INTERNATIONAL STANDARD

ISO/IEC 15909-3

First edition
2021-11

Systems and software engineering — High-level Petri nets —

Part 3: Extensions and structuring mechanisms

Ingénierie du logiciel et des systèmes — Réseaux de Petri de haut niveau —

Partie 3: Extensions et mécanismes de structuration

Reference number
ISO/IEC 15909-3:2021(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*.

A list of all parts in the ISO/IEC 15909 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

Petri nets have been used to describe a wide range of systems since their invention in 1962. The technique is mathematically defined and can thus be used to provide unambiguous specifications and descriptions of applications. It is also an executable technique, allowing specification prototypes to be developed to test ideas at the earliest and cheapest opportunity. Specifications written in the technique can be subjected to analysis methods to prove properties about the specifications, before implementation commences, thus saving on testing and maintenance time and providing a high level of quality assurance.

A problem with Petri nets is the explosion of the number of elements in their graphical form when they are used to describe complex systems. High-level Petri nets were developed to overcome this problem by introducing higher level concepts, such as the use of complex structured data as tokens, and using algebraic expressions to annotate net elements. The use of "high-level" to describe these Petri nets is analogous to the use of "high-level" in high-level programming languages (as opposed to assembly languages), and is the usual term used in the Petri net community. Two of the early forms of high-level nets that this document builds on are predicate-transition nets and coloured Petri nets, first introduced in 1979 and developed during the 1980s. It also uses some of the notions developed for algebraic Petri nets, first introduced in the mid-1980s. It is believed that this document captures the spirit of these earlier developments (see Bibliography).

The technique has multiple uses. For example, it can be used directly to specify systems or to define the semantics of other less formal languages. It can also serve to integrate techniques currently used independently such as state-transition diagrams and data flow diagrams. The technique is particularly suited to parallel and distributed systems development as it supports concurrency. The technique is able to specify systems at a level that is independent of the choice of implementation (i.e. by software, hardware (electronic and/or mechanical) or humans or a combination). This document may be cited in contracts for the development of systems (particularly distributed systems) or used by application developers or Petri net tool vendors or users.

The ISO/IEC 15909 series is concerned with defining a modelling language and its transfer format, known as high-level Petri nets. ISO/IEC 15909-1 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. It also introduces some common notational conventions for HLPNGs.

ISO/IEC 15909-2 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 types of Petri nets in addition to high-level Petri nets, ISO/IEC 15909-2 defines the core concepts of Petri nets along with an XML syntax, which can be used for exchanging any kind of Petri nets. Based on this PNML core model, ISO/IEC 15909-2 also defines the transfer syntax for the types of Petri nets that are defined in ISO/IEC 15909-1: place/transition nets, symmetric nets[1)], high-level Petri nets, Petri nets with priorities, and Petri nets with time. Place/transition nets and symmetric nets can be considered to be restricted versions of high-level Petri nets. Petri nets with priorities and Petri nets with time are considered as extensions of the other types.

This document defines extensions to the types of Petri nets that are defined in ISO/IEC 15909-1. These extensions comprise enrichments of Petri net types and definitions of new Petri net types. This document also defines structuring mechanisms for these Petri net types.

In this document, the semantics which is considered is always the interleaving semantics.

This document provides an abstract mathematical syntax and a formal semantics for the technique. Conformance to the document is possible at several levels. The level of conformance depends on the class of high-level net chosen. The usual graphical notations are depicted in Annex A.

1) Symmetric nets have been first introduced as well-formed nets and are currently standardized as ISO/IEC 15909-1.

Systems and software engineering — High-level Petri nets —

Part 3:
Extensions and structuring mechanisms

1 Scope

This document defines enrichments, extensions and structuring mechanisms of Petri nets, applied on the definitions proposed in ISO/IEC 15909-1. This document facilitates the definitions of new kinds of Petri nets and their interoperability, while remaining compatible with those defined in ISO/IEC 15909-1.

This document is written as a reference for designers of new Petri net variants, by defining common enrichments, extensions and structuring mechanisms, as well as a generalized process for defining new ones.

This document is applicable to a wide variety of concurrent discrete event systems and in particular distributed systems. Generic fields of application include:

— requirements analysis;

— development of specifications, designs and test suites;

— descriptions of existing systems prior to re-engineering;

— modelling business and software processes;

— providing the semantics for concurrent languages;

— simulation of systems to increase confidence;

— formal analysis of the behaviour of systems;

— and development of Petri net support tools.

This document can be applied to the design of a broad range of systems and processes, including aerospace, air traffic control, avionics, banking, biological and chemical processes, business processes, communication protocols, computer hardware architectures, control systems, databases, defence command and control systems, distributed computing, electronic commerce, fault-tolerant systems, games, hospital procedures, information systems, Internet protocols and applications, legal processes, logistics, manufacturing systems, metabolic processes, music, nuclear power systems, operating systems, transport systems (including railway control), security systems, telecommunications and workflow.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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 15909-1, *Systems and software engineering — High-level Petri nets — Part 1: Concepts, definitions and graphical notation*

3 Terms and definitions

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

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
capacity
maximum multiset of tokens a *capacity place* (3.2) can hold

3.2
capacity place
special kind of place that can hold no more than a specified *capacity* (3.1)

3.3
FIFO
special kind of queue that is operated first in first out

3.4
inhibitor arc
special kind of arc that reverses the logic of an input place

Note 1 to entry: Instead of testing the presence of some tokens in the related place, it tests the lack of these.

3.5
read arc
special kind of arc that tests the presence of some tokens in the related place, without consumption

3.6
reset arc
special kind of arc that empties an input place

3.7
sort generator
generator of new sorts and operators built from a given signature (passed as a parameter)

4 Conformance

4.1 General

There are different levels of conformance to this document.

4.2 Enrichment process

A Petri net model is conformant to the enrichment process level when it contains a subset of the features defined as enrichments by Clause 5.

4.3 Extension process

A Petri net model is conformant to the extension process level when it contains a single feature defined as extension by Clause 6.

4.4 Structuring mechanism

A Petri net model is conformant to the structuring mechanism level when it embeds a structuring feature as defined by Clause 7.

5 Enrichment process

5.1 General

This clause defines enrichments of Petri nets. For each enrichment, it first provides a syntactic definition, then its individual semantics, and finally a formulation of the semantics allowing for composition with other enrichments or extensions.

5.2 Instances of enrichment

5.2.1 General

This subclause defines commonly used enrichments. The list provided here is not exhaustive. The general mechanism to define a new enrichment is provided in 5.3.

NOTE Enrichments are orthogonal one to another. Hence, they can be used in combination among themselves, and to enrich the different types of nets defined in this document.

5.2.2 Inhibitor arcs

5.2.2.1 Definition of Petri nets with inhibitor arcs

A Petri net with inhibitor arcs is a Petri net N together with a matrix $I \in AN^{|P|\times|T|}$ of inhibitor arcs.

5.2.2.2 Definition of enabling rule for Petri nets with inhibitor arcs

A transition $t \in T$ is enabled in marking M, denoted by $M[t\rangle$, iff:

$$\forall p \in {\bullet t}, (M(p) \geq \mathrm{Pre}(p,t)) \wedge (I(p,t) = 0 \vee M(p) < I(p,t))$$

5.2.2.3 Filtering function for inhibitor arcs

The filtering function $F_{\mathrm{i}}\,(N \in \mathcal{N}, t \in T_N)$ returns true when there are less tokens than the value specified on the inhibitor arc ($I(p,t) \neq 0$).

$$F_{\mathrm{i}}\,(N \in \mathcal{N}, t \in T_N) : \begin{cases} \text{True, iff } \forall p \in {\bullet t} : M(p) < I(p,t) \\ \text{False, otherwise} \end{cases}$$

The enabling function for place/transition nets with inhibitor arcs is thus:

$$E_{\mathrm{pti}}\,(N,t) = E_{\mathrm{pt}}\,(N,t) \wedge F_{\mathrm{i}}\,(N,t)$$

The enabling function for symmetric nets with inhibitor arcs is thus:

$$E_{\mathrm{sni}}\,(N,t) = E_{\mathrm{sn}}\,(N,t) \wedge F_{\mathrm{i}}\,(N,t)$$

The enabling function for high-level Petri nets with inhibitor arcs is thus:

$$E_{\mathrm{hlpni}}\,(N,t) = E_{\mathrm{hlpn}}\,(N,t) \wedge F_{\mathrm{i}}\,(N,t)$$

NOTE Multiple combinations of filtering functions can be defined (see 5.3 and 6.3), e.g. for Petri nets with priorities and inhibitor arcs.

5.2.2.4 Firing rule for Petri nets with inhibitor arcs

The firing rule for Petri nets with inhibitor arcs is the same as for the corresponding Petri net type. Hence, A_{pre} and A_{post} are those of the corresponding Petri net type.

5.2.3 Reset arcs

5.2.3.1 Definition of Petri nets with reset arcs

A Petri net with reset arcs is a Petri net N together with a matrix $Rt \in \{0,1\}^{|P|\times|T|}$ of reset arcs.

5.2.3.2 Definition of enabling rule for Petri nets with reset arcs

A transition $t \in T$ is enabled in marking M, denoted by $M[t\rangle$, iff:

$$\forall p \in {\bullet t}, M(p) \geq \mathrm{Pre}(p,t)$$

5.2.3.3 Filtering function for reset arcs

The filtering function $F_{\mathrm{reset}}(N \in \mathcal{N}, t \in T_N)$ returns true for all places.

5.2.3.4 Augmented firing rule for reset arcs

The firing rule for Petri nets with reset arcs relies on the one of the corresponding Petri net type. A_{pre} is the one of the corresponding Petri net type. A_{post} is the one of the corresponding Petri net type augmented by removing all tokens referenced in AN from the connected place.

5.2.4 Read arcs

5.2.4.1 Definition of Petri nets with read arcs

A Petri net with read arcs is a Petri net N together with a matrix $Rd \in AN^{|P|\times|T|}$ of read arcs.

5.2.4.2 Definition of enabling rule for Petri nets with read arcs

A transition $t \in T$ is enabled in marking M, denoted by $M[t\rangle$, iff:

$$\forall p \in {\bullet t}, \ (M(p) \geq \mathrm{Pre}(p,t)) \wedge M(p) \geq Rd(p,t)$$

5.2.4.3 Filtering function for read arcs

The filtering function $F_{\mathrm{read}}(N \in \mathcal{N}, t \in T_N)$ is the same as the enabling function, applied to Rd.

5.2.4.4 Augmented firing rule for read arcs

The firing rule for Petri nets with read arcs is the same as for the corresponding Petri net type. Hence, A_{pre} and A_{post} are those of the corresponding Petri net type.

NOTE Tokens in read arc annotation $Rd(p,t)$ are not consumed.

5.2.5 Capacity places

5.2.5.1 Definition of Petri nets with capacity places

A Petri net with capacity place is a Petri net N together with a vector $C \in (\mathbb{N} \cup \{\infty\})^{|P|}$ of place capacities.

5.2.5.2 Definition of enabling rule for Petri nets with capacity places

A transition $t \in T$ is enabled in marking M, denoted by $M[t\rangle$, iff:

$$[\forall p \in {\bullet t} : M(p) \geq \mathrm{Pre}(p,t)] \wedge [\forall p' \in t{\bullet} : M(p') + \mathrm{Post}(p',t) - \mathrm{Pre}(p',t) \leq C(p')]$$

NOTE For all Petri net types, only the number of tokens in a place is limited by its capacity.

5.2.5.3 Filtering function for capacity places

The filtering function $F_c(N \in \mathcal{N}, t \in T_N)$ returns true when the firing would not lead to exceeding the capacity for output places.

$$F_c(N \in \mathcal{N}, t \in T_N) : \begin{cases} \text{True, iff } [\forall p \in {\bullet t} : M(p) \geq \mathrm{Pre}(p,t)] \wedge [\forall p' \in t{\bullet} : M(p') + \mathrm{Post}(p',t) - \mathrm{Pre}(p',t) \leq C(p')] \\ \text{False, otherwise} \end{cases}$$

The enabling function for place/transition nets with capacity places is thus:

$$E_{\mathrm{ptc}}(N,t) = E_{\mathrm{pt}}(N,t) \wedge F_c(N,t)$$

The enabling function for symmetric nets with capacity places is thus:

$$E_{\mathrm{snc}}(N,t) = E_{\mathrm{sn}}(N,t) \wedge F_c(N,t)$$

The enabling function for high-level Petri nets with capacity places is thus:

$$E_{\mathrm{hlpnc}}(N,t) = E_{\mathrm{hlpn}}(N,t) \wedge F_c(N,t)$$

NOTE Multiple combinations of filtering functions can be defined (see 5.3 and 6.3), e.g. for Petri nets with priorities and capacity places.

5.2.5.4 Firing rule for Petri nets with capacity place

The firing rule for Petri nets with capacity places is the same as for the corresponding Petri net type. Hence, A_{pre} and A_{post} are those of the corresponding Petri net type.

5.3 Generalized enrichment process

5.3.1 General

This subclause defines the template used to define a new enrichment to an existing Petri net type.

5.3.2 Definition of Petri nets with enrichment

A Petri net with ε is a Petri net N together with a definition D of the enrichment according to the net elements.

5.3.3 Definition of enabling rule for Petri nets with enrichment ε

A transition $t \in T$ is enabled in marking M, denoted by $M[t\rangle$, iff: $M(p) \geq \mathrm{Pre}(p,t) \wedge \mathrm{Cond}(\varepsilon,t) = \mathrm{true}$, where $\mathrm{Cond}(\varepsilon,t)$ is the condition enforced by enrichment ε on the enabling of transition t.

5.3.4 Filtering function for enrichment ε

The filtering function $F_\varepsilon\,(N \in \mathcal{N}, t \in T_N)$ returns true when $\mathrm{Cond}(\varepsilon,t)$ is satisfied.

$$F_\varepsilon\,(N \in \mathcal{N}, t \in T_N): \begin{cases} \text{True, iff } \mathrm{Cond}(\varepsilon,t) \\ \text{False, otherwise} \end{cases}$$

The enabling function for place/transition nets with enrichment ε is thus:

$$E_{\mathrm{pte}}\,(N,t) = E_{\mathrm{pt}}\,(N,t) \wedge F_\varepsilon\,(N,t)$$

The enabling function for symmetric nets with enrichment ε is thus:

$$E_{\mathrm{sne}}\,(N,t) = E_{\mathrm{sn}}\,(N,t) \wedge F_\varepsilon\,(N,t)$$

The enabling function for high-level Petri nets with enrichment is thus:

$$E_{\mathrm{hlpne}}\,(N,t) = E_{\mathrm{hlpn}}\,(N,t) \wedge F_\varepsilon\,(N,t)$$

5.3.5 Firing rule for Petri nets with enrichment ε

The firing rule for Petri nets with enrichment ε shall define $\mathrm{A}_{\mathrm{pre}}$ and $\mathrm{A}_{\mathrm{post}}$ according to the Petri net type.

5.3.6 Compatibility with extensions

By default, an enrichment ε cannot be combined with any extension as defined in Clause 6, unless explicitly stated.

6 Extension process

6.1 General

This clause defines extensions of Petri nets. For each extension it provides its syntax and semantics. This clause then defines a generalization of the extension process, allowing for user-defined extensions, compatible with this document.

6.2 An instance of extension: FIFO nets

6.2.1 General

FIFO nets are nets Petri nets where some places behave as a FIFO queue. When adding an element in the place, it is pushed in the queue, while retrieving an element corresponds to popping it from the queue. FIFO nets can be built upon any net type, but using a place/transition net as a basis is meaningless. Indeed, the FIFO places would not contain distinguished elements in their queue, and thus would completely be equivalent to a normal place.

6.2.2 Definition of FIFO nets

A FIFO net is a net $N_f = \langle P_f = P \cup Q, T_f, \mathrm{Pre}_f, \mathrm{Post}_f, D_f, \mathcal{C}_f, V_f, \Sigma_f, AN_f, G_f \rangle$ where Q is a set of FIFO places and $P \cap Q = \varnothing$.

6.2.3 Behavioural semantics

The semantics of FIFO nets is similar to the semantics of high-level Petri nets, except for handling FIFO places, which behave as queues. Each arc expression input of a FIFO place determines the value which is pushed into the FIFO queue. Similarly, each output arc carries an expression which is mapped to the value popped from the queue. Hence, the arc expressions are exactly the same as in high-level Petri nets.

The marking of a FIFO place is a queue.

6.2.4 Definition of equivalent high-level Petri net

Let N_f be a FIFO net. Its equivalent high-level Petri net is

$N = \langle P_f, T_f, \mathrm{Pre}, \mathrm{Post}, D, \mathcal{C}, V, \Sigma, AN, G \rangle$ such that:

— $\mathrm{D} = D_f \cup \{\mathrm{Fifo}(\mathcal{C}_f(q)), \mathrm{q} \in Q\}$, where $\mathrm{Fifo}(X)$ is the type of FIFO queues over type X

— $\mathcal{C}(p) = \begin{cases} \mathcal{C}_f(p), & \text{if } p \in P \\ \mathrm{Fifo}(\mathcal{C}_f(p)), & \text{if } p \in Q \end{cases}$

— $V = V_f \cup \{f_q, q \in Q\}$

— $\Sigma = \Sigma_f \cup \{(\mathrm{Fifo}(\mathcal{C}(q)), \{\mathrm{push}, \mathrm{pop}, \mathrm{top}, \mathrm{empty}\}), q \in Q\}$

— $AN = \mathrm{TERM}(O_\Sigma \cup V)$

— $\mathrm{Pre}(p,t) = \begin{cases} \mathrm{Pre}_f(p,t), & \text{if } p \in P \\ f_p, & \text{if } p \in Q \cap (\bullet t \cup t \bullet) \end{cases}$

— $\mathrm{Post}(p,t) = \begin{cases} \mathrm{Post}_f(p,t), & \text{if } p \in P \\ \mathrm{pop}(f_p), & \text{if } p \in Q \cap \bullet t \\ \mathrm{push}(\mathrm{Pre}_f(p,t), f_p), & \text{if } p \in Q \cap t \bullet \end{cases}$

— $\mathrm{G}(t) = G_f(t) \wedge_{p \in Q \cap \bullet t} \mathrm{Pre}_f(p,t) = \mathrm{top}(f_p)$

A FIFO Petri net and its equivalent high-level Petri net are isomorphic.

6.2.5 Compatibility with enrichments

The FIFO nets extension is compatible with the inhibitor arcs, reset arcs and capacity places enrichments, defined in 5.2.2, 5.2.3 and 5.2.5.

6.3 The generalized extension process

6.3.1 General

The generalized extension process comprises the following steps:

a) definition of the net type;

b) definition of the behavioural semantics (enabling and firing rules).

6.3.2 Definition of the net type

The concepts defining the new Petri net type shall be provided in an unambiguous mathematical syntax. When the definition bases on an already existing net type in the ISO/IEC 15909 series, the notations used shall be consistent. Moreover, the relationship between the two net types shall be made explicit.

Textual explanation of the concepts shall capture their specifics.

The graphical notation shall be defined when relevant.

6.3.3 Definition of the behavioural semantics

The behavioural semantics for enabling and firing rules shall be defined either explicitly or through an equivalent high-level net. When the net type bases on an already existing net type in the ISO/IEC 15909 series, the semantics of the new type shall agree with the existing one.

The semantics for both the enabling rules and for the firing rules shall be defined, similar to the definitions in the ISO/IEC 15909 series. The firing rule should apply to one or more transitions chosen non-deterministically among all enabled transitions.

7 Structuring mechanism

7.1 General

This clause formalizes the notion of modules, their interfaces and implementation. Modules are the basic structuring mechanism of Petri nets, whereby implementation is encapsulated and their behaviour is exposed via interfaces. Modules interact with one another through their interfaces. The module interface describes which places and transitions are imported or exported, and which sort and operation symbols are imported or exported on a purely syntactical level.

7.2 Module definition

7.2.1 Definition of sort generator

Let $\Sigma=(S,O)$ be an arbitrary signature, then $GS(\Sigma)=(S',O')$ is defined as follows:

— S' is the least set of sorts such that:

  a) $S\subseteq S'$,

  b) $(\text{bool})\in S'$,

  c) $(ms,s)\in S'$ for every $s\in S'$, and

  d) $(\times,s_1,\ldots,s_n)\in S'$ for all sorts $s_1,\ldots,s_n\in S'$.

— O' is the least S'-indexed set of operators such that:

  a) $O\subseteq O'$,

  b) $(\text{true}),(\text{false})\in O'_{(\text{bool})}$,

  c) $(\text{not},(\text{bool}))\in O'_{(\text{bool})(\text{bool})}$,

  d) $(\text{and},(\text{bool}),(\text{bool})),(\text{or},(\text{bool}),(\text{bool}))\in O'_{(\text{bool})(\text{bool})(\text{bool})}$,

  e) $(([\ ],s),s,\ldots,s)\in O'_{s\ldots s(ms,s)}$ for every sort $s\in S'$, where the number of s is the same in both constructs,

f) $(+,(ms,s),(ms,s)) \in O'_{(ms,s)(ms,s)(ms,s)}$ for every $s \in S'$, and

g) $((\ ),s_1,\ldots,s_n) \in O'_{s_1\ldots s_n(\times,s_1,\ldots,s_n)}$ for all $s_1,\ldots,s_n \in S'$.

7.2.2 Definition of module interface

A module interface $I=((\Sigma_I,P_I,T_I),(\Sigma_O,P_O,T_O),\text{sort}_{IO})$ consists of two signatures Σ_I and Σ_O with disjoint sets of symbols, four pairwise disjoint sets P_I, T_I, P_O, T_O and a mapping $\text{sort}_{IO}: P_I \cup P_O \to S^{GS(\Sigma_I \cup \Sigma_O)}$.

The elements of the module interface are called: (Σ_I,P_I,T_I) the import interface, Σ_I the imported signature, P_I the imported places, and T_I the imported transitions, (Σ_O,P_O,T_O) the export interface, Σ_O the exported signature, P_O the exported places, and T_O the exported transitions.

The mapping sort_{IO} assigns a sort to every place of the interface. Note that this can be any sort that can be generated from the sorts of the import and export signatures.

7.2.3 Definition of module implementation

Let $I=((\Sigma_I,P_I,T_I),(\Sigma_O,P_O,T_O),\text{sort}_{IO})$ be a module interface. Then, a module implementation $M=(I,N,A)$ of interface I, consists of:

a) the interface I itself;

b) a Net N with signature $\Sigma=(S,O)$ that extends Σ_I and Σ_O, and such that $P \supseteq P_I \cup P_O$, $T \supseteq T_I \cup T_O$, and $\text{sort} \supseteq \text{sort}_{IO}$;

c) a $\Sigma \setminus \Sigma_I$-algebra A.

Note that this definition does not require that A is a Σ-algebra since some symbols from Σ are imported from Σ_I. The interpretation of the symbols from Σ_I comes from the imported symbols when the module is instantiated. To assign the meaning to the remaining symbols, $\Sigma \setminus \Sigma_I$ shall be a signature; and A shall be a $\Sigma \setminus \Sigma_I$-algebra.

Let J be a set of module interfaces, which can then be used for defining another module. For each $I_k \in J$, it is denoted: $I_k=\left(\left(\Sigma_I^{I_k},P_I^{I_k},T_I^{I_k}\right),\left(\Sigma_O^{I_k},P_O^{I_k},T_O^{I_k}\right),sort_{IO}^{I_k}\right), \Sigma_I^{I_k}=\left(S_I^{I_k},O_I^{I_k}\right)$ and $\Sigma_O^{I_k}=\left(S_O^{I_k},O_O^{I_k}\right)$.

7.3 Module instantiation

7.3.1 General

First, a notation for module instances and the use of modules is introduced.

7.3.2 Definition of module instances and uses

For some $n \in \mathbb{N}$ and for every $k \in \{1,\ldots,n\}$, let $I_k \in J$. Then, $U=\{(1,I_1),\ldots,(n,I_n)\}$ is a set of n module instances of J. The set U is called the module uses.

Note that the interfaces I_k are not required to be different since the same module may be used multiple times in another module definition (i.e., several instances). Therefore, to be able to distinguish different instances of the same module, a different number is associated with each of them.

7.3.3 Definition of module definition

A module definition $D=\left(I,N,U,(si_k)_{k=1}^n,(pi_k)_{k=1}^n,(ti_k)_{k=1}^n,(so_k)_{k=1}^n,(po_k)_{k=1}^n,(to_k)_{k=1}^n,A\right)$ for interface I_k over some module interfaces J, consists of the following:

a) its own interface $I=((\Sigma_I,P_I,T_I),(\Sigma_O,P_O,T_O),\text{sort}_{IO})$,

b) a net N with signature $\Sigma=(S,O)$,

c) a set of module instances $U=\{(1,I_1),\ldots,(n,I_n)\}$ of J,

d) for each $k\in\{1,\ldots,n\}$,

1) a signature homomorphism $si_k:\Sigma_I^{I_k}\to\Sigma$,

2) an injective signature homomorphism $so_k:\Sigma_O^{I_k}\to\Sigma\setminus\Sigma_I$,

3) a mapping $pi_k:P_I^{I_k}\to P$,

4) an injective mapping $po_k:P_O^{I_k}\to P\setminus P_I$,

5) a mapping $ti_k:T_I^{I_k}\to T$,

6) an injective mapping $to_k:T_O^{I_k}\to T\setminus T_I$,

such that the co-domains of all homomorphisms so_k are pairwise disjoint, the co-domains of all mappings po_k are pairwise disjoint, and the co-domains of all mappings to_k are pairwise disjoint. Moreover, for every k and for every $p\in P_I^{I_k}$, $si_k\left(\text{sort}_{IO}^{I_k}(p)\right)=\text{sort}(pi_k(p))$, and for every $p\in P_O^{I_k}$, $so_k\left(\text{sort}_{IO}^{I_k}(p)\right)=\text{sort}(po_k(p))$, such that $\Sigma_D=(S',O')$ with $S'=S\setminus\left(S^{\Sigma_I}\cup\bigcup_{k=1}^n so_k\left(S^{\Sigma_O^{I_k}}\right)\right)$ and $O'=O\setminus\left(O^{\Sigma_I}\cup\bigcup_{k=1}^n so_k\left(O^{\Sigma_O^{I_k}}\right)\right)$ is a signature, and

e) A is a Σ_D-algebra.

Basically, the module definition consists of a net N, where the homomorphisms and mappings (see condition d) from the interfaces of the used module instances U to Σ indicate how the instances of the modules are embedded into Σ. This concerns the embedding of places and transitions as well as the use of the different symbols of the signatures. Note that if an export symbol of a module instance is mapped to a symbol of Σ, this symbol gets its meaning from this module instance. Therefore, condition d) requires that these mappings do not overlap. The meaning of the symbols of the import signature is defined when the module is used; therefore, the module definition itself does not need to give a definition to these symbols. Therefore, the algebra A does not need to assign a meaning for the symbols coming from the import signature of the defined module or from the export symbols of the used modules.

The remaining part of the signature, denoted by Σ_D in the above definition, shall be a signature and A shall be a Σ_D-algebra (condition e).

The module implementation is inferred from a module definition (i.e. based on other modules).

Consider a module definition

$$D=\left(I,N,U,(si_k)_{k=1}^n,(pi_k)_{k=1}^n,(ti_k)_{k=1}^n,(so_k)_{k=1}^n,(po_k)_{k=1}^n,(to_k)_{k=1}^n,A\right)$$

as defined in 7.3.3 using module instances $U=\{(1,I_1),\ldots,(n,I_n)\}$. To define the module implementation, the implementations of the used modules shall be known. Assume that for each $k\in\{1,\ldots,n\}$, $M_k=(I_k,\Sigma_k,A_k)$ is an implementation for interface I_k.

The basic idea of the module defined by D is to make a disjoint union of all signatures and nets of the implementations and the module definition itself, and to transform the arc, place, and transition labels accordingly. However, some parts need to be identified, as defined by the homomorphism between the signatures and the mappings from the interface places and transitions to the places and transitions of the module definition.

Start with defining the signature of the module implementation, which is denoted with $\hat{\Sigma}$. First, summarize the available signatures:

a) The signature $\Sigma=(S,O)$ from the module definition.

b) For every use of a module (k,I_k), there are two disjoint signatures $\Sigma_I^{I_k}$ and $\Sigma_O^{I_k}$. Define $\Sigma^{I_k}=\Sigma_I^{I_k}\cup\Sigma_O^{I_k}$. Moreover, there is a signature homomorphism $f_k:\Sigma^{I_k}\to\Sigma$ defined by $f_k=so_k\cup si_k$.

c) For every use of a module (k,I_k), the implementation $M_k=(I_k,\Sigma_k,A_k)$ has a signature Σ_k and variables X_k. By definition $\Sigma^{I_k}\subseteq\Sigma_k$ holds.

7.3.4 Definition of signature $\hat{\Sigma}$ and homomorphisms σ_k

For each $k\in\{1,\ldots,n\}$, the mapping σ_k is a signature homomorphism from Σ_k to $\hat{\Sigma}$, defined as follows: $\hat{\Sigma}=\left(\hat{S},\hat{O}\right)$ for every $x\in\Sigma_k$ for which $f_k(x)$ is defined: $\sigma_k(x)=f_k(x)$; for every other symbol $x\in\Sigma_k:\sigma_k(x)=(k,x)$. Define $\hat{\Sigma}=\left(\hat{S},\hat{O}\right)$ by $\hat{S}=S\cup\bigcup_{k=1}^n\sigma_k\left(S^{\Sigma_k}\right)$ and $\hat{O}=O\cup\bigcup_{k=1}^n\sigma_k\left(O^{\Sigma_k}\right)$, where the arities carry over by interpreting σ_k as a signature homomorphism.

Note that, in the definition of $\sigma_k(x)$, the pair (k,x) is used to make this symbol of Σ_k different from all the other symbols. The signature $\hat{\Sigma}$ is the signature of the defined module implementation. The signature homomorphisms σ_k relate the signatures of the implementations to $\hat{\Sigma}$; they are used to transfer the labels from the different module implementations to the defined module implementation.

The variables from the different modules are made disjoint in the same way.

7.3.5 Definition of variables $\hat{X}$

For every $k\in\{1,\ldots,n\}$, the mapping ξ_k is defined by: $\xi_k(x)=(k,x)$ for every variable $x\in X_k$. The set of all variables is defined by $\hat{X}=X\cup\bigcup_{k=1}^n\xi_k\left(X_k\right)$.

The meaning of the non-imported symbols of $\hat{\Sigma}$ is defined by an $\hat{\Sigma}\setminus\Sigma_I$-algebra $\hat{A}$. Basically, this meaning carries over from the other algebras via the respective homomorphisms.