes or arcs, or concerning the colour, font and font size of labels. Note that this information can be used for automatically transforming a *Petri Net* into *Scalable Vector Graphics* (SVG) by XSLT transformations (see [8] for more details). This transformation, however, is not defined in this part of ISO/IEC 15909.

Figure 3 shows the different graphical information that can be attached to the different types of *objects* and the different attributes. Note that this still belongs to the *PNML Core Model*; it is shown in a different figure only for better understandability. Table 1 gives an overview of the meaning and the domain of the attributes of the different graphical features. The exact XML syntax is defined in Clause 7.1.3.

A *position* element defines the absolute position for *nodes* and *pages*. For an *annotation* an *offset* element defines its relative position to the *object* it is attached to – if it is a *global annotation*, the *offset* also defines the absolute position on that *page*. Each absolute or relative position consists of a pair of Cartesian coordinates (x, y), where the units are points (pt). As for many graphical tools, the x-axis runs from left to right and the y-axis from top to bottom. And the reference point for the position of an *object* is its centre.

For an *arc*, the (possibly empty) sequence of *positions* defines its intermediate points (bend points). Note that the positions of the *start point* and the *end point* of an *arc* are not given explicitly for the arc. These positions are determined from the position of the source and the target *node* of the *arc* and the direction of the respective segment of the *arc*. They are defined by the line starting or ending at the reference point of the respective node in the direction of the arc and the intersection with the border of that node. Altogether, the *arc* is displayed as a path from the *start point* on the border of the *source node* to the *end point* on the border of the *target node* via the intermediate points. Depending on the value of the attribute *shape* of element *line*, the path is displayed as a broken *line* (polyline) or as a quadratic Bezier *curve*. In the case of a Bezier curve the intermediate positions alternately are the line connectors or Bezier control points. The reference point of an arc is the middle of the arc, which is the middle of the middle segment of the arc, if there are an odd number of segments; it is the middle point, if there are an even number of segments.

[2] Sometimes, a read arc is called a test arc.

Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

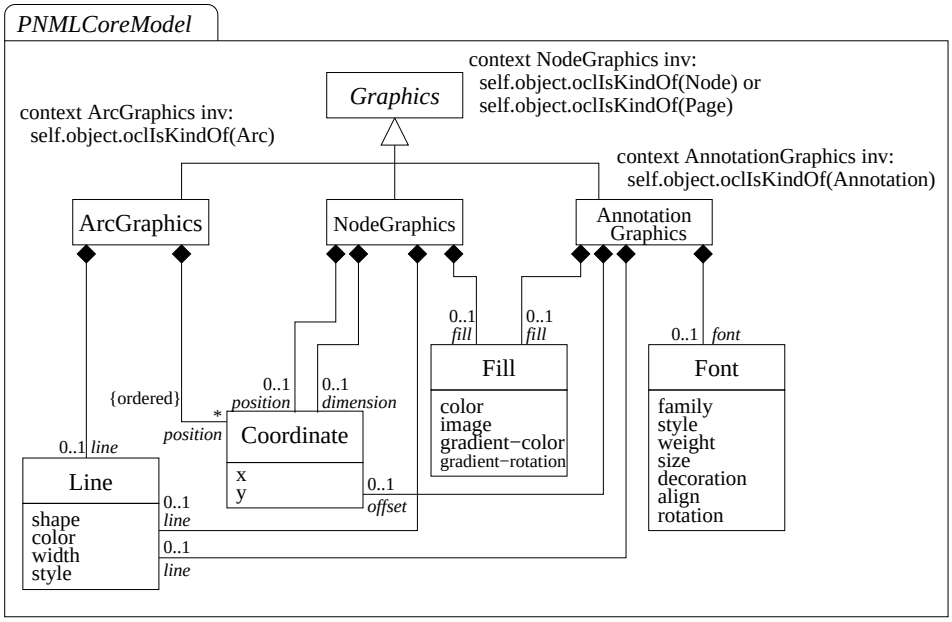

Figure 3: The *PNML Core Model*: graphical information

Table 1: PNML attributes of graphical information

Class	Attribute	Domain
Coordinate	x	decimal
	y	decimal
Fill	color	CSS2-color
	image	anyURI
	gradient-color	CSS2-color
	gradient-rotation	{vertical, horizontal, diagonal}
Line	shape	{line, curve}
	color	CSS2-color
	width	nonNegativeDecimal
	style	{solid, dash, dot}
Font	family	CSS2-font-family
	style	CSS2-font-style
	weight	CSS2-font-weight
	size	CSS2-font-size
	decoration	{underline, overline, line-through}
	align	{left, center, right}
	rotation	decimal

Although there is no way to define the *start point* and the *end point* of an arc explicitly, the above concepts allow a user to make sure that an arc starts and ends exactly where it should. If the first, (or last) intermediate point of an arc lies exactly on the border of the respective node (or very close to it), this will be the *start point* (or the *end point* respectively) of that arc.

For an *arc*, there are some *line* attributes, which define the *style*, the *colour*, and the *width* in which it is displayed.

The *dimension* of a *node* or *page* gives its size, again as a pair referring to its width (x) and height (y). Depending on the ratio of height and width, a *place* is displayed as an ellipse rather than a circle. A *transition* is displayed as a rectangle of the corresponding size. If the dimension of an element is missing, each tool is free to use its own default value for the dimensions.

The two elements *fill* and *line* define the interior and outline colours of the corresponding element. The value assigned to a *color* attribute must be a RGB value or a predefined colour as defined by CSS2. When the attribute

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved
Full document details: BS ISO/IEC 15909-2:2009

gradient-color is defined, the fill colour continuously varies from color to gradient-color. The additional attribute *gradient-rotation* defines the orientation of the gradient. When the attribute *image* is defined, the node is displayed as the image which is provided at the specified URI, which must be a graphics file in JPEG (ISO/IEC 15444) or PNG (ISO/IEC 15948) format. In this case, all other attributes of *fill* and *line* are ignored.

For an *annotation*, the *font* element defines the font used to display the text of the *label*. The attributes *family*, *style*, *weight*, and *size* are CSS2 attributes for defining the appearance of the text. The detailed description of the possible values of these attributes and their effect can be found in the CSS2 specification (CSS 2.1 Specification). The *decoration* attribute defines additional properties of the appearance of the text such underlining, overlining, or striking-through the text. Additionally, the *align* attribute defines the alignment of the text to the *left*, *right* or *center*. The *rotation* attribute defines a clockwise rotation of the text.

The reference point of an annotation is always its centre.

5.2.5 Tool Specific Information

For some tools, it might be necessary to store *tool specific information* (ToolInfo), which is not meant to be used by other tools. In order to store this information, *tool specific information* may be associated with each *object* and each *label*. The internal structure of the *tool specific information* depends on the tool and is not specified by PNML. PNML provides a mechanism for clearly marking *tool specific information along* with the name and the version of the tool adding this information. Therefore, other tools can easily ignore it, and adding *tool specific information* will never compromise a *Petri Net Document*.

The same *object* may be tagged with *tool specific information* from different tools. This way, the same document can be used and changed by different tools at the same time. The intention is that a tool should never change or delete the information added by another tool as long as the corresponding *object* is not deleted. Moreover, *tool specific information* should be self-contained and not refer to other *objects* of the net because the deletion of other *objects* by a different tool might make this reference invalid and leave the *tool specific information* inconsistent. This use of the *tool specific information* is strongly recommended; however, it is not normative!

5.2.6 Data Types

This Clause defines six data types, which can be used when defining labels for some *Petri net type* with numerical and textual information. They correspond to different data types in XMLSchema. Figure 4 gives an overview on these data types and their relation.

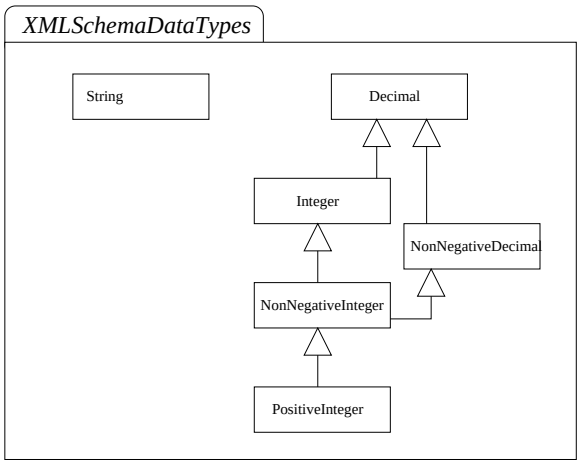

Figure 4: Standard data types

String refers to any printable sequence of characters, which will be mapped to XML PCDATA. *Decimals*, *NonNegativeDecimals*, *Integer*, *NonNegativeInteger*, and *PositiveInteger* refer to any character sequence which denotes a number of the corresponding kind.

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

5.3 Petri Net Type Meta Models and their Built-in Sorts

This Clause defines meta models for three versions of *Petri nets*: *Place/Transition Net*, *Symmetric Nets*, and *High-Level Petri Net Graphs* (HLPNGs) as defined in part 1 of ISO/IEC 15909, where *Symmetric Nets* are a restricted version of *High-Level Petri Net Graphs* currently defined as an Amendment to part 1. This general structure has already been shown in Fig. 1. These meta models define the labels of the respective *Petri net type*.

Annex B.1 of part 1 of ISO/IEC 15909 defines a mapping from the concepts of *Place/Transition Nets* to the concepts of *High-Level Petri Net Graphs* and shows how the usual mathematical representation of *Place/Transition Nets* can be represented as a restricted version of *High-level Petri Nets*. This will be *Place/Transiton Nets in High-level Notation*, which will be defined in Clause 5.3.12. This part of ISO/IEC 15909 introduces an explicit transfer format for *Place/Transition Nets* in order not to force tools for *Place/Transition Nets* to use the syntax of *High-Level Petri Net Graphs*. This format reflects the usual mathematical definition of *Place/Transition Nets*.

5.3.1 Place/Transition Nets

This Clause defines the meta model for *Place/Transition Nets* in terms of a UML package *PT-Net*.

A *Place/Transition Net* is a *net graph*, where each *place* can be labelled with a natural number representing the *initial marking* and an *arc* can be labelled with a non-zero natural number representing the *arc annotation*, with the meaning as defined in part 1 of ISO/IEC 15909 (see Clause B.1): the *label* of a *place* p denotes the initial marking $M_0(p)$, the *label* of an *arc* f denotes the arc weight $W(f)$.

Figure 5 shows the package PT-Net. Note that the only classes defined here are *PTMarking* and *PTArcAnnotation*. The classes *Place*, *Arc*, and *Annotation* come from the package *PNML Core Model*. They are imported (actually, they are merged) here in order to define the possible *labels* for the particular *nodes* of *Place/Transition Nets*. Likewise, the classes prefixed with *XMLSchemaDataTypes* are imported from the standard data type package (see Clause 5.2.6).

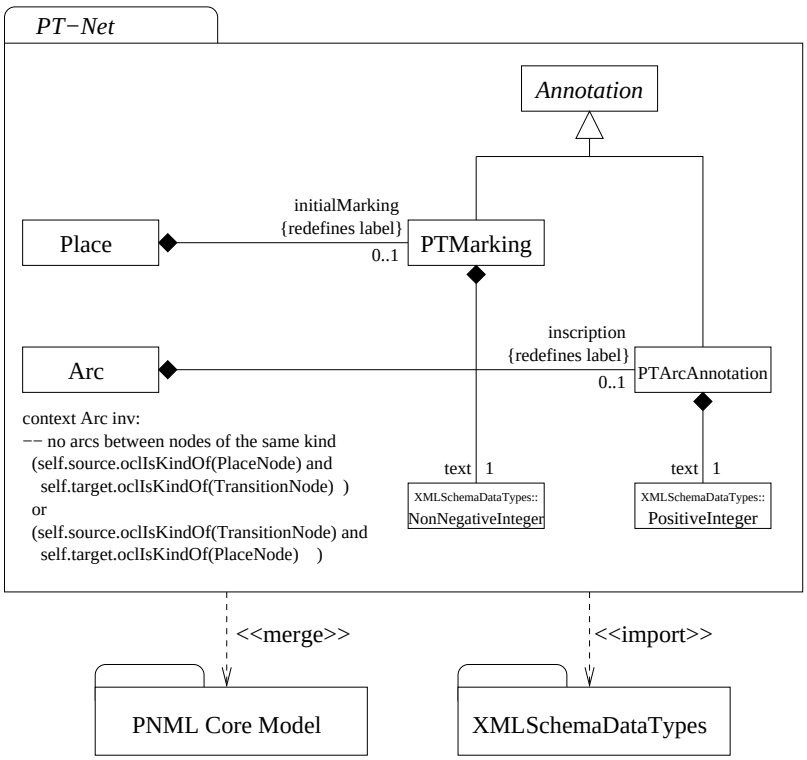

Figure 5: The package PT-Net

Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved
Full document details: BS ISO/IEC 15909-2:2009

The *initial marking* of a *place* is represented by the *annotation PTMarking*, the contents of which must be a non-negative integer. Technically, the representation of the contents of this *label* is defined by referring to the data type *NonNegativeInteger*.

The *arc annotation* is represented by the *annotation PTArcAnnotation*, the contents of which must be a non-zero natural number, which is defined by referring to the data type *PositiveInteger*.

Note that, according to this definition, it is legal that a *place* does not have an *annotation* for the *initial marking*. In that case, the *initial marking* is assumed to be empty, i. e. 0. Likewise, there may be no *annotation* for an *arc*. In that case, the *arc annotation* is assumed to be 1.

In addition to the definition of the new *labels*, this package also defines the structural restriction that, in a *Place/-Transition Net*, an *arc* must not connect a *place* to a *place* or a *transition* to a *transition*. This is captured by the OCL expression below class *arc*.

Note that the UML meta model of this package does not define the XML syntax for the representation of these labels. The mapping to XML syntax is defined in Clauses 7.1.4 (data types) and 7.2 (labels for *P/T nets*).

Sometimes, one wants to store the position of the individual tokens within a place. In order not to mandate all tools to support this feature, this part of ISO/IEC 15909 suggests using tool specific information for this purpose, which would refer to the tool *org.pnml.tool*. Since no tool must support tool specific features, every tool is free to use and support this feature or not.

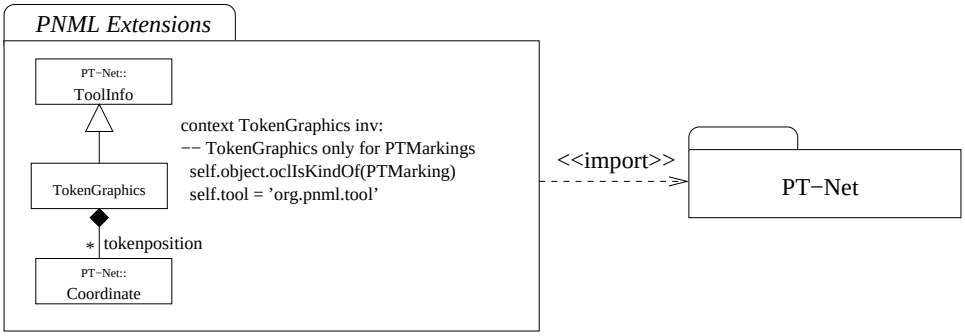

Figure 6: Tool specific extension for token positions

This extension is shown in Fig. 6. For a *PTMarking*, this label can be equipped with the *tool specific* information attached to *AnnotationGraphics*. This consists of information on a list of *tokenpositions*. Each of these elements represents the position of a token relative to the centre of the *place*; represented as a *Coordinate*, which is imported from package *PT-Net*, where it was obtained by a merge with the PNML *Core Model*. If this graphical information is present at all, the number of *tokenpositions* should be the same as indicated by the *PTMarking*.

5.3.2 High-Level Core Structure

Part 1 of ISO/IEC 15909 defines *High-level Petri Net Graphs*. *Symmetric Nets* and *Place/Transition Nets* are introduced as restricted versions of *High-level Petri Net Graphs*. The difference is in the *types* and *functions* that may be used in the restricted versions.

This Clause defines the general structure of *Symmetric Nets*, *High-Level Petri Net Graphs*, and *Place/Transition Nets in High-level Notation*. This is called the *High-Level Core Structure*. In Clauses 5.3.10, 5.3.11, and 5.3.12, this *High-Level Core Structure* will be used for defining the three Petri net types.

Syntactically, the basic features of a *High-Level Petri Net* are the *annotations* of *places*, *transitions*, and *arcs*. For each *place*, the *sort* defines the *type* of the tokens on this *place*. Note that, in contrast to part 1, the place is indirectly associated with a *type* via the *sort* and its fixed interpretation. The *term* associated with a *place* denotes the *initial marking* and must have the respective *sort*. The *term* associated with an *arc* to or from a *place*, defines which tokens are added or removed, when the corresponding *transition* fires. These *terms* also must be of the respective *sort*. Note that part 1 is a bit more general since it allows, any expression that always evaluate to elements of the correct type, which is a semantical condition. Here, we use *terms*, which enforce the correct typing syntactically.

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

For constructing such *terms*, one can use built-in *operators* and *sorts*, and user-defined *variables*, which are defined in a *variable declaration*. The *variable declarations* are *annotations* of the *net* or a *page*. Moreover, a *transition* can have a *condition*, which is a *term* of *sort* Bool and imposes additional conditions on the situations in which a transition can fire. These concepts and their semantics are precisely defined in part 1 of ISO/IEC 15909. The only difference is that, in part 2, there is a fixed relation between *sorts* and *types* and between *operators* and *functions*.

The UML diagram for the package *Terms*, which defines all these concepts, is shown in Fig. 7: It defines the concepts of *sorts*, *operators*, *declarations*, and *terms*, and how *terms* are constructed from *variables* and *operators*.

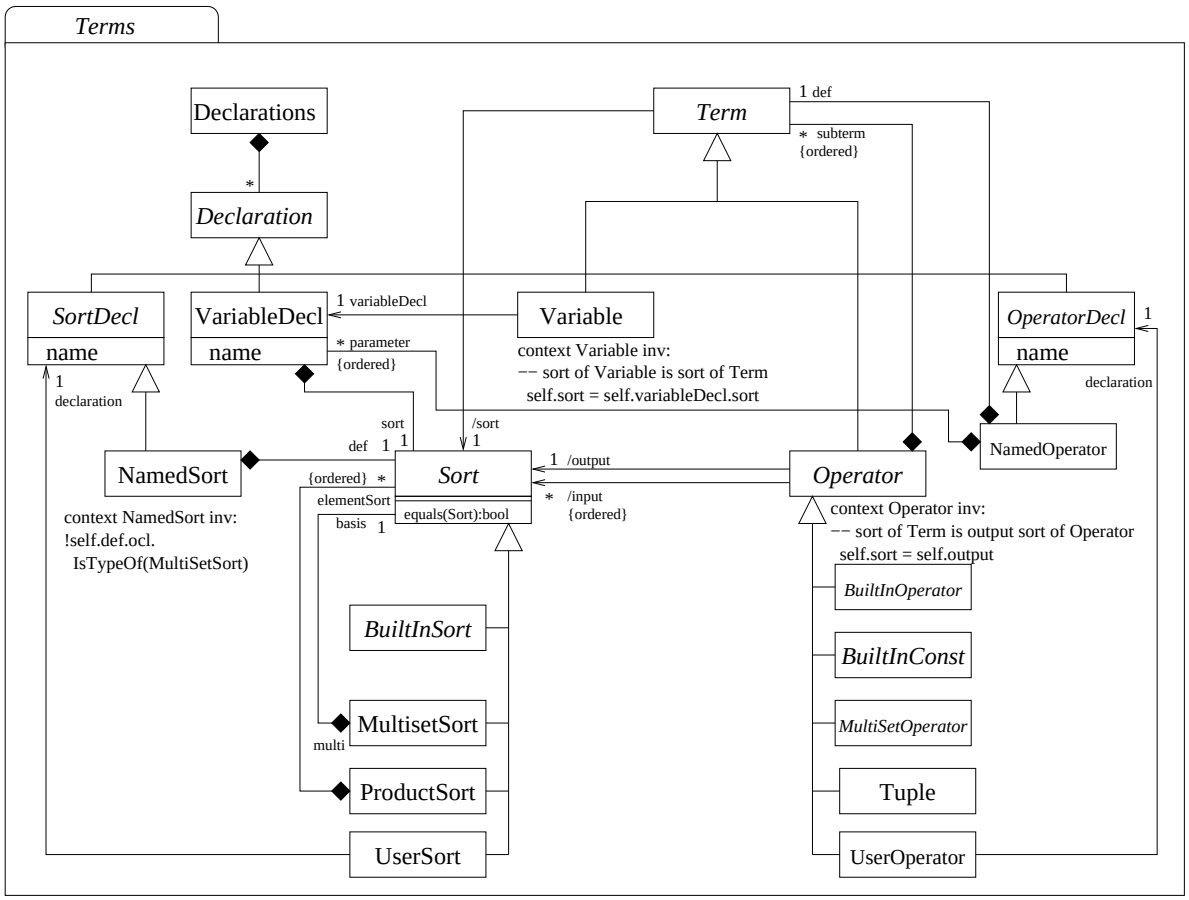

Figure 7: The meta model for *Terms*

For each *variable declaration* there is a corresponding *sort*. A *sort* can be a *built-in sort*, a *multiset sort* over some *basis sort*, a *product sort* over some *sorts*, or a *sort* which is given in a *user declaration*. In the core structure, the only possible *sort declaration* of a user is by constructing a new *sort* from existing ones and by giving them a new name. From these, the user can define new ones. Note that cyclic references in user defined *sorts* are not allowed. In addition to these user defined sorts, called *named sorts*, there are *arbitrary sorts*. Since these are not allowed in *Symmetric Nets* these are defined only in Clause 5.3.11.

An *operator* can be a *built-in constant* or *built-in operator*, a *multiset operator* which among others can construct a *multiset* from an enumeration of its elements, or a *tuple* operator. Each *operator* has a sequence of *sorts* as its *input sorts*, and exactly one *output sort*, which defines its signature. As for *sorts*, the user can define his own *operators*. Here, it is only possible to define an abbreviation, which is called a *named operator*: It can use a *term*, which is built from existing *operators* and variables, for defining a new *operator*. As for *sorts*, cycles (recursion) in these definitions are not allowed. As for sorts, there will be arbitrary operator declarations for *High-level Petri Net Graphs*, but not for *Symmetric Nets*. Therefore, these concepts will be defined in Clause 5.3.11.

From the built-in *operators* and the user-defined *operators* and *variables*, *terms* can be constructed in the usual way. The *sort* of a *term* is the *sort* of the *variable* or the *output sort* of the *operator*. Therefore, *sort* is indicated as a derived association; its definition is expressed by an OCL expression in the UML meta model. Note that the *input* and

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved
Full document details: BS ISO/IEC 15909-2:2009

output sort of the *operator* are also represented as derived associations because, in some situations, they need not be given explicitly since they can be derived from the type of the *operator*.

Figure 8 shows the package *High-Level Core Structure*, which defines all the *annotations* for both *Symmetric Nets* and *High-Level Petri Net Graphs*. Note that, since the classes for built-in *sorts* and *operators* are abstract, we do not have any built-in *sorts* and *operators* yet. These will be defined in Clauses 5.3.3 to 5.3.9.

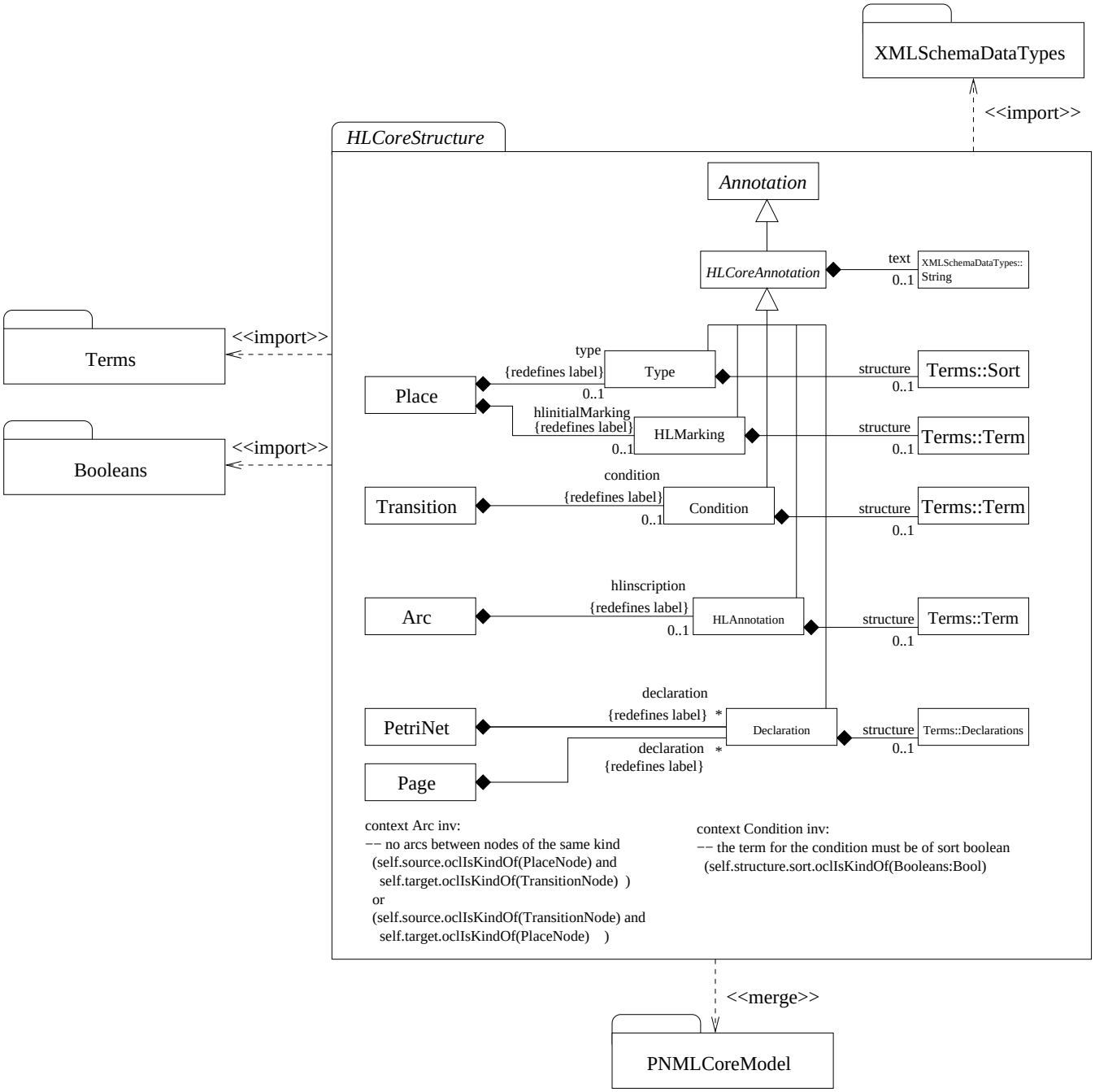

Figure 8: The package *High-Level Core Structure*

In addition to the *annotations* defined above, the package *High-Level Core Structure* requires that *arcs* must not connect two *places* and must not connect two *transitions*.

Note that this model defines an abstract syntax (*structure*) for all these concepts only. In order to allow tools to store some concrete text, all *annotations* of *High-level Nets* may also consist of text, which should be the same

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

expression in some concrete syntax of some tool. The concrete syntax, however, is not defined in this part of ISO/IEC 15909. This common structure of all *annotations* is captured by *HLCoreAnnotation* in the package *High-Level Core Structure*.

The concepts defined in Fig. 8 correspond to the concepts defined in part 1 of ISO/IEC 15909 in the following way. The *label Type* of a *place* defines the *type* by referring to some *sort*; by the fixed interpretation of built-in *sorts*, this *sort* defines the *type* of the *place*. The *label HLMarking* of a *place* is a *term* with some *multiset sort* denoting a collection of tokens on the corresponding *place*, which defines its initial marking. The *label Condition* is a *term* of *sort Bool*, which is the *transition condition*. The *label HLAnnotation* refers to a *term*, which is the expression defining the *arc annotation*.

Declaration labels correspond to the *declarations*, which can be *variable declarations* and declarations of *sorts* and *operators*. In the *High-level Core Structure* the user can define abbreviations only; later, in *High-Level Petri Net Graphs*, there will be arbitrary declarations.

As discussed above, this Clause defines the core structure of *High-level Petri Net Graphs*. The following Clauses 5.3.3 to 5.3.9 define different built-in *sorts* and *operators* along with their interpretations as *types* and *functions* respectively. For each built-in *sort*, there will be a UML package.

5.3.3 Dots

To enable the use of *Place/Transition Net* annotations as high-level annotations, the concept of Dots has been defined. *Place/Transition Nets* as High-level Net Graphs are defined in Clause 5.3.12.

Dots represent the *common understanding* of *non coloured annotation* in a High-level net. That is to say, dots are the high-level representation of the natural numbers labels used in Place/Transition Nets. The difference is that they are typed as a built-in sort to be compatible with the corresponding high-level annotation defined in part 1 of ISO/IEC 15909. Dots are depicted in Fig. 9.

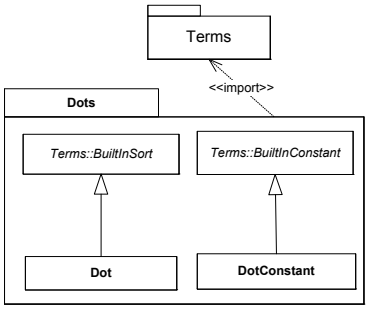

Figure 9: The package for Dots

They consist of a *sort* named Dot. The *type* associated with this *sort* is the set with a single element: $\{\bullet\}$. The constant $\mathrm{DotConstant}$ is associated with the 0-ary function that returns $\bullet$, i. e. the constant denotes that single element.

This *sort* can be used for representing *non coloured annotations* in *High-level Petri Nets*.

5.3.4 Multisets

Figure 10 shows the UML package for *multisets* and their *operators*. For every *sort*, the *multiset sort* over this *basis sort* is interpreted as the set of multisets over the type associated with the *basis sort*.

Below we define the *operators* on *multiset sorts* along with their interpretation:

- $\mathrm{Add} : \mathrm{Multiset} \times \mathrm{Multiset} \times \ldots \times \mathrm{Multiset} \rightarrow \mathrm{Multiset}$ is interpreted as the multiset addition. For example, $\mathrm{Add}([a, b], [b, c])$ evaluates to $[a, 2b, c]$.

AVA Documents
Document Sharing Hub

 © ISO/IEC 2009 – All rights reserved
Full document details: BS ISO/IEC 15909-2:2009

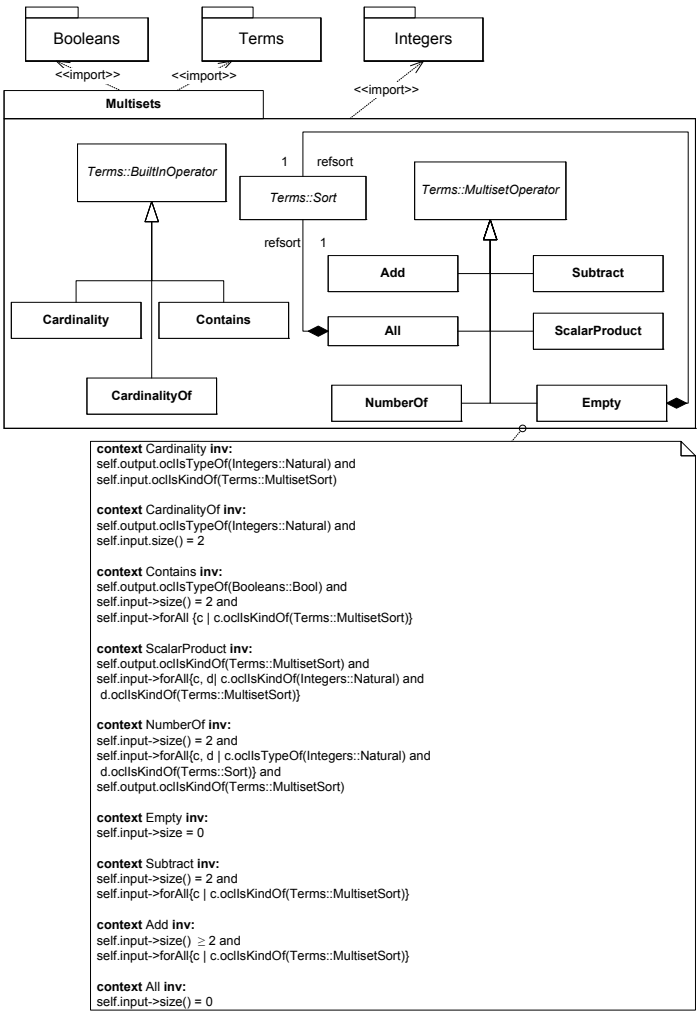

Figure 10: The package *multiset* and its built-in and *operators*

- $\mathrm{All} :\rightarrow \mathrm{Multiset}$ is a *constant* that, for a given finite *sort*, is interpreted as the multiset that contains every element of the type associated with that sort exactly once. For example, for a *sort* s with associated type $\{a, b, c\}$ the *constant* All_s denotes the multiset $[a, b, c]$. Note that the *sort* which is indicated by index s in this example is represented by the composition *refsort* in the meta model since the *output sort* cannot be derived from the *operator* or its *subterms*.

- $\mathrm{Empty} :\rightarrow \mathrm{Multiset}$ is a *constant* which constructs the empty multiset over the associated type of the *sort*, i. e. Empty_s denotes $[]$. Note that the *sort* which is indicated by the index s in this example is represented by the composition *refsort* in the meta model since the *output sort* cannot be derived from the *operator* or its *subterms*.

- $\mathrm{Subtract} : \mathrm{Multiset} \times \mathrm{Multiset} \rightarrow \mathrm{Multiset}$ is interpreted as the multiset subtraction. For example, the term $\mathrm{Subtract}([a, 2b, c], [b, c])$ evaluates to $[a, b]$.

- $\mathrm{ScalarProduct} : \mathbb{N} \times \mathrm{Multiset} \rightarrow \mathrm{Multiset}$ is interpreted as the product of a multiset with a scalar. For example, $\mathrm{ScalarProduct}(2, [a, c])$ evaluates to $[2a, 2c]$.

- $\mathrm{NumberOf} : \mathbb{N} \times \mathrm{Sort} \rightarrow \mathrm{Multiset}$ is interpreted as a function that produces a multiset in which the element occurs exactly in the given number and no other elements in it. For example, $\mathrm{NumberOf}(3, a)$ evaluates to $[3a]$.

- $\mathrm{Cardinality} : \mathrm{Multiset} \rightarrow \mathbb{N}$ is interpreted as the cardinality function, i. e. it returns the number of elements of a multiset. For example, $\mathrm{Cardinality}([a, 2b, c])$ evaluates to 4.

Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

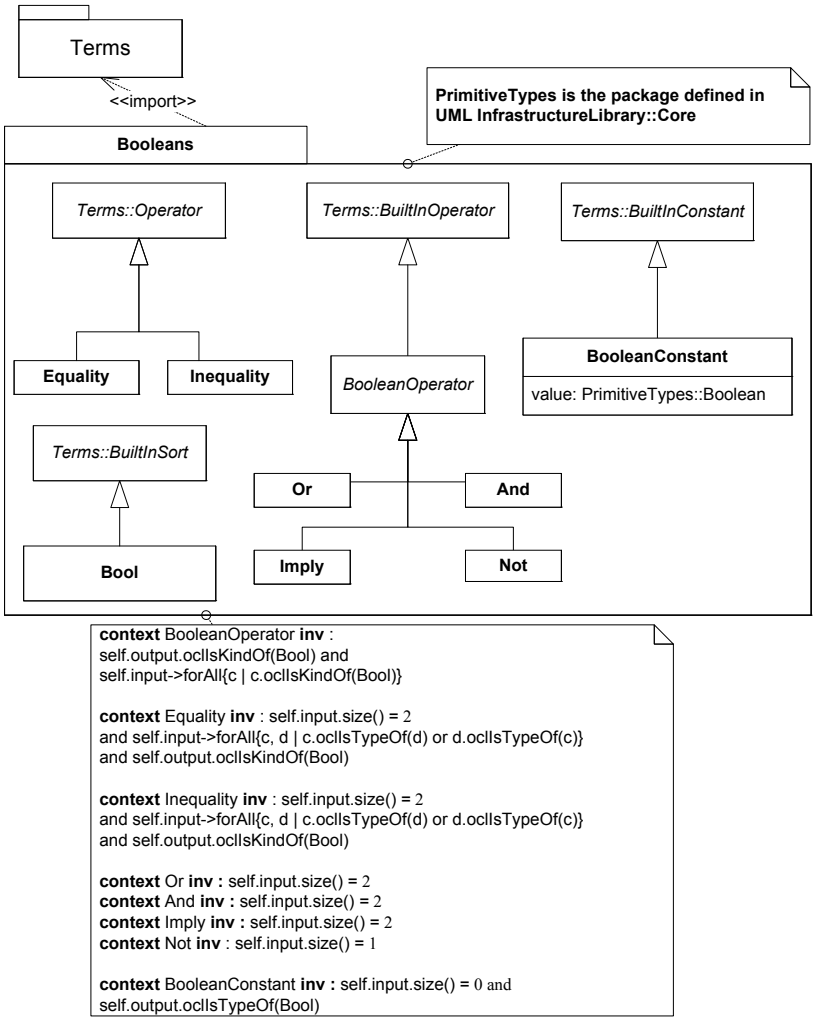

Figure 11: The package *Booleans*

- $\mathrm{CardinalityOf} : \mathrm{Multiset} \times \mathrm{Sort} \rightarrow \mathbb{N}$ is interpreted as the function that returns the number of times the given element occurs in the multiset. For example, the term $\mathrm{CardinalityOf}([a, 2b, c], a)$ evaluates to 1 and the term $\mathrm{CardinalityOf}([a, 2b, c], b)$ evaluates to 2.

- $\mathrm{Contains} : \mathrm{Multiset} \times \mathrm{Multiset} \rightarrow \mathrm{Bool}$ is interpreted as multiset-inclusion function. For example, let $m_1 = [a, b, 2c]$ and $m_2 = [2a, 2b, 2c]$ be two multisets over type $\{a, b, c\}$. Then, $\mathrm{Contains}(m_1, m_2)$ evaluates to false and $\mathrm{Contains}(m_2, m_1)$ evaluates to true.

Multiset literals on typical annotations such as an arc's *HLAnnotation* are defined using Add and $\mathrm{NumberOf}$.

5.3.5 Booleans

Figure 11 shows the UML package for *sort Bool* and its *operators*.

The boolean operators Not, And, Or, Imply operate on boolean arguments as stated by the OCL constraint on *BooleanOperator*. They are interpreted as the classical logical functions.

The $\mathrm{Equality}$ and $\mathrm{Inequality}$ operators take arguments of the same *sort* (not only booleans) and return a *Bool*. They are interpreted as the equality function and the inequality function. Any other built-in sort package may import and use these two operators.

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved
Full document details: BS ISO/IEC 15909-2:2009

The BooleanConstant is a built-in constant for representing the two values of its type.

5.3.6 Finite Enumerations

Figure 12 shows the UML package for *finite enumerations* and their *operators*.

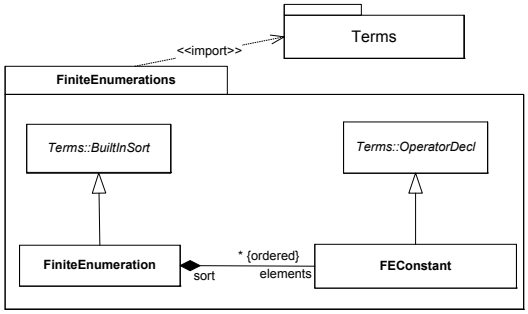

Figure 12: The package *Finite Enumerations* and its built-in *sorts* and *operations*

A *FiniteEnumeration* is defined by explicitly naming all its elements, which are called *FEConstant* in the meta model and the attribute *name* (inherited from *OperatorDecl*) represents the actual name of the respective element. On the one hand, these *FEConstants* are part of the *declaration* of the *FiniteEnumeration sort*. On the other hand, each of these *FEConstants* defines a 0-ary *operation*, i. e. is a *declaration* of a *constant*.

The *type* associated with the declaration of a *FiniteEnumeration* is the set of these constants. The operations on a *FiniteEnumeration* are the use of these *FEConstant* declarations. The *function* that is associated with this *FEConstant* is the 0-ary *function* that returns the respective element of the type.

5.3.7 Cyclic Enumerations

Figure 13 shows the UML package for *cyclic enumerations* and their *operators*.

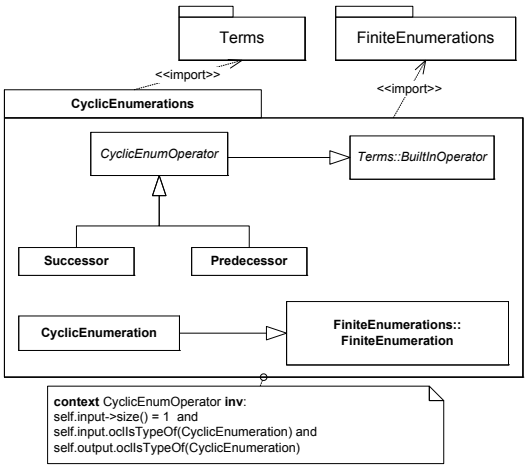

Figure 13: The package *Cyclic Enumerations* and its built-in *sorts* and *operators*

The declaration of a *CyclicEnumeration* has same structure as the *FiniteEnumeration*. This is why *CyclicEnumeration* is derived from *FiniteEnumeration* in the meta model. The only difference are two additional built-in *operations Successor* and *Predecessor*.

Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

The order of the elements on the associated *type* is the order in which its elements are defined in the declaration. And the *functions* associated with the *operations Successor* and the *Predecessor* are the successor resp. predecessor function with respect to this order, where the order wraps at the first and last element.

5.3.8 Finite Integer Ranges

Figure 14 shows the UML package for *finite integer ranges* and their *operators*.

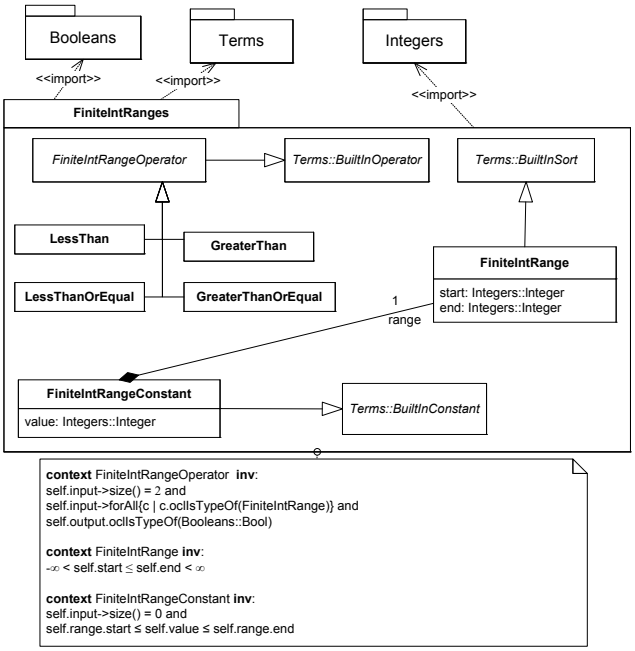

Figure 14: The package *Finite Integer Ranges* and its built-in *sorts* and *operators*

An integer range is defined by refering to *FiniteIntRange*, where the attributes *start* and *end* define the first and the last element of the range. The *type* associated with that *sort* is the set of all integers between (and including) the first and the last element.

The four built-in operations on that type are *LessThan*, *LessThanOrEqual*, *GreaterThan*, and *GreaterThanOrEqual*. The functions associated with these operations are the resp. comparisons with respect to integers.

In addition there are the *constants FiniteIntegerRangeConstant* which are interpreted as a 0-ary function refering to the respective value of the integer range, which is given by the attribute *value* and must refer to a value between the start and end of the respective range. In the package, this is expressed by an OCL constraint.

Note that there is no need to explicitly declare a $\mathrm{FiniteIntRangeConstant}$ in the declarative part of a Petri Net. It can be directly used in the body of the Petri Net and must therefore only refer to the declared range it belongs to.

5.3.9 Partitions

Figure 15 shows the UML package for *Partitions* and their *operators*.

The purpose of a *partition* is to partition the elements of a *type* into equivalence classes. Technically, a $\mathrm{Partition}$ is a *declaration* of a new *sort* refering to the partitioned *sort* and naming all its partitions, each of which is called $\mathrm{PartitionElement}$. Each $\mathrm{PartitionElement}$ is also an *operator declaration*. For each $\mathrm{PartitionElement}$, the declaration lists all the elements that belong to that partition, which are represented as a Term of the partitioned *sort*. The newly declared *partition sort* is then interpreted as the set of all its *partition elements*.

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved
Full document details: BS ISO/IEC 15909-2:2009

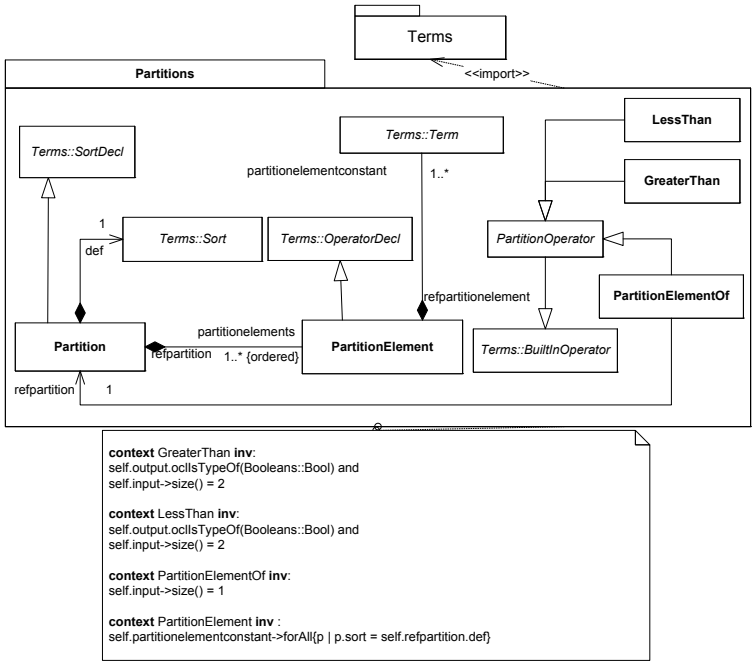

Figure 15: The package *Partitions* and its built-in *sorts* and *operators*

The operations that are defined on a *sort* Partition are the LessThan and GreaterThan. These are interpreted as comparison functions on the partition elements which is based on the order in which they occur in the declaration of the partition.

Moreover, there is an *operator* PartitionElementOf from the partitioned sort to the partition. This operator is interpreted as the function that maps each element of the type associated with the partitioned sort to the partition element it belongs to as defined in the declaration of the partition. Note that in some cases the *partition* to which the *operator* PartitionElementOf refers to cannot be derived from the subterms. Therefore, this partition is explicitly represented by a reference from this *operator* to the *partition*.

5.3.10 Symmetric Nets

Symmetric Nets as currently defined in an Amendment of part 1 of ISO/IEC 15909 (creation of Annex B.2), are *High-Level Petri Net Graphs* with some restrictions. Basically, the carrier sets of all basic *sorts* are finite, and only a fixed set of *operations*, as defined below, are allowed. Therefore, the *sort* of each *place* must not be a *multiset sort*. The available built-in *sorts* are defined below.

Moreover, for *Symmetric Nets*, it is required that every HLPNG *annotation* has the *structural* information.

The built-in *sorts* of *Symmetric Nets* are the following (as defined in the previous Clauses): *Bool*, *range of integers*, *finite enumerations*, *cyclic enumerations* and *dots*. Moreover, for every *sort*, there is the operator *all*, which evaluates to a multiset that contains exactly one occurrence of each of the elements of the type associated with the basis *sort*. This is often called the broadcast function.

Altogether, the *Symmetric Nets* can be defined by importing the packages as shown in Fig. 16.

5.3.11 High-Level Petri Net Graphs

This Clause defines the meta model for *High-Level Petri Net Graphs* (HLPNGs). This, basically, means adding more built-in *sorts* and *operators*; moreover, user *declarations* of arbitrary *sorts* and arbitrary *operators* are introduced.

This clause defines the built-in *sorts* and *operators* of HLPNGs.

Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

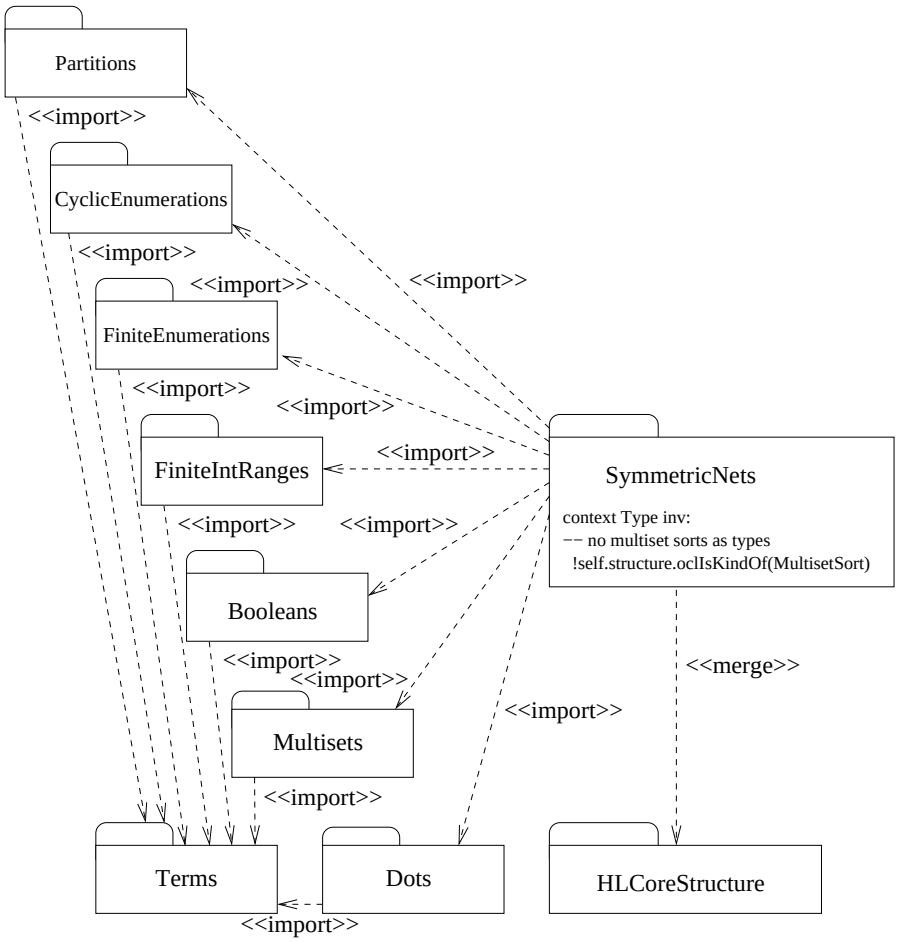

Figure 16: The package *Symmetric Nets* and its built-in *sorts* and *functions*

In addition to the built-in *sorts* and *operators* of *Symmetric Nets*, *High-Level Petri Net Graphs* have the following *sorts* and *operators*:

- Sort *integer* (corresponding to the set of all positive and negative numbers) with *addition*, *subtraction*, *multiplication*, integer *division*, and *modulo* operators as the functions as well as the comparison operators $\leq$, $<$, $>$, and $\geq$. Moreover, there is a constant for any integer number.

  This part of ISO/IEC 15909 does not mandate the exact implementation of integers; it could be finite or infinite.

- The sort of *strings* with the *concatenation* and comparison operators $\leq$, $<$, $>$, and $\geq$ as functions, where the order refers to the lexicographic order.

- For each *sort* there is a *sort* List over that *sort*. Moreover, there is a constant for the *empty list*, an operator for *appending* a single element of the type corresponding to the sort to the list, and an operator for *concatenating* two lists.

Figure 17 shows the UML package *Integers* with the defined *sorts* and *operators*.

Three sets of numbers are defined: $\mathrm{Natural}$, $\mathrm{Positive}$ and $\mathrm{Integer}$. Any number specified as a constant can be declared as an instance of $\mathrm{NumberConstant}$. It refers to one of the sets and its value must be an element of the set it refers to. OCL constraints have been used to enforce this statement. Note that it is not necessary to declare these constants; number constants can be used wherever necessary without an explicit declaration.

Classical arithmetic operators are defined such as $\mathrm{Addition}$, $\mathrm{Subtraction}$, etc. and comparison operators such as $\mathrm{LessThan}$, $\mathrm{GreaterThan}$, etc.

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved
Full document details: BS ISO/IEC 15909-2:2009

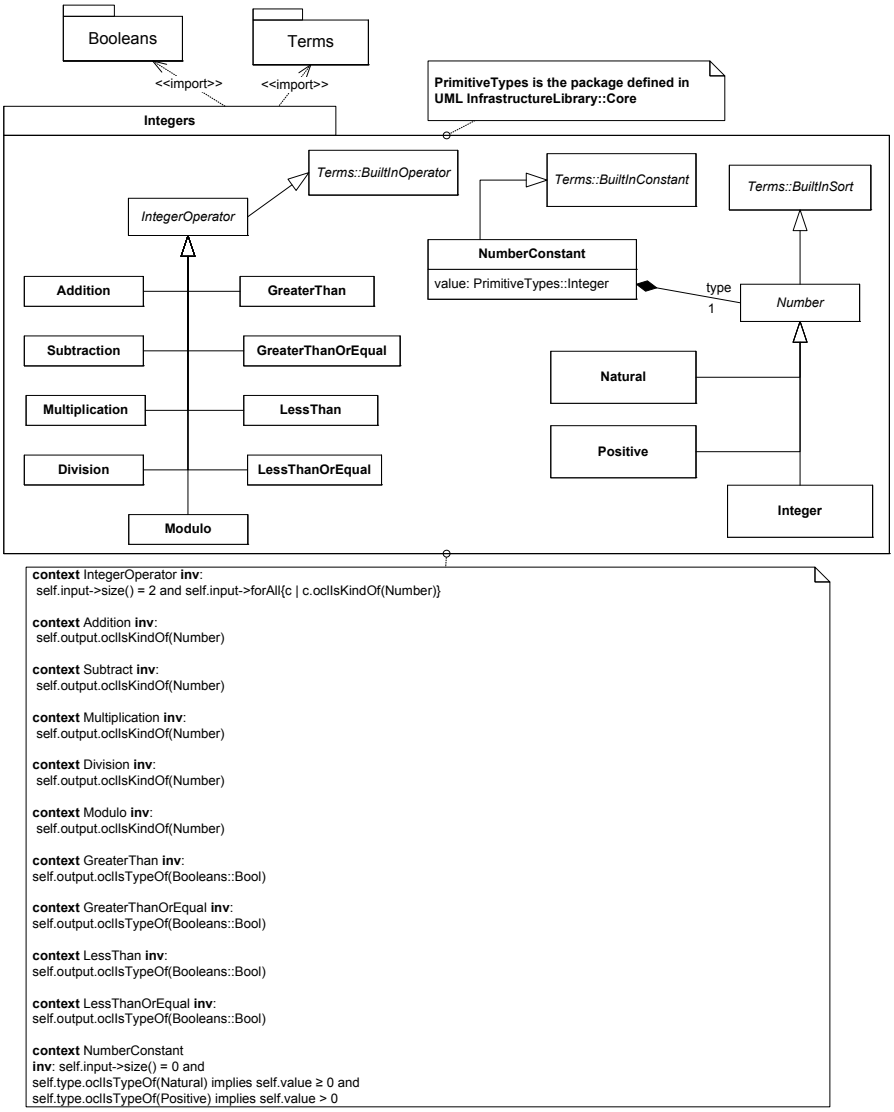

Figure 17: The package *Integers* and its built-in *sorts* and *operators*

Figure 18 shows the UML package *Strings* with the defined *sort* and its *operators*.

Any constant string is defined as an instance of StringConstant, and can be directly used in the body of a Petri Net.

Strings can be concatenated, resulting in a new string. The relational operators compare two strings according to the *lexicographic order*. Their output is thus a boolean. The operator Length of a string returns a natural. Substrings can be extracted from existing strings, using the Substring operator.

Figure 19 shows the UML package *Lists* with the defined *sort* and its *operators*.

A list is constructed over a pre-declared sort using the List class, or over one or more instances of declared sort elements, using MakeList, by enumerating all its elements. Note that in some cases the basis *sort* in the MakeList *operator* cannot be derived from its *subterms*; therefore, this is expliclity represented by a composition in the meta model.

The EmptyList is defined as a built-in constant. As a function, its arity is thus zero. Note that the basis *sort* of the EmptyList *operator* cannot be derived from its *subterms*; therefore, this is expliclity represented by a composition in the meta model.

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

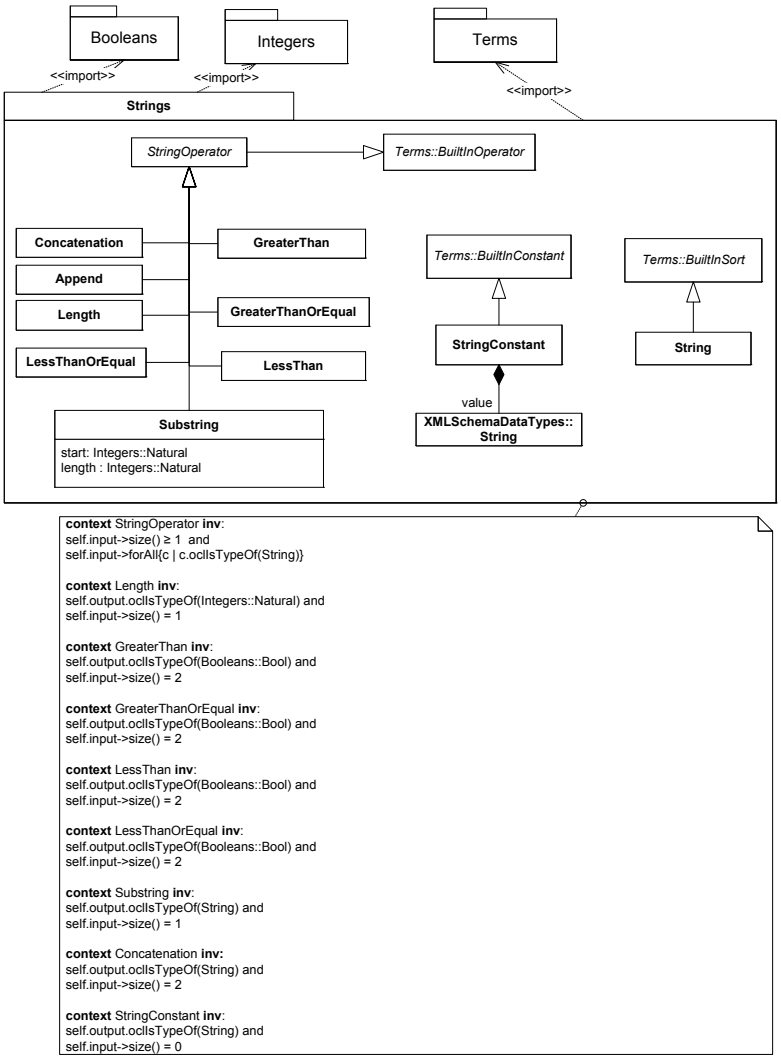

Figure 18: The package *String* and its built-in *sorts* and *operators*

Lists can be concatenated, and the length of a list can be defined. Sublists can be extracted from existing lists, using the Sublist definition. MemberAtIndex returns the element of a List at a given index.

Moreover, HLPNGs allow the user to define arbitrary *sorts* and *operators*. This package *ArbitraryDeclarations* is shown in Fig. 20. In contrast to *named sorts* and *named operators*, arbitrary *sorts* and *operators* do not come with a definition of the *sort* or *operation*; they just introduce a new symbol without giving a definition for it. So, these symbols do not have a meaning, but can be used for constructing *terms*.

The additional concept *Unparsed* in *terms* provides a means to include any text, which will not be parsed and interpreted by the tools. This might be helpful for exchanging the general structure of a term, but not all its details.

The complete definition for *High-Level Petri Net Graphs* is shown in Fig. 21. It extends *Symmetric Nets* by declarations for *sorts* and *functions* and the additional built-in *sorts* for *Integer*, *String*, and *List*.

5.3.12 Place/Transition Nets as High-level Net Graphs

Annex B of part 1 of ISO/IEC 15909 defines *Place/Transition Nets* as a restricted form of *High-level Net Graphs*. Fig. 22 shows the corresponding UML definition. This class allows for the use of the built-in *sorts Bool* and *Dot* only. It does not allow any user *declarations*, neither *variables*, nor *sorts*, nor *operators*.

AVA Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved
Full document details: BS ISO/IEC 15909-2:2009

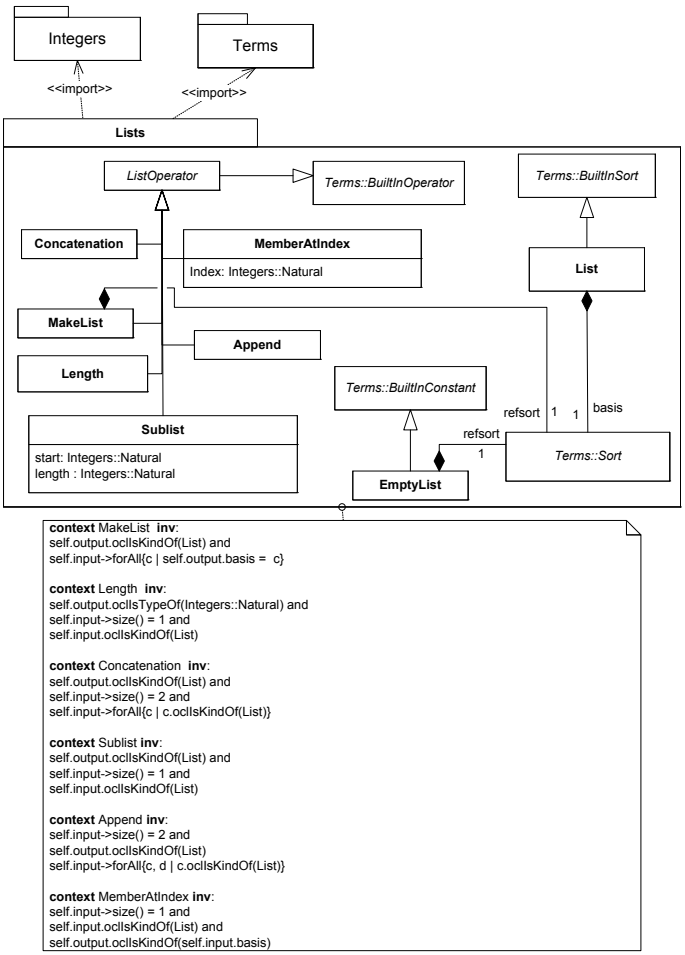

Figure 19: The package *List* and its built-in *sorts* and *operators*

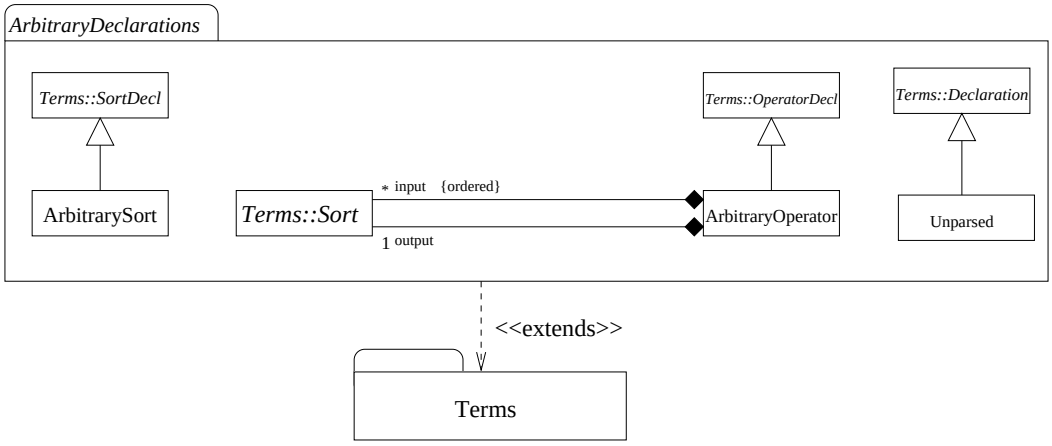

Figure 20: The package *ArbitraryDeclarations*

The *type* of each *place* must refer to *sort Dot*. All transition *conditions* need to be the constant *true*, if this label is present. And the *arc annotations* and the *initial markings* are *ground terms* of the *mulitset sort* over *Dot*.

Note that this *Place/Transition Nets in High-level Notation* is a restricted form of *High-level Net Graphs* and even of *Symmetric Nets*.

Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009

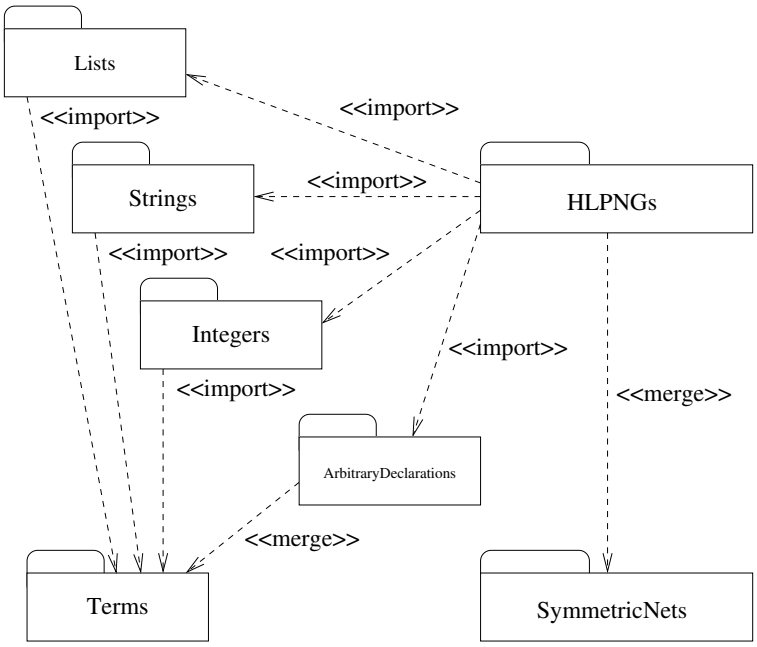

Figure 21: The package *HLPNGs*

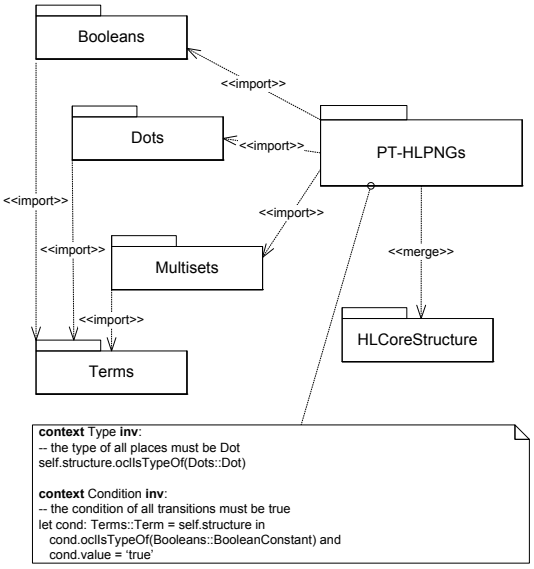

Figure 22: The package of *P/T Nets* defined as restricted *HLPNGs*

Annex B of part 1 of ISO/IEC 15909 defines a mapping from these *Place/Transition Nets in High-level Notation* to the usual mathematical definition of *Place/Transition Nets*. This way, there is a mapping from *Place/Transition Nets in High-level Notation* to the notation given in Clause 5.3.1.

Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved
Full document details: BS ISO/IEC 15909-2:2009

6 Mapping between Part 1 and Part 2

This part of ISO/IEC 15909 defines a transfer format for *High-level Petri Net Graphs*. The concepts of *High-level Petri Net Graphs* have been defined in part 1 of ISO/IEC 15909. The focus of part 1, however, is on the semantical concepts. Therefore, some concepts are defined semantically only. For example, the *type* of a *place* can be any set. But, part 1 does not define any syntax for defining sets. Therefore, this part of ISO/IEC 15909 introduces some syntax for the concepts, which are defined semantically only in part 1. To this end, we use concepts which have been introduced in part 1 already, and which, in particular, have been used in the definition of *High-level Petri Net Schemas* (*HLPNS*) in the informative Annex C of part 1.

Moreover, this part of ISO/IEC 15909 defines some concepts that have not been defined in part 1 of ISO/IEC 15909. These are the concepts for defining the graphical appearance of *Petri Nets* (see Clause 5.2.4) and the concepts for structuring large *Petri Nets* (see Clause 5.2.2).

This Clause discusses the differences of part 1 of ISO/IEC 15909 and this part of ISO/IEC 15909 and maps the syntactical concepts of this part of ISO/IEC 15909 (actually the abstract syntax as defined in Clause 5) to the semantical concepts as defined in Clause 6.2 of part 1 of ISO/IEC 15909.

6.1 Graphics and Structuring

Part 1 of ISO/IEC 15909 defines general rules for the graphical appearance of *Petri Nets* and *High-level Petri Net Graphs* (see Clause 7 of part 1). These rules, basically, say that *places* are shown as ellipses, that *transitions* are shown as rectangles, and that *arcs* are shown as arrows. But there is no information on the size, position, and style of these shapes. Moreover, part 1 states that the other features of *High-level-Petri Net Graphs* are shown as annotations of the associated *object*.

This part of ISO/IEC 15909 allows for the provision of much more information on the graphical appearance of the different objects. This information is more detailed; e. g. concerning colour, line-width, line-style, and fonts, but the graphical result is compatible with the general rules of part 1.

In addition, this part of ISO/IEC 15909 introduces the concepts of *pages* and *reference nodes*. These are purely graphical and do not carry any meaning. By merging the *reference nodes* to the *node* they ultimately refer to, and by ignoring the *labels* of the *reference nodes* and the *page structure*, the additional concepts of this part of ISO/IEC 15909 map to the concepts as defined in part 1 of ISO/IEC 15909. This is called flattening of a *Petri Net* with *pages*.

6.2 Annotations of HLPNGs

In this Clause, it is shown how the concepts as defined in Clause 6.2 and Annex A of part 1 of ISO/IEC 15909 are represented by the concepts as defined in Clause 5.

The *net graph* $NG = (P, T; F)$ is defined by the *places*, *transitions*, and *arcs* of the *Petri Net Document*. As mentioned in Clause 6.1, *pages* and *reference nodes* can be eliminated by flattening.

The *signature* $SIG = (S, O)$ consists of two parts: all the used built-in *sorts* and *operators*; in addition, there are the *user defined sorts* and *operators*.

The S-indexed set of *variables* V is defined by all *declarations* of *variables*.

The many-sorted algebra H is not explicitly given, because there is no syntax for defining algebras. Instead, each built-in *sort* comes with a fixed interpretation, which is a set. These sets are the carrier sets of the algebra. The built-in *operators* have a fixed interpretation, which define the *functions* of the algebra.

Note that, for the *declaration* of *arbitrary sorts* and *operators*, there is no associated meaning. This corresponds to the concepts of *High-level Petri Net Schemas* that do not have an associated algebra (see Annex C of part 1).

The function $Type$ that assigns a *type* to each *place* is defined by the label *type* of the *place*. Since there is no syntax for denoting a type, this label refers to a *sort*. By the fixed interpretation of *sorts*, this implicitly refers to a set, which is the *type* of that *place*.

Documents
Document Sharing Hub

© ISO/IEC 2009 – All rights reserved

Full document details: BS ISO/IEC 15909-2:2009