Systems Biology Graphical Notation: Process Description language Level 1

Version 2.0

Date: November 28, 2012

Disclaimer: This is a working draft of the SBGN Process Description Level 1 Version 2.0 specification. It is not a normative document.

Editors:

Stuart Moodie	*EMBL European Bioinformatics Institute, UK*
Nicolas Le Novère	*EMBL European Bioinformatics Institute, UK*
Emek Demir	*Sloan-Kettering Institute, USA*
Huaiyu Mi	*University of Southern California, USA*
Alice Viléger	*London, UK*

Contents

Chapter 1

Introduction

The goal of the **S**ystems **B**iology **G**raphical **N**otation (SBGN) is to standardize the graphical/visual representation of biochemical and cellular processes. SBGN defines comprehensive sets of symbols with precise semantics, together with detailed syntactic rules defining their use. It also describes the manner in which such graphical information should be interpreted. For a general description of SBGN, one can read:

> Nicolas Le Novère, Michael Hucka, Huaiyu Mi, Stuart Moodie, Falk Schreiber, Anatoly Sorokin, Emek Demir, Katja Wegner, Mirit I Aladjem, Sarala M Wimalaratne, Frank T Bergman, Ralph Gauges, Peter Ghazal, Hideya Kawaji, Lu Li, Yukiko Matsuoka, Alice Villéger, Sarah E Boyd, Laurence Calzone, Melanie Courtot, Ugur Dogrusoz, Tom C Freeman, Akira Funahashi, Samik Ghosh, Akiya Jouraku, Sohyoung Kim, Fedor Kolpakov, Augustin Luna, Sven Sahle, Esther Schmidt, Steven Watterson, Guanming Wu, Igor Goryanin, Douglas B Kell, Chris Sander, Herbert Sauro, Jacky L Snoep, Kurt Kohn & Hiroaki Kitano. The Systems Biology Graphical Notation. *Nature Biotechnology* **27**, 735 - 741 (2009). http://dx.doi.org/10.1038/nbt.1558

This document defines the *Process Description* visual language of SBGN. Process Descriptions are one of three views of a biological process offered by SBGN. It is the product of many hours of discussion and development by many individuals and groups.

1.1 SBGN levels and versions

It was clear at the outset of SBGN development that it would be impossible to design a perfect and complete notation right from the beginning. Apart from the prescience this would require (which, sadly, none of the authors possess), it also would likely need a vast language that most newcomers would shun as being too complex. Thus, the SBGN community followed an idea used in the development of other standards, i.e. stratify language development into levels.

A *level* of one of the SBGN languages represents a set of features deemed to fit together cohesively, constituting a usable set of functionality that the user community agrees is sufficient for a reasonable set of tasks and goals. Within *levels*, *versions* represent small evolution of a language, that may involve new glyphs, refined semantics, but no fundamental change of the way maps are to be generated and interpreted. In addition new versions should be backwards compatible, i.e., Process Description maps that conform to an earlier version of the Process Description language within the same level should still be valid. This does not apply to a new levels.

Capabilities and features that cannot be agreed upon and are judged insufficiently critical to require inclusion in a given level, are postponed to a higher level or version. In this way, the development of SBGN languages is envisioned to proceed in stages, with each higher levels adding richness compared to the levels below it.

1.2 Developments, discussions, and notifications of updates ₃₇

The SBGN website (http://sbgn.org/) is a portal for all things related to SBGN. It provides a web ₃₈
forum interface to the SBGN discussion list (sbgn-discuss@caltech.edu) and information about ₃₉
how anyone may subscribe to it. The easiest and best way to get involved in SBGN discussions is to ₄₀
join the mailing list and participate. ₄₁

Face-to-face meetings of the SBGN community are announced on the website as well as the mail- ₄₂
ing list. Although no set schedule currently exists for workshops and other meetings, we envision ₄₃
holding at least one public workshop per year. As with other similar efforts, the workshops are likely ₄₄
to be held as satellite workshops of larger conferences, enabling attendees to use their international ₄₅
travel time and money more efficiently. ₄₆

Notifications of updates to the SBGN specification are also broadcast on the mailing list and an- ₄₇
nounced on the SBGN website. ₄₈

Chapter 2

Concepts

To set the stage for what follows, we first give a brief overview of some of the concepts in the Process Description language with the help of an example shown in Figure 2.1.
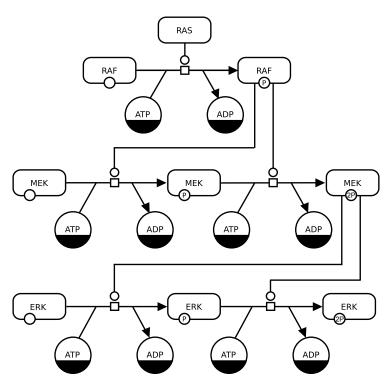
Figure 2.1: *This example of a Process Description uses two kinds of entity pool nodes: one for pools of different macromolecules (Section 4.5.16) and another for pools of simple chemicals (Section 4.5.12). Most macromolecule nodes in this map are adorned with state variables (Section 4.5.29) representing phosphorylation states. This map uses one type of process node, the process node (Section 4.5.25), and three kind of connecting arc, consumption (Section 4.5.34), production (Section 4.5.34) and catalysis (Section 4.5.35). Finally, some entity pool nodes have dark bands along their bottoms; these are clone markers (Section 4.5.38) indicating that the same pool nodes appear multiple times in the map.*

The map in Figure 2.1 is a simple map for part of a mitogen-activated protein kinase (MAPK) cascade. The larger nodes in the figure (some of which are in the shape of rounded rectangles and others in the shape of circles) represent biological materials—things like macromolecules and simple chemicals. The biological materials are altered via processes, which are indicated in Process Description language by lines with arrows and other decorations. In this particular map, all of the processes happen to be the same: processes catalyzed by biochemical entities. The directions of the arrows indicate the direction of the processes; for example, unphosphorylated RAF kinase processes to phosphory-

lated RAF kinase via a process catalyzed by RAS. Although ATP and ADP are shown as incidental to the phosphorylations on this particular graph, they are involved in the same process as the proteins getting phosphorylated. The small circles on the nodes for RAF and other entity pools represent state variables (in this case, phosphorylation sites).

The essence of the Process Descriptions is *change*: it shows how different entities in the system process from one form to another. The entities themselves can be many different things. In the example of Figure 2.1, they are either pools of macromolecules or pools of simple chemicals, but as will become clear later in this chapter, they can be other conceptual and material constructs as well. Note also that we speak of *entity pools* rather than individuals; this is because in biochemical network models, one does not focus on single molecules, but rather collections of molecules of the same kind. The molecules in a given pool are considered indistinguishable from each other. The way in which one type of entity is transformed into another is conveyed by a *process node* and links between entity pool nodes and process nodes indicate an influence by the entities on the processes. In the case of Figure 2.1, those links describe consumption Section 4.5.34, production Section 4.5.34 and catalysis Section 4.5.35, but others are possible. Finally, nodes in Process Descriptions are usually not repeated; if they do need to be repeated, they are marked with *clone markers*—specific modifications to the appearance of the node (Section 4.5.38). The details of this and other aspects of Process Description notation are explained in the following chapters.

2.1 Definitions and Nomenclature

2.1.1 Language versus notation

SBGN specifications propose symbols, ways to organise them, but also semantic rules to analyse the resulting representations. SBGN "drawings" can be translated into English, but also into computer readable formats. Those specifications really propose true languages. SBGN is therefore made up of three languages.

2.1.2 What are the languages?

PD is a language that permits the description of all the processes taking place in a biological system. The ensemble of all these processes constitute a Description. **ER** is a language that permits the description of all the relations involving the entities of a biological system. The ensemble of all these relations constitute a Relationship. **AF** is a language that permits the description of the flow of activity in a biological system.

2.1.3 Nomenclature

The three languages of SBGN should be referred to as:
- the Process Description language.
- the Entity Relationship language.
- the Activity Flow language.

Abbreviated as:
- the PD language.
- the ER language.
- the AF language.

A specific representation of a biological system in one of the SBGN languages should be referred to as:
- a Process Description map.
- an Entity Relationship map.
- an Activity Flow map.

Abbreviated as: 107
- a PD map.   108
- an ER map.   109
- an AF map.   110

The corpus of all SBGN representations should be referred to as: 112
- Process Descriptions.   113
- Entity Relationships.   114
- Activity Flows.   115

The capitalization is important. PD, ER and AF are names of languages. As such they must be capi-   117
talized in English. This is not the case of the accompanying noun (language or map).   118

2.1.4 Graph, diagram or map?   119

A graph is a very technical term that belongs to mathematics and is uncommon in biology. Dia-   120
gram is a concept that encompasses more than just graph. Examples are Venn diagrams for instance.   121
Therefore, we recommend using the term map for SBGN representations. Those representations ef-   122
fectively permit users to travel and orient themselves in a biological system. Map is also the term   123
most frequently used by the different communities, whether in metabolism, signaling or genomics.   124

Chapter 3

A Process Description Primer

SBGN Process Description Level 1 is a visual language. Like any language like English it has a vocabulary, which in the case of SBGN is represented by the symbols that we call glyphs. Again like English our language has grammar, which are the underlying rules and concepts of the language that define its meaning. This specification aims to provide a detailed description of both the vocabulary and grammar rules of the language as a reference. However, to understand this it is essential to grasp what an Process Description map describes.

The first thing to understand is the SBGN-PD does not describe biochemical reactions or gene expression — at least not directly. Instead this language describes collections — pools — of entities that are manipulated by processes which can convert them from one type of entity to another. An entity pool that is the input to a process is said to consumed and that which is an output is produced. There is a third class of entity pool associated with a process: namely those that affect — modulate — the rate at which a process converts entity from one set of pools to another. The amount of entity in a modulating pools is not changed by the process. These very briefly are the key concepts of PD and are illustrated in figure **??**.

So how do this enable us to describe biological processes? Very simply. Figure **??** illustrates a biochemical reaction and shows how the essential components of that reaction, substrate, product and enzyme catalyst are all conveniently described by the representation. Likewise in figure **??** we can see again the concept of entity pools and process flow can be applied to gene regulation. This approach gives us a lot of flexibility and enables one to summarise biological processes that may incorporate many biochemical reactions (figure **??**).

In the above examples of biological processes a number of different glyphs where used to illustrate different biological entities: the macromolecule, the simple chemical and the gene.

Chapter 4

Language Elements

4.1 Introduction

In this chapter we aim to describe the Process Description language by describing its elements and underlying concepts in detail. The challenge in doing this is to provide enough detail to minimise ambiguity, but to also make the rules understandable to the users of the specification. An additional goal is to minimise duplication or rules as much as possible, which makes the specification more difficult to update in the future. To achieve this we took our lead from other successful standards: the Unified Modelling Language (UML) [1] which is a graphical language that has been used by tens of thousands of software developers for over decade to describe and software system; and the Systems Biology Markup Language (SBML) Level 3 Core specification [2] which is the "original" computational systems biological standard and after 10 years is firmly embedded in the scientific community it serves. In both these specifications the language is modelled in UML where each language element is described as an UML class. The specification defines each class in turn and in so doing describes how the language elements fit together (syntax) and how they are applied and their rules (semantics). Since UML is a graphical language, the class definition also includes, where applicable to the language element, a description of description of the symbols used and guidelines about how the symbols should be drawn or laid out.

Based on these specifications we will follow the following conventions in this chapter:

- The language will be described by a UML class model, a language element of language concept being represented as a class.

- Each class will be defined in detail. The definition will describe parent classes, attributes, interactions with other classes and any rules the apply to that class.

- Where classes correspond to glyphs, or where appropriate, a detailed description of the glyph will be given.

- Any rules or concepts that apply to the language as a whole are defined after the individual class definitions.

4.2 Note on typographical convention

The concept represented by a glyph is written using a normal font, while a *glyph* means the SBGN visual representation of the concept. For instance "a biological process is encoded by the SBGN PD *process*". A UML class name is written in camel case and presented as CamelCaseClassName. Attributes and associations are written in lower case, using the underscore to separate words: an_attribute.

Note that in drafts of this document the margin-note[1] is used to indicate that this part of the

⚠
See footnote.

---

[1]The footnote contains a description of why this part of the specification is contentious, may require further discussion, or should be noted by the reviewer of the document.

document may require special attention from the reader as it may contain material that is new or ₁₈₂ potentially contentious. ₁₈₃

4.3 How to read the Language Specification ₁₈₄

Here will describe the elements of the class definitions that are the core of the language specifica- ₁₈₅ tion. Each definition starts with an introduction describing the purpose of the class and what it ₁₈₆ represents conceptually or physically. It's context within the UML Process Description language is ₁₈₇ then described by a detailed figure showing the class, its associations and its interaction with other ₁₈₈ key classes to help understand its context and any rules that are part of the definition. ₁₈₉

A number of terms are used within the specification that for clarity we define here: ₁₉₀

class A class describes a set of objects that share the same specifications of features, constraints, and ₁₉₁ semantics (from the UML specification [1]). ₁₉₂

subclass A class that inherits attributes, behaviour and associations from another class. For example ₁₉₃ in figure 4.1 "class C is a subclass of class A". ₁₉₄

superclass A class that is an ancestor of another class. For example in figure 4.1 "class A is a super- ₁₉₅ class of class B"'. ₁₉₆

generalisation A relationship in UML that defines a subclass/superclass relationship. ₁₉₇

association A semantic relationship between two classes. Typically at least one, but often both ₁₉₈ classes require the other class to complete its definition. ₁₉₉

instance An instance is equivalent to an object — a single example or realisation of a class. ₂₀₀

role The role describes the nature of an association from the perspective of one of the classes in that ₂₀₁ relationship. ₂₀₂

cardinality In the case of an attribute the number of separate values the attribute can hold and for ₂₀₃ an association the number of instances of each class that can be associated with each other. ₂₀₄ The permitted cardinality values are: ₂₀₅

R Required (attributes only). ₂₀₆

O Optional (attributes only). ₂₀₇

1 required - only one instance is permitted (association only). ₂₀₈

0..1 optional - zero or one instance ir permitted (association only). ₂₀₉

1..* at least 1 instance is required. ₂₁₀

* any number of instances are permitted. ₂₁₁

type A type constrains the values represented by an attribute. For example an attribute of type int ₂₁₂ must have a value that is a integer. ₂₁₃

The UML class diagrams used will look something like the examples in figure 4.1. The UML ele- ₂₁₄ ments in this diagram are the only ones used in this specification and again as an aid to understand- ₂₁₅ ing their meanings are described below: ₂₁₆

class See above for definition. The class symbol displays each attribute and its type using the con- ₂₁₇ vention: "attribute : type". ₂₁₈

generalisation (open triangle arrowhead) Defines an inheritance relationship where class C inher- ₂₁₉ its the attributes, associations and semantics of class A. This means that C also has attributes ₂₂₀ attrib1 and attrib2 as well as an association called container to class B. ₂₂₁
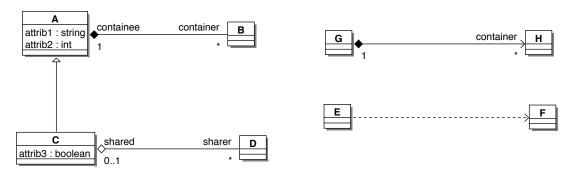
Figure 4.1: *An example of the UML elements used in this specification. The diagram is explained more fully in the text.*

composite aggregation (black diamond) The aggregating class (A) is the one adjacent to the diamond and at the other end is the contained class (B). This represents a whole/part relationship where class B is part of class A and B cannot exist independently of A. The cardinality and role of a given class are shown at the opposite end of the association to it. 222 223 224 225

shared aggregation (open diamond) The aggregating class (C) is adjacent to the diamond and is connected to the other class (D) via a solid line. Here there is a whole/part relationship, but there the part (class D) can be shared with another class and can exists independently of C. 226 227 228

navigability (black diamond anchor arrowhead) If the association ends in an arrow, this indicates the direction of the association. Here this should be taken to mean that only one aside of the association (G) is 'aware' of the relationship. 229 230 231

dependency Indicates that the dependent class (E) requires the other class adjacent to the arrow head (F) to satisfy its specification or implementation. 232 233

The specification uses a number of primitive types that are used in attribute definitions. These are: 234 235

int An integer. 236

string A string of UniCode characters. 237

boolean A Boolean value that can be either True or False. 238

object A type that can be any value. 239

cv A controlled vocabulary (see 4.6). 240

enum A value that must be chosen from on of an enumerated set of predefined values. 241

4.3.1 An example definition 242

Here we will provide an example specification. In this section is an overview of the class and its concepts. 243 244

Generalisation 245

This section defines the inheritance relationship(s) between this class and any other classes. 246

Attributes 247

Here any attributes specific to this class are defined and their meaning or purpose described. Attributes from superclasses are part of this classes definition but are not defined here explicitly. 248 249

Associations   250

Any associations are defined and described. Again associations from superclasses are not included   251
here, but are part of this classes definition.   252

Notation   253

If the class corresponds to one or more glyphs then each glyph is described here. The glyph is de-   254
scribed in words and graphically. In cases where several glyphs combine in complex ways usage   255
examples are provided.   256

Layout Rules and Guidelines   257

In some cases the graphical layout of a glyph and is more complicated and requires some additional   258
explanation. If this is the case this will be provided here.   259

Rules and Constraints   260

The semantics of the class are defined here in the form of itemised rules and constraints on the be-   261
haviour of the class. The scope of this section is ideally restricted to rules and behaviours the related   262
to the class itself or classes it has some form of immediate relationship with (an association or de-   263
pendency). In the later case the rules should relate to that relationship.   264

Changes from Previous Version   265

In order to help track changes between versions of the specification this section documents where   266
this class definition differs from that in previous versions. Where appropriate ticket numbers for bugs   267
or issues addressed in this version should be included.   268

4.4 Overview   269

The UML model describing the Process Description language is summarised in figure 4.2. The model   270
has a root class SBGNElement for all language elements and care has been taken keep the model as   271
simple as possible. This includes minimising the use of multiple inheritance and reflecting the di-   272
rected graph structure inherent in the language with the SBGNNode and SBGNArc classes. All graphi-   273
cal elements that can be drawn directly onto a Process Description map are glyphs (SBGNGlyph) and   274
all those that decorate glyphs are auxiliary units (AuxiliaryUnit). There are a number of language rules   275
that the model and the individual class specification cannot capture and these are dealt with later in   276
the chapter after the class definitions in section 4.5.   277

4.5 Definitions   278

4.5.1 SBGNElement   279

⚠
See footnote.

All the glyphs in SBGN Process Description Level 1 inherit from SBGNElement[2]. This is an abstract   280
or conceptual class that helps organise Process Description conceptually. SBGNElement (figure 4.3)   281
has a single attribute id that is an identifying attribute. This means that all SBGN elements defined   282
here, which ultimately extend SBGNBase, can all be uniquely identified from each other. This makes   283
sense if you think that a glyph drawn on a map is distinct from another glyph drawn on the map. The   284
id attribute reflects this and is not shown explicitly in a Process Description map.   285

Generalisation   286

None   287
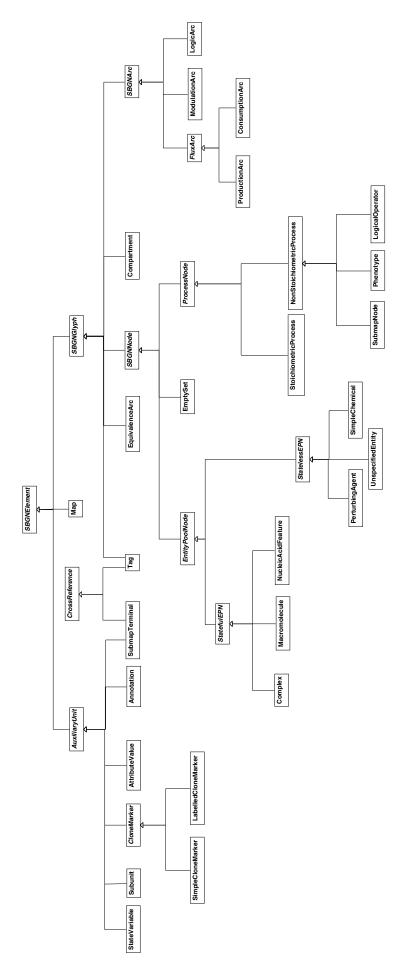
Figure 4.2: *A view of the UML model describing Process Description language. This diagram shows the classes and their inheritance relationships. No attributes or associations between classes are shown to simplify the diagram. These details are provided in the UML class diagrams associated with most class definitions.*

Attributes 288

⚠
See footnote.

id: identifier **(R)** uniquely identifies all SBGN elements in the same namespace[3]. 289

Changes from Previous Version 290

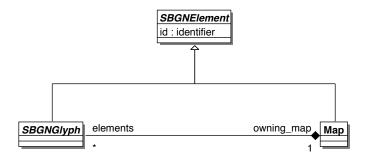Not defined in the previous version. 291

4.5.2 Map 292

Figure 4.3: *UML definition of Map and SBGNElement.*

⚠
See footnote.

The Map[4] (figure 4.3) is a container that holds all the glyphs (SBGNGlyph (see section 4.5.3)) drawn 293
in a Process Description map. A map may represent a submap or a supermap and should comply 294
with the rules set out in section 4.8. The elements held should be logically unique and conform to the 295
identity rules in section 4.7. 296

Generalisation 297

- SBGNElement (see section 4.5.1) 298

Attributes 299

No additional attributes 300

Associations 301

elements:SBGNGlyph (*) The collection of glyphs held by the map. 302

Rules and Constraints 303

- A map is valid if it is empty (although not very useful). 304

- All instances of SBGNGlyph (see section 4.5.3) must be unique (see section 4.7). 305

Notation 306

The map is the canvas upon which the Process Description language is drawn. It's only visible feature 307
is its colour. It can take any pattern or colour (or be transparent for that matter), but as SBGN is 308
'colour blind' this does not convey any meaning in itself. 309

Changes from Previous Version 310

Not defined explicitly in previous versions. 311

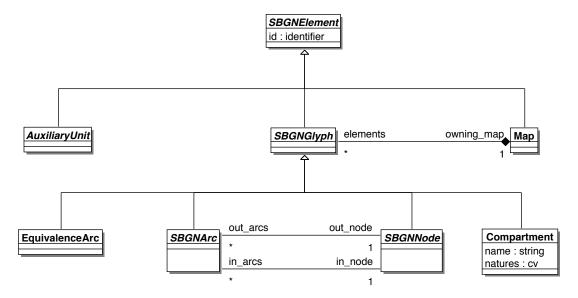**Figure 4.4:** *UML definition of the SBGNGlyph and its subclasses.*

4.5.3 SBGNGlyph   312

The SBGNGlyph is the fundamental building blocks of the Process Description language. It is the only   313
element that can be drawn directly on a map (Map (see section 4.5.2)).   314

Generalisation   315

- SBGNElement (see section 4.5.1)   316

Attributes   317

No additional attributes.   318

Associations   319

owning_map:Map (1) The map that contains this class.   320

Rules and Constraints   321

No additional rules and constraints.   322

Changes from Previous Version   323

Not defined in previous version.   324

4.5.4 AuxiliaryUnit   325

The AuxiliaryUnit (figure 4.5) represents symbols that may be used to adorn glyphs. In doing so they   326
change the meaning of the glyph and/or provide additional information about it.   327

Generalisation   328

- SBGNElement (see section 4.5.1)   329

Attributes   330

No additional attributes.   331

---

[2]A new concept

[3]Not defined previously, but doesn't change Process Description semantics. reinforces the idea of instance identity that exists for all glyphs.

[4]Defining this explicitly is new, but the concept of the map as a container of glyphs has always existed in the Process Description language

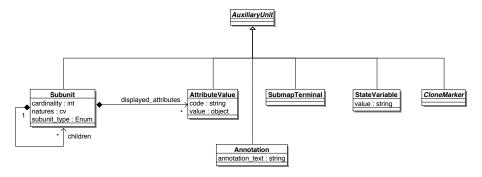**Figure 4.5:** *UML definition of the Auxiliary Unit and its subclasses.*

Associations   332

No additional associations.   333

Rules and Constraints   334

No additional rules and constraints.   335

Changes from Previous Version   336

Not defined in previous version.   337

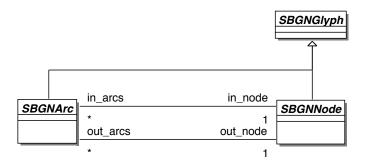### 4.5.5 SBGNNode   338

Figure 4.6: *UML definition of the SBGNNode and SBGNArc classes.*

The SBGNNode (figure 4.6) represents the nodes in the graph structure that is the core represen-   339
tation within the Process Description language. The nodes are connected to glyphs descended from   340
SBGNArc for form a direct graph.   341

Generalisation   342

- SBGNGlyph (see section 4.5.3)   343

Attributes   344

No additional attributes.   345

Associations   346

out_arcs:SBGNArc (*) The arcs that are leaving this node.   347
in_arcs:SBGNArc (*) The arcs that are entering this node.   348

Rules and Constraints   349

- The set of SBGNNodes linked to this node via a SBGNArc (its adjacent nodes) must be all belong   350
  to different entity pools (as defined by EntityPoolNode) and cannot include more than one clone   351
  of the same entity pool.   352

Not defined in the previous version. 354

4.5.6 SBGNArc 355

The SBGNArc (figure 4.6) represents the directed arcs (also know as directed edges) in the directed 356
graph structure that is the core representation within Process Description language. The arc is con- 357
nected to two nodes descended from SBGNNode, one at each end. As the arc has a direction these 358
nodes are by convention designated the out_node to indicate the nodes that the arc is leaving and 359
in_node to indicate the node that it is entering. 360

Generalisation 361

- SBGNGlyph (see section 4.5.3) 362

Attributes 363

No additional attributes. 364

Associations 365

out_node:SBGNNode (1) The node that this arc is leaving. 366
in_node:SBGNNode (1) The node that this arc is entering. 367

Rules and Constraints 368

No additional rules and constraints. 369

Changes from Previous Version 370

Not defined in the previous version. 371
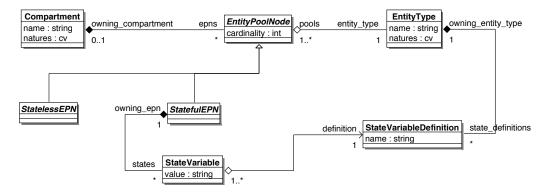
4.5.7 EntityType 372

Figure 4.7: *UML definition of the entity type and the state variable definition. The diagram shows how these classes interact with the entity pool, state variable and so influence EntityPoolNode logical identity.*

⚠️ See footnote.

The EntityType[5] is one of the core concepts in the Process Description language. It defines the type 373
of entity that is instantiated by one or more entity pools in a Process Description map. The EntityType 374
has associated state variable definitions (see figure 4.7) and this enforces one of the core rules in the 375
Process Description language that once a state is associated with an entity type it must be used by all 376
entity pools of that type. 377

---

[5]The concept of an entity pool's type has been there implicitly in previous versions and comes up in discussions. This class aims to formalise that concept and the rules associated with it and enable us to formalise rules associated with EPNs.

Generalisation 378

None. 379

Attributes 380

name: string **(R)** The name that identifies the entity in the Process Description map. EPNs with the 381
same label should be from the same entity. The string cannot be empty and must start and end 382
with a non-space character. Any Unicode character is acceptable[6]. 383

natures: cv(*) The nature of the entity pool node as defined by a controlled vocabulary. Zero, one or 384
more values may be set, but each one must belong to a different controlled vocabulary (see 385
section 4.6)[7]. 386

Associations 387

state_definitions:StateVariableDefintion (*) The state definitions associated with this type. 388
pools:EntityPoolNode (1..*) The entity pool nodes that used this type. 389

Logical Identity 390

Logical Key: 391
- name 392
- natures 393

Rule and Constraints 394

- All instances of EntityPoolNode associated with a particular EntityType must be of the same class. 395

- If an instance of EntityType contains one or more instances of StateVariableDefinition then the 396
EntityPoolNodes associated with it must be subclasses of StatefulEPN. 397

Changes from Previous Version 398

Although not defined explicitly in the previous version, this concept and the associated rules did exist 399
in the language. 400

4.5.8 StateVariableDefinition 401

The StateVariableDefinition[8] defines the state variables used by an EntityType and therefore those state 402
variables that must exists in an EntityPoolNode (see figure 4.7). 403

Generalisation 404

None. 405

Attributes 406

name: string **(O)** The name that of the state variable. This is optional, but if defined cannot be an 407
empty string or just white space characters. It should also start with an alpha-numeric charac- 408
ter and end with a non-space character. It should not contain a '@' character[9]. 409

Associations 410

owning_entity_type:EntityType (1) The EntityType that owns this definition. 411

Rule and Constraints 412

None. 413

In left margin:
⚠ See footnote.
⚠ See footnote.
⚠ See footnote.
⚠ See footnote.

---

[6]Not discussed or defined anywhere, but would make sense to define this explicitly.

[7]This has been discussed on the mailing list where this seems to be the consensus solution

[8]As with EntityType this is new and aims to formalise the concept that an entity pool must preserve the same state variables whenever it is used.

[9]No rule defined previously, but this would seem to make sense.

Changes from Previous Version 414

Although not defined explicitly in the previous version, arguably this concept did exist in the lan- 415
guage. 416

4.5.9 EntityPoolNode 417

An entity pool is a population of entities that cannot be distinguished from each other, when it comes 418
to the SBGN Process Description Level 1 map. For instance all the molecular entities that fulfill the 419
same role in a given process form an entity pool. As a result, an entity pool can represent different 420
granularity levels, such as all the proteins, all the instances of a given protein, only certain forms of a 421
given protein. To belong to a different compartment is sufficient to belong to different entity pools. 422
Calcium ions in the endoplasmic reticulum and calcium ions in the cytosol belong to different entity 423
pools when it comes to representing calcium release from the endoplasmic reticulum. 424
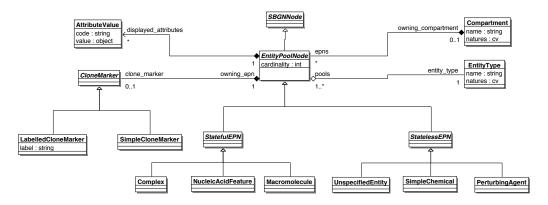
Figure 4.8: *UML definition of the entity pool node and its descendant glyphs.*

The EntityPoolNode (figure 4.8) is the definition of the entity pool and it shares an EntityType with 425
other identical entities[10]. An instance of an entity pools is therefore distinguished from other pools 426
⚠ See footnote. with the same entity type by its cardinality, its owning_compartment and the values of its StateVariables 427
(where appropriate). It must belong to a compartment or be associated with the map (c.f. section 428
4.5.27) and can contain a clone marker if it is cloned (see section 4.7)— note that not all EPNs can be 429
cloned. 430

Generalisation 431

- SBGNNode (see section 4.5.5) 432

Attributes 433

cardinality: int **(R)** The number of copies of the entity. Must be a positive non-zero integer. 434

Associations 435

owning_compartment:Compartment (0..1) The compartment that this EPN belongs too. 436
entity_type:EntityType (1) The type of this entity pool. 437
clone_marker:CloneMarker (0..1) The clone marker decorator. See section 4.5.38 for its use. 438
⚠ See footnote. displayed_attributes:AttributeValue (*) One or more decorators used to display attribute values[11]. 439

Logical Identity 440

Logical Key: 441
- owning_compartment 442

---

[10]Although this concept is discussed it is not explicitly defined previously.

[11]This is an alternate way of using the Unit of Information to display information, but to constrain it so that it presents attributes of the EPN not general annotation. See the AttributeValue class for more information.

- entity_type443
- cardinality444

Rules and Constraints445

- If cardinality > 1 then the descendant glyph must be displayed as a multimer.446

- If the EPN is drawn directly on a *Map* then owning_compartment is not set. We interpret this as447 belonging to an invisible default compartment.448

- natures can only use the material type (section 4.6.1), conceptual type (section 4.6.2) or physical449 characteristics (section 4.6.4) controlled vocabularies.450

- The appropriate subclass of CloneMarker must be used to distinguish logically identical in-451 stances of this class.452

- the EntityPoolNode must be associated with at least one SBGNArc (see section 4.5.6) (degree453 > 0).454

- All StateVariableDefinitions associated with the EntityType must have an associated StateVariable.455

Notation456

Although there is no direct graphical representation of this class the appearance of the AttributeValue457 and its associated glyph the *Unit of Information* us common to all subclasses so it is convenient458 to describe it here. The AttributeValue can be used to present the cardinality and natures of an EPN459 subclasses. These used the following codes to indicate which attribute is being presented:460

pc container physical characteristic461

mt entity pool material type462

ct entity pool conceptual type463

N multimer cardinality464

Changes from Previous Version465

Not defined explicitly in the previous version, but the concept of the EPN and its semantics existed.466 The main change to previous semantics is that of the natures, which didn't formally exist before, but467 which now must contain a unique set of controlled vocabularies and is part of the logical key of the468 EntityPoolNode.469

4.5.10 Empty Set470

It is useful to have the ability to represent the creation of an entity or a state from an unspecified471 source, that is, from something that one does not need or wish to make precise. For instance, in a472 model where the production of a protein is represented, it may not be desirable to represent all of473 the amino acids, sugars and other metabolites used, or the energy involved in the protein's creation.474 Similarly, we may not wish to bother representing the details of the destruction or decomposition of475 some biochemical species into a large number of more primitive entities, preferring instead to simply476 say that the species "disappears into a sink". Yet another example is that one may need to represent477 an input (respectively, output) into (resp. from) a compartment without explicitly representing a478 transport process from a source (resp. to a target).479

For these and other situations, SBGN defines a single glyph to handle these situations represent-480 ing the involvement of an external pool of entities. The symbol used in SBGN is borrowed from the481 mathematical symbol for "empty set", but it is important to note that it does not actually represent a482 true absence of everything or a physical void—it represents the absence of the corresponding struc-483 tures in the model, that is, the fact that the external pool is conceptually outside the scope of the484 map.485

A frequently asked question is, why bother having an explicit symbol at all? The reason is that one cannot simply use an arc that does not terminate on a node, because the dangling end could be mistaken to be pointing to another node in the map. This is specially true if the map is rescaled, causing the spacing of elements in the map to change. The availability and use of an explicit symbol for sources and sinks is critical.
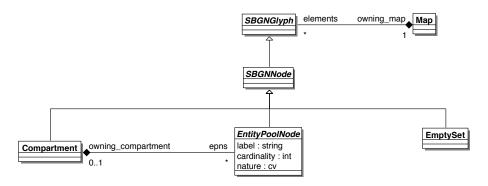

Figure 4.9: *The UML definition of the* EmptySet *and its context in relation to other elements of the Process Description language.*

The definition of the *Empty Set* is shown in figure 4.9. The empty set is not a subclass of EntityPoolNode as it does not represent a single pool of entities and does not share any of the other attributes of an EntityPoolNode, nor does it belong to a particular compartment[12].

⚠
See footnote.

Generalisation

- SBGNNode (see section 4.5.5)

Attributes

No additional attributes.

Associations

No additional associations.

Rules and Constraints

- All instances of *Empty Set* can be regarded as identical therefore not special decoration is used to indicate replication on the map.

- the EmptySet must be associated with at least one SBGNArc (see section 4.5.6) (degree > 0).

Notation

Glyph: *Empty Set*
SBO Term: SBO:0000291 ! empty set
Container: Represented by the mathematical symbol for "empty set", that is, a circle crossed by a bar linking the upper-right and lower-left corners of an invisible square drawn around the circle (∅). Figure 4.10 illustrates this. The symbol should be linked to one and only one edge in a map.
Label: None

---

[12]This is a significant change to the semantics from v1.3 since it is no longer an EPN.

Figure 4.10: *The* empty set *glyph.*

Changes from Previous Version

The EmptySet and *Empty Set* glyph has replaced the *Source* and *Sink* glyphs. This symbols used remains the same, but the underlying concept has changed. The *Source* and *Sink* glyphs where types of EPN, representing single entity pools, while the EmptySet is not.

4.5.11 StatelessEPN

The StatelessEPN (figure 4.11) represents a pool where the entities do not change 'state'. In otherwords the entities do not undergo any physical change that is useful to record in a Process Description map. Therefore they cannot be assigned a state-variable.
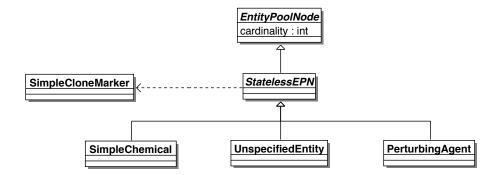
Figure 4.11: *UML definition of the stateless entity pool node and its descendant glyphs.*

Generalisation

- EntityPoolNode (see section 4.5.9)

Attributes

No additional attributes.

Associations

No additional associations.

Rules and Constraints

- if a clone marker is used it must be of type SimpleCloneMarker.

Changes from Previous Version

Not defined in the previous version.

4.5.12 Simple chemical

A SimpleChemical is the 'opposite' of a macromolecule (Section 4.5.16): it is a chemical compound that is *not* formed by the covalent linking of pseudo-identical residues. Examples of simple chemicals are an atom, a monoatomic ion, a salt, a radical, a solid metal, a crystal, etc.

Generalisation 534

- StatelessEPN (see section 4.5.11) 535

Attributes 536

No additional attributes. 537

Associations 538

No additional associations. 539

Rules and Constraints 540

No additional rules and constraints. 541

Notation 542

There are two glyphs associated with SimpleChemical. The first *simple chemical monomer* is used 543
when cardinality = 1 and the second *simple chemical multimer* is used when cardinality > 1. 544

Glyph: *Simple chemical monomer* 545
SBO Term: SBO:0000247 ! simple chemical 546
Container: A *simple chemical* is represented by a 'stadium' symbol: a circle split in two with a rect- 547
angle inserted between them (see figure 4.12). If desired the rectangle can have zero length and 548
the symbol is then identical to a circle (Figure 4.12). To avoid confusion with the Unspecified 549
Entity (4.5.13), this form of the glyph must remain a circle and cannot be deformed into an 550
eclipse. 551
Label: The identification of the *simple chemical* is carried by an unbordered box containing a string 552
of characters. The characters may be distributed on several lines to improve readability, al- 553
though this is not mandatory. The label box has to be attached to the center of the circular 554
container. The label is permitted to spill outside the container. 555

Figure 4.12: *The Process Description glyph for* simple chemical monomer. *The stadium form and
circular forms are shown, as are the cloned forms of the glyph.*

Glyph: *Simple chemical multimer* 556
SBO Term: SBO:0000421 ! multimer of simple chemicals 557
Container: A *simple chemical multimer* is represented by two identical containers shifted horizon- 558
tally and vertically and stacked one on top of the other. Figure 4.13 illustrates the glyph. 559
Label: The multimer carries an identifying label. The label is placed in an unbordered box con- 560
taining a string of characters. The characters can be distributed on several lines to improve 561
readability, although this is not mandatory. The label box must be attached to the center of the 562
top monomer's container. The label may spill outside of the container. 563

Figure 4.13: *The Process Description glyph for* simple chemical multimer. *The figures show the stadium and circular forms, and their cloned variants.*

Changes from Previous Version 564

The glyphs used for the SimpleChemical have been changed to the stadium glyph. Previously the 565
glyph was a circle. To maintain compatibility with previous versions the stadium symbol can be 566
drawn without straight horizontal elements so that it becomes a circle. 567

4.5.13 UnspecifiedEntity 568

The simplest type of EntityPoolNode is the UnspecifiedEntity — one whose type is unknown or simply 569
not relevant to the purposes of the map. This arises, for example, when the existence of the entity has 570
been inferred indirectly, or when the entity is merely a construct introduced for the needs of a map, 571
without direct biological relevance. These are examples of situations where the UnspecifiedEntity is 572
appropriate. (Conversely, for cases where the identity of the entities composing the pool *is* known, 573
there exist other, more specific glyphs described elsewhere in the specification.) 574

Generalisation 575

- StatelessEPN (see section 4.5.11) 576

Attributes 577

No additional attributes. 578

Associations 579

No additional associations. 580

Rules and Constraints 581

- The UnspecifiedEntity cannot have cardinality > 1. This means there is no multimer glyph. 582

- It cannot have a natures[13], therefore natures must **not** be set. 583

⚠
See footnote.

Notation 584

Glyph: *Unspecified entity* 585
SBO Term: SBO:0000285 ! material entity of unspecified nature 586
Container: An *unspecified entity* is represented by an elliptic container, as shown in 4.14. Note that 587
this must remain an ellipse to avoid confusion with the Simple Chemical glyph, which is a circle 588
(c.f. 4.5.12). 589
Label: An *unspecified entity* is identified by a label placed in an unbordered box containing a string of 590
characters. The characters can be distributed on several lines to improve readability, although 591
this is not mandatory. The label box must be attached to the center of the container. The label 592
may spill outside of the container. 593

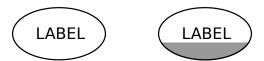[13]Not explicitly discussed, but natures don't make sense here.

Figure 4.14: *The Process Description glyph for* unspecified entity.

Changes from Previous Version 594

No changes from the previous version. 595

4.5.14 Perturbing Agent 596

Biochemical networks can be affected by external influences. Those influences can be the effect of 597 well-defined physical perturbing agents, such as a light pulse or a change in temperature; they can 598 also be more complex and not well-defined phenomena, for instance the outcome of a biological 599 process, an experimental setup, or a mutation. For these situations, SBGN provides the *perturbing* 600 *agent* glyph. It is an EPN, and represents the amount to perturbing agent applied to a process. 601

Generalisation 602

- StatelessEPN (see section 4.5.11) 603

Attributes 604

No additional attributes. 605

Associations 606

No additional attributes. 607

Rules and Constraints 608

- The PerturbingAgent cannot have cardinality > 1. This means there is no multimer glyph. 609

Notation 610

Glyph: *Perturbing agent* 611
SBO Term: SBO:0000405 ! perturbing agent 612
Container: A *perturbing agent* is represented by a modified hexagon having two opposite concave 613
faces, as illustrated in Figure 4.15. 614
Label: A *perturbing agent* is identified by a label placed in an unbordered box containing a string of 615
characters. The characters can be distributed on several lines to improve readability, although 616
this is not mandatory. The label box must be attached to the center of the *perturbing agent* 617
container. The label may spill outside of the container. 618

Figure 4.15: *The Process Description glyph for* perturbing agent.

Changes from Previous Version 619

No changes from pervious version. 620

4.5.15 StatefulEPN 621

Stateful entity pools can undergo physical changes, for example chemical modifcation or conformational change, which we wish to record in a Process Description map. This information is captured via the StateVariable (as can be seen in figure 4.16). The LabellecCloneMarker must be used to indicated that the StatefulEPN is cloned.
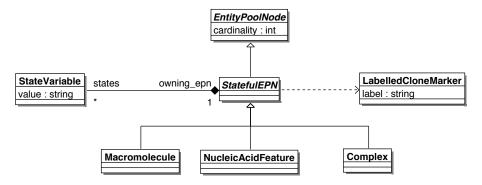
622
623
624
625

Figure 4.16: *UML definition of the stateful entity pool node: showing its descendants and its association with state variables.*

Generalisation 626

- EntityPoolNode (see section 4.5.9) 627

Attributes 628

No additional attributes. 629

Associations 630

states:StateVariable (*) The state variables that belong to this class. 631

Rules and Constraints 632

- State variables do not need to be logically unique, therefore two or more state variables with the same name are permitted. 633
634

- The LabelledCloneMarker must be used to indicate cloning for instances of StatefulEPN and its subclasses, with a must use the same 635
636

Changes from Previous Version 637

Not defined explicitly in the previous version. 638

4.5.16 Macromolecule 639

Many biological processes involve *macromolecules*: biochemical substances that are built up from the covalent linking of pseudo-identical units. Examples of macromolecules include proteins, nucleic acids (RNA, DNA), and polysaccharides (glycogen, cellulose, starch, etc.). Attempting to define a separate glyph for all of these different molecules would lead to an explosion of symbols in SBGN, so instead, SBGN Process Description Level 1 defines only one glyph for all macromolecules. The same glyph is to be used for a protein, a nucleic acid, a complex sugar, and so on. The exact nature of a particular macromolecule in a map is then clarified using its label and decorations, as will become clear below.

640
641
642
643
644
645
646
647

Generalisation 648

- StatefulEPN (see section 4.5.15) 649

Attributes 650

No additional attributes. 651

Associations 652

No additional associations. 653

Rules and Constraints 654

No additional rules and constraints. 655

Notation 656

There are two glyphs associated with Macromolecule. The first *Macromolecule monomer* is used when 657
cardinality = 1 and the second *Macromolecule multimer* is used when cardinality > 1. 658

Glyph: *Macromolecule monomer* 659
SBO Term: SBO:0000245 ! macromolecule 660
Container: A macromolecule is represented by a rectangular container with rounded corners, as 661
illustrated in Figure 4.17. 662
Label: A *macromolecule* is identified by a label placed in an unbordered box containing a string of 663
characters. The characters can be distributed on several lines to improve readability, although 664
this is not mandatory. The label box must be attached to the center of the container. The label 665
may spill outside of the container. 666

Figure 4.17: *The Process Description glyph for* macromolecule, *shown plain and unadorned on the left, and with an additional state variable and a unit of information in the middle and the cloned form on the right.*

Glyph: *Macromolecule multimer* 667
SBO Term: SBO:0000420 ! multimer of macromolecules 668
Container: A *multimer* is represented by two identical containers shifted horizontally and vertically 669
and stacked one on top of the other. Figure 4.18 illustrates the glyph. 670
Label: As monomer 671

Figure 4.18: *The Process Description glyph for* macromolecule multimer, *shown plain and unadorned on the left, and with an additional state variable and a unit of information in the right and the cloned form on the right.*

Usage Examples In this section, we provide examples of Entity Pool Node representations drawn 672
using the SBGN Process Description Level 1 glyphs described above. 673
Figure 4.19 represents calcium/calmodulin kinase II, with phosphorylation on the sites threonine 674
286 and 306, as well as catalytic and autoinhibitory domains. Note the use of *units of information* and 675
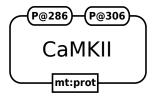*state variables.* 676

Figure 4.19: *An example representation of calcium/calmodulin kinase II.*

Figure 4.20 represents the glutamate receptor in the open state, with both phosphorylation and glycosylation. The entity carries two functional domains, the ligand-binding domain and the ion pore, and its chemical nature is precided.
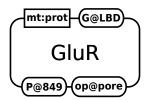

Figure 4.20: *An example of a glutamate receptor in the open state.*

Changes from Previous Version

No changes from the previous version.

4.5.17 NucleicAcidFeature

The NucleicAcidFeature represents a fragment of a macromolecule carrying genetic information. A common use for this construct is to represent a gene or transcript. The label of this EPN and its natures are often important for making the purpose clear to the reader of a map.

Generalisation

- StatefulEPN (see section 4.5.15)

Attributes

No additional attributes.

Associations

No additional associations.

Rules and Constraints

No additional rules and constraints.

Notation

The NucleicAcidFeature has two associated glyphs. The first *Nucleic acid feature monomer* is used when cardinality = 1 and the second, *Nucleic acid feature multimer* is used when cardinality > 1.

Glyph: *Nucleic acid feature monomer* This glyphs represents a monomeric macromolecule.
SBO Term: SBO:0000354 ! informational molecule segment
Container: A *nucleic acid feature* is represented by a rectangular container whose bottom half has rounded corners, as shown in Figure 4.21.

Label: The identity of a particular *Nucleic acid feature* is established by a label placed in an un- 701
ordered box containing a string of characters. The characters may be distributed on several 702
lines to improve readability, although this is not mandatory. The label box must be attached to 703
the center of the container. The label may spill outside of the container. 704

Figure 4.21: *The Process Description glyph for* nucleic acid feature monomer, *shown plain and unadorned on the left and with an additional state variable and a unit of information in the middle and the cloned form on the right.*

Glyph: *Nucleic acid feature multimer* This glyphs represents a multimeric macromolecule. 705
SBO Term: SBO:0000419 ! multimer of informational molecule segments 706
Container: A *Nucleic acid feature multimer* is represented by two identical containers shifted hori- 707
zontally and vertically and stacked one on top of the other. Figure 4.22 illustrates the glyph. 708
Label: As monomer glyph. 709
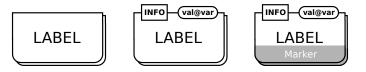
Figure 4.22: *The Process Description glyph for* nucleic acid feature multimer, *shown plain and unadorned on the left and with an additional state variable and a unit of information in the middle and the cloned form on the right.*

Changes from Previous Version 710

No changes from the previous version. 711

4.5.18 Complex 712

A Complex represents a biochemical entity composed of other biochemical entities, whether macro- 713
molecules, simple chemicals, multimers, or other complexes (figure 4.23). The Complex can de- 714
scribed its composition by the set of Subunits it contains (see figure 4.5.19). This description is entirely 715
optional and is their to assist the user with a visual shorthand about the composition of the complex. 716
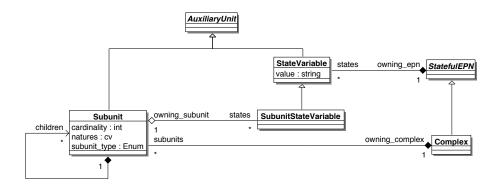

Figure 4.23: *The UML definition of the Complex and its associated subunits. In particular this describes organisation of the state variables that belong to both the subunit, but also the complex.*

Generalisations 717

- EntityPoolNode (see section 4.5.9) 718

Attributes 719

No additional attributes 720

Associations 721

subunits:Subunit (*) The subunits that describe the composition of this complex. 722

Special Rules and Constraints 723

⚠ See footnote.

- Once a set of subunits are defined for an Complex with a given EntityType, then they must be 724 used by every instance using that entity type.[14]. 725

- The set of subunits in the Complex does not identify it. One or more Complexes that contain 726 the same set of subunits, but have different labels are **not** identical. 727

Notation 728

The Complex is represented by two glyphs, the *Complex Monomer* which represents a Complex where 729 the cardinality is one and the *Complex Multimer* where the cardinality is greater than that. 730

Complex Monomer 731

SBO Term: SBO:0000253 ! non-covalent complex 732

Container: A *complex* possesses its own container box surrounding the juxtaposed container boxes 733 of its components. This container box is a rectangle with cut-corners (an octagonal box with 734 sides of two different lengths). The size of the cut-corners are adjusted so that there is no 735 overlap between the container and the components. The container boxes of the components 736 must not overlap. 737

Label: The identification of a *named complex* is carried by an unbordered box containing a string of 738 characters. The characters may be distributed on several lines to improve readability, although 739 this is not mandatory. Ideally the label box should be attached to the midway between the 740 border of the complex's container box and the border of the components' container boxes. 741 However, if the Complex contains Subunit glyphs then the label may be positions to optimise 742 the clarity and avoid overlapping. 743
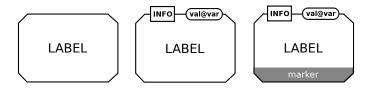

Figure 4.24: *The* complex *glyph.*

Complex Multimer 744

SBO Term: SBO:0000418 ! multimer of complexes 745

Container: A *Complex Multimer* is represented by two identical *Complex* containers shifted hori- 746 zontally and vertically and stacked one on top of the other. Figure 4.25 illustrates the glyph. 747

Label: As monomer 748
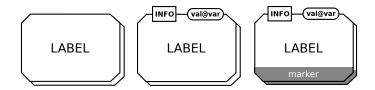
---

[14]New rule.

Figure 4.25: *The* Complex Multimer *glyph.*

Layout Rules and Guidelines   749

- The subunits inside the complex must not overlap.   750

- The subunits should sit above the clone marker so that they are not obscured by it.   751

- The label should not be obscured by subunits or obscure them.   752

Changes from Previous Version   753

- Clarified that complex must have a label and the label identifies the complex irrespective of its   754
  subunit composition.   755

- The label positioning does not need to be at the centre of the Complex glyph.   756

4.5.19 Subunit   757

⚠
See footnote.

The Subunit[15] is used to describe the composition of the Complex (see section 4.5.18). A complex can   758
optionally be decorated with one or more subunits, which represent the types of EntityPoolNode (see   759
section 4.5.9) that may aggregate to form a complex. As we can see from the UML representation   760
(figure 4.23) the Subunit is an auxiliary unit that decorates the Complex and does not represent an   761
entity pool directly. In addition it does not mimic the EntityPoolNode class hierarchy (Subunit, but   762
rather uses the subunit_type attribute to indicate the type of subunit.   763

Generalisation   764

- EntityPoolNode (see section 4.5.9)   765

Attributes   766

cardinality: int **(R)** The number of copies of the subunit.   767
name: string **(O)** The name of the subunit.   768
subunit_type: enum **(R)** The type of the subunit. It can have one of the following values that corre-   769
    spond to the equivalent EPN class: SimpleChemical, UnspecifiedEntity, PerturbingAgent, Macro-   770
    molecule, NucleicAcidFeature, Complex.   771

Associations   772

owning_complex:Complex (1) The complex that owns the subunit.   773
states:SubunitStateVariable (*) The state variables assigned to this subunit.   774
children:Subunit (*) Subunits that are contained by this subunit.   775

Rules and Constraints   776

- Two or more state variables with the same name are permitted.   777

- State variables with no name set are permitted.   778

- Subunits can also contain subunits. There is no limit on such nesting. The namespace rules   779
  below apply.   780

---

[15]The principles of the subunit have been agreed, but the details of this implementation should be reviewed. In particular
the rules about state variables.

- The subunit defines a namespace for its state variables, e.g. subunit "A" assigned a state variable "P@Ser202" and a subunit "B" assigned the same state variable can be distinguised as A:P@Ser202 and B:P@Ser202. 781
782
783

- If the subunit is of type Complex then children can contain one or more Subunit instances. 784

- If the subunit has a cardinality > 1 then this should be displayed by the AttributeValue (see section 4.5.28). 785
786

- If natures contains one or more instances then these must be displayed via an AttributeValue. 787

Notation 788

The subunit symbol used for the *subunit* glyph varies depending on the subunit_type and cardinality. The symbols available are equivalent to those used by the EPN glyphs including the *complex*. Therefore it is possible to describe complexes within complexes. The mapping between these and the symbol used is shown table 4.1. Not that subunits may contain labels corresponding to their name. 789
790
791
792

Table 4.1: *Mapping between the subunit_type, cardinality values of Subunit and the glyphs used to represent it. These are essentially the EPN glyphs described in this document.*

subunit_type	cardinality = 0	cardinality > 0
SimpleChemical	*Simple Chemical Monomer*	*Simple Chemical Multimer*
UnspecifiedEntity	*Unspecified Entity*	None
PerturbingAgent	*Perturbing Agent*	None
Macromolecule	*Macromolecule Monomer*	*Macromolecule Multimer*
NucleicAcidFeature	*Nucleic Acid Feature Monomer*	*Nucleic Acid Feature Multimer*
Complex	*Complex Monomer*	*Complex Multimer*

793

The example in figure 4.26 illustrates the use of subunits in a complex. It also shows an equivalent compex without subunits. This is an import point. For every *Complex* drawn with subunits it will always be possible to drawn an equivalent version that does not use contains subunits[16]. 794
795
796
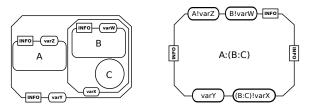
⚠️
See footnote.

Figure 4.26: *Both these complex glyphs are equivalent. The one on the left is described using subunit decorators, the one on the right describes the same thing without them.*

Changes from Previous Version 797

In previous version of the spec the subunits of a Complex were regarded as an EPN. This however, is incorrect as it implies there are pools within pools, which breaks one of the fundamental paradigms of the Process Description language. This is corrected in the current version and subunits are now adornments of the Complex. 798
799
800
801

4.5.20 ProcessNode 802

The Process (figure 4.27) represents a process that transforms one or more entity pools into one or more entity pools, that are identical or different. A process may be used to represent or summarise more than one known process. 803
804
805

---

[16]Not discussed in detail. This must be true if states can be drawn on subunits, but actually belong to the complex. Either this or we enforce a rule that all state vars must be named uniquely.
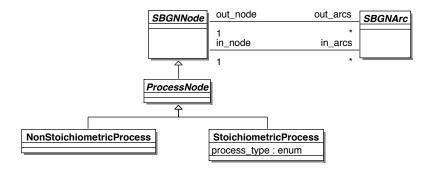
Figure 4.27: *The UML definition of the Process and its associated subclasses. Note that the Process extends SBGNNode so all its descendants can potentially be nodes in a directed graph.*

Generalisation 806

- SBGNNode (see section 4.5.5) 807

Attributes 808

No additional attributes. 809

Associations 810

No additional associations. 811

Rules and Constraints 812

No additional rules and constraints. 813

Changes from Previous Version 814

- This was not explicitly defined in the previous version, but this version did define a glyph called 815 *Process*. To avoid ambiguity this glyph has now been renamed *Stoichiometric Process* (see sec- 816 tion 4.5.25). 817

- Previous specifications stated that processed could be duplicated when all associated EPNs 818 were cloned. This behaviour has been changed the current status where all processes are 819 unique in a Process Description map. 820

4.5.21 NonStoichiometricProcess 821

⚠ See footnote.

The NonStoichiometricProcess[17] (figure 4.28) is a type of process. It does not necessarily result in a 822 measurable change of entity pools, nor does it necessarily have a defined start and end point. In 823 many cases the process is not well defined. This may because it is not well understood or because 824 the detail is not important or is being summarised. 825
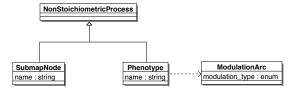
Figure 4.28: *The UML definition of the NonStoichiometricProcess and its associated subclasses.*

Generalisation 826

- ProcessNode (see section 4.5.20) 827

---

[17]This has been discussed and agreed in past meetings

Attributes 828

No additional attributes. 829

Associations 830

No additional associations. 831

Rules and Constraints 832

No additional rules and constraints. 833

Changes from Previous Version 834

Not defined in the previous version. 835

4.5.22 Phenotype 836

A biochemical network can generate phenotypes or affect biological processes. Such processes can 837
take place at different levels and are independent of the biochemical network itself. To represent 838
these processes in a map, SBGN defines the Phenotype (figure 4.28). 839

Generalisation 840

- NonStoichiometricProcess (see section 4.5.21) 841

Attributes 842

name: string **(R)** The name of the phenotype. 843

Associations 844

No additional associations. 845

Logical Identity 846

Logical Key: 847
- owning_map 848
- name 849

Rules and Constraints 850

- The number of in_arcs must be > 0. 851

- in_arc can only contain instances of ModulationArc (see section 4.5.35) and its subclasses. 852

- out_arcs must be empty. 853

Notation 854

Glyph: *Phenotype* 855
SBO Term: SBO:0000358 ! phenotype 856
Container: A *phenotype* is represented by an elongated hexagon, as illustrated in Figure 4.29. 857
Label: A *phenotype* is identified by a label placed in an unbordered box containing a string of char- 858
acters. The characters can be distributed on several lines to improve readability, although this 859
is not mandatory. The label box must be attached to the center of the *phenotype* container. The 860
label may spill outside of the container. 861

Figure 4.29: *The Process Description glyph for* phenotype.

Changes from Previous Version 862

This definition clarifies that the Phenotype cannot be cloned as it is now a subclass of Process, which 863
is always unique. 864

4.5.23 SubmapNode 865

⚠
See footnote.
The SubmapNode[18] (figure 4.30) is a placeholder for another process and is used when one wishes to 866
hide the detail of this process from the Process Description map, but make it available to the reader 867
as a separate related map. The Submap is not equivalent to an OmittedProcess (section 4.5.25). The 868
Submap allows the detail of section of the Process Description map to be exported to another Process 869
Description map and replaced by the SubmapNode, which acts as a place-holder. This is described in 870
section 4.5.2 and the semantics of submap linking is defined in section 4.8. 871
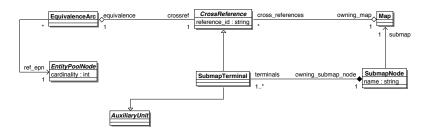
Figure 4.30: *The UML definition of the SubmapNode and its relationship to its submap, tags etc.*

Generalisation 872

- NonStoichiometricProcess (see section 4.5.21) 873

Attributes 874

name: string **(R)** The name of the submap that is being summarised. Note that this name ideally will 875
indicate the function or the processes that are being summarised. 876

Associations 877

terminals:SubmapTerminal (1..*) The terminals provide a reference between the EPNs in the Main Map 878
and those in the submap, which are identified by a Tag. 879

Logical Identity 880

Logical Key: 881
- owning_map 882
- name 883

Rules and Constraints 884

- All instances of SubmapTerminal (see section 4.5.32) held by this class must be logically unique. 885

- attribin_arcs and out_arcs must be empty (i.e.,, degree = 0). 886

Notation 887

Glyph: *Submap Node* 888
SBO Term: SBO:0000395 ! encapsulating process 889
Container: The *submap* is represented as a square box to remind the viewer that it is fundamentally 890
a process. 891

---

[18]This name change has not been discussed at the time of writing. The aim is to provide clarity between the submap and
this glyph.

Label: The identification of the *submap* is carried by an unbordered box containing a string of char- 892
acters. The characters may be distributed on several lines to improve readability, although this 893
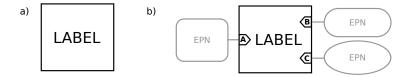is not mandatory. The label box has to be attached to the center of the container box. 894

Figure 4.31: *The Process Description glyph for* submap. *(a) the basic glyph symbol, without the* submap terminal *auxiliary units that would normally be associated with it. (b) The glyph as it would typically be used within a map — associated with EPN glyphs and containing* submap *terminals.*

Changes from Previous Version   895

This glyph was called *Submap* in previous version of the Process Description specification. This is 896
confusing when talking about the Submap itself so this glyph is now referred to as the SubmapNode 897
to distinguish it. 898

4.5.24 LogicalOperator   899
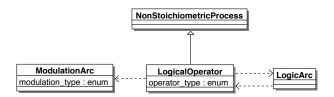
Figure 4.32: *The UML definition of the LogicalOperator.*

   The LogicalOperator (figure 4.32 performs a Boolean operation on one or more inputs to give a 900
binary output. The input must be a Boolean value, and are obtained from the LogicArc (see section 901
4.5.36) connected to the LogicOperator. The output a two-value quantity, 0 for False and positive non- 902
zero for True. This is required because the output of the LogicOperator must be connected to either 903
a LogicArc or a ModulationArc (see section 4.5.35) both of which require their out node to provide a 904
quality. The behaviour of the logical operator for each type of operator_type is shown in the following 905
table: 906
                                                                                      907

AND	All inputs must be True for output to be True, otherwise output is false.
OR	At least one input must be True for output to be True. If all inputs are False then output is False.
NOT	Only one input is permitted and the output is the inversion of the input. Therefore True gives False and False gives True.

Generalisation   908

- NonStoichiometricProcess (see section 4.5.21)   909

Attributes   910

operator_type: enum **(R)** The operator type must be one of the following enumerations: AND, OR, 911
NOT. 912

Associations 913

No additional associations. 914

Rules and Constraints 915

- in_arc can only contain one or more instances of LogicArc. 916

- out_arc can only contain one or more instances of LogicArc or ModulationArc. 917

- if operator_type is AND or OR, then in_arc must contain two or more arcs. 918

- if operator_type is NOT then in_arc must contain only one arc. 919

- out_arc can contain only one arc. 920

Notation 921

Three glyphs are used to represent the different operator types. The glyphs are names after the cor- 922
responding type. 923

Glyph: *And* 924
SBO Term: SBO:0000173 ! and. 925
Node: *And* is represented by a circle carrying the word "AND". 926

Figure 4.33: *The Process Description glyph for* and. *Only two inputs are represented, but more would be allowed.*

Glyph: *Or* 927
SBO Term: SBO:0000174 ! or. 928
Node: *Or* is represented by a circle carrying the word "OR". 929
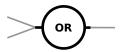
Figure 4.34: *The Process Description glyph for* or. *Only two inputs are represented, but more would be allowed.*

Glyph: *Not* 930
SBO Term: SBO:0000238 ! not. 931
Node: *Not* is represented by a circle carrying the word "NOT". 932

Figure 4.35: *The Process Description glyph for* not.

Changes from Previous Version 933

Although the LogicOperator was not explicitly defined in the previous version the semantics and glyphs 934
are unchanged. 935

4.5.25 StoichiometricProcess 936
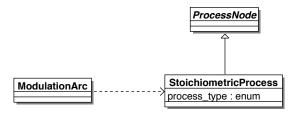
Figure 4.36: *The UML definition of the StiochiometricProcess. The class interacts with subclasses of FluxArc and ModulationArc.*

⚠
See footnote.
A stoichiometric process[19] produces a measurable change in the quantities of entity pools con- 937
sumed and produced. This might imply modification of covalent bonds (conversion), modification 938
of the relative position of constituents (conformational process) or movement from one compart- 939
ment to another (translocation). Such a process will have a basal rate at which this change occurs, 940
which can be affected positively or negatively by the other entity pools, which 'modulate' the pro- 941
cess. Examples of this include stimulation, inhibition and catalysis. In an irreversible process the 942
entity pools interacting with it can be grouped into inputs and outputs. However, a stoichiometric 943
process can also be reversible and so for convenience we refer to these groupings as the "left-hand- 944
side" (LHS) and "right-hand-side" (RHS) of the process[20] (figure 4.37). 945
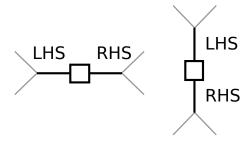
Figure 4.37: *An illustration of the "sidedness" of a process. The designation of LHS and RHS is essentially arbitrary.*

In the Process Description language this is represented by the StoichiometricProcess (figure 4.36). 946
It can be one of several different types, which indicate the amount that is known about the process 947
or in some cases the nature of the process, for example association and dissociation. The permitted 948
values for process_type are described in the following table: 949

---

[19]New concept, but discussed in previous meetings. The semantics of the process being stoichiometrically balanced has not been discussed in detail for the stoichiometric process, and this is the subject of a tracker query. The spec previously stated that the process should be balanced and this is therefore consistent with that.

[20]Note this designation is purely for grouping and is used even then the sides of the reaction are above and below the process.

generic	A generic stoichiometric process that transforms a set of entity pools into another set of entity pools.
omitted	Omitted processes are processes that are known to exist, but are omitted from the map for the sake of clarity or parsimony. A single *omitted process* can represent any number of actual processes. The *omitted process* is different from a *submap*. While a *submap* references to an explicit content, that is hidden in the main map, the *omitted process* does not "hide" anything within the context of the map, and cannot be "unfolded".
uncertain	Uncertain processes are processes that may not exist. A single *uncertain process* can represent any number of actual processes.
association	The association between one or more *EPNs* represents the non-covalent binding of the biological objects represented by those *EPNs* into a larger complex.
dissociation	The dissociation of an *EPN* into one or more *EPNs* represents the rupture of a non-covalent binding between the biological entities represented by those *EPNs*.

Since this process is stoichiometric the relative quantities of the entity pools participating the process must be specified. For this reason the FluxArc (see section 4.5.34) has an stoichiometry attribute and each EntityPoolNode (see section 4.5.9) has a cardinality, which should be balanced in a valid Process Description map. This is especially important where there is potential ambiguity in the stoichiometry of the process (figure 4.38).
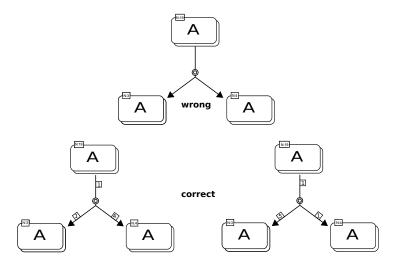
Figure 4.38: *The figure illustrates why for the stoichiometry label is required to clarify potentially ambiguous stoichiometry. In the top example there is more than one possible solution, which can only be made clear using the stoichiometry labels in the bottom examples.*

A stoichiometric process is deemed to be reversible its in_arcs are FluxArcs of type 'reversible' (see figure 4.39). Semantically, this permits a reversible flow of entities through the process. Modulation of a reversible process affects the rate of flux through the process, but does not directly affect the direction of that flow.
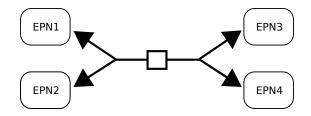
Figure 4.39: *A valid reversible process. A process is reversible if its LHS and RHS contain only* production *arcs.*

Generalisation 960

- ProcessNode (see section 4.5.20) 961

Attributes 962

process_type: enum **(R)** This must be one of the following enumerations: generic, omitted, uncertain, 963 association, dissociation. 964

Associations 965

No additional associations. 966

Rules and Constraints 967

General 968

- The in_arc must contain one or more FluxArcs containing the same flux_type value. 969

- The in_arc may only contains FluxArc instances with a flux_type of 'consumption', or 'reversible'. 970

- In addition the in_arc may contain zero, one or more instances of ModulationArc. 971

- The out_arc must contain one or more instances of FluxArc with a flux_type or 'production'. 972

- If in_arcs contains one or more FluxArcs of type 'reversible' this process reversible. 973

- The EntityPoolNodes that make up the LHS of the process should be consistent with the RHS, 974 i.e. the process should be stoichiometrically balanced.[21] 975

⚠ See footnote.

- If at least one FluxArc associated with a StoichiometricProcess displays its stoichiometry via a 976 *stoichiometry label* then all must.[22] 977

⚠ See footnote.

- If more than one set of stoichiometries can be applied to the flux arcs of the process then the 978 stoichiometry of the flux arcs must be displayed. 979

Association These rules apply if the process_type is 'association'. 980

- The process must be irreversible. 981

- There can only be one 'production' FluxArc, with stoichiometry = 1. 982

- If a Complex is on the RHS of the association then there must be at least 2 EPNS on the LHS.*Is* 983 *this too restrictive? It prevents multimers being represented as a complex of 2 identical subunits.* 984 *It is taken from v1.0 of the spec and got lost in later versions.* 985

---

[21]Tracker issue 329060. If the process is stoichiometric this must make sense. The previous spec states this so this is consistent with it.

[22]Take from previous spec, but that said if one displays stoichiometry in a map which is too restrictive.

Dissociation   These rules apply if the process_type is 'dissociation'. 986

- The process must be irreversible. 987

- There can only be one 'consumption FluxArc, with stoichiometry = 1. 988

- If a Complex is on the LHS of the dissociation then there must be at least 2 EPNS on the RHS. *see* 989
  comment in association rules. 990

Notation 991

Glyph: *Process* 992

SBO Term: SBO:0000375 ! process 993

Node: A process is represented by a square box linked to two connectors: small arcs attached to the 994
⚠ the centers of opposite sides and referred to here as 'lugs'[23]. The flux arcs arcs are linked to the ends 995
See footnote. of the lugs as shown in figure 4.40. The lug's purpose is to 'gather' the flux arcs together before 996
meeting the process node proper and in doing so they emphasis the 'sides' of the reaction. 997
⚠ Therefore the lug must have a visually appreciable length[24] and must be placed on opposite 998
See footnote. sides of the process square. The modulatory arcs (section 4.5.35) point to the other two sides 999
of the box. 1000
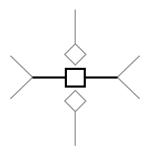
Figure 4.40: *The Process Description glyph for* process.

The example in Figure 4.41 illustrates the use of a *process* node to represent the phosphorylation 1001
of a protein in a Process Description. 1002

Figure 4.41: *Phosphorylation of the protein MAP kinase.*

The example in Figure 4.42 illustrates the use of a *process* node to represent a reaction between 1003
two reactants that generates three products. 1004

---

[23]The term lugs is used in discussion. We haven't discussed this in detail or at least come to a consensus on it. In
particular does the lug need to be perpendicular to the process and does it need to be a straight line? How should it be
used when the arc connecting to it is curved.

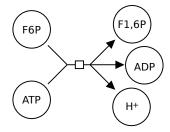[24]Undefined previously, but if we define it then it should be visible.

Figure 4.42: *Reaction between ATP and fructose-6-phosphate to produce fructose-1,6-biphosphate, ADP and a proton.*

The example in Figure 4.43 illustrates the use of a *process* node to represent a translocation. The 1005
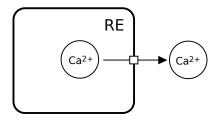large round-cornered rectangle represents a compartment border (see Section 4.5.27). 1006

Figure 4.43: *Translocation of calcium ion out of the endoplasmic reticulum. Note that the* process *does not have to be located on the boundary of the* compartment. *A* process *is not attached to any* compartment.

The example in Figure 4.44 illustrates the use of a *process* node to represent the reversible opening 1007
and closing of an ionic channel in a Process Description. 1008
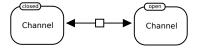
Figure 4.44: *Reversible opening and closing of an ionic channel.*

When such a reversible process is asymmetrically modulated, it must be represented by two dif- 1009
ferent processes in a Process Description. Figure 4.45 illustrates the use of two *process* nodes to 1010
represent the reversible activation of a G-protein coupled receptor. In the absence of any effector, 1011
an equilibrium exists between the inactive and active forms. The agonist stabilises the active form, 1012
while the inverse agonist stabilises the inactive form. 1013
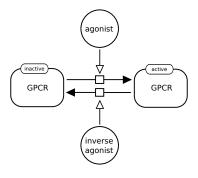
Figure 4.45: *The reversible activation of a G-protein coupled receptor.*

The example in Figure 4.46 presents the conversion of two galactoses into a lactose. Galactoses 1014
are represented by only one *simple chemical,* the cardinality being carried by the *consumption* arc. 1015
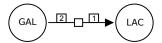

Figure 4.46: *Conversion of two galactoses into a lactose.*

Glyph: *Omitted process* 1016
SBO Term: SBO:0000397 - omitted process. 1017
Node: An *omitted process* is represented by a *process* in which the square box contains a two parallel 1018
   slanted lines oriented northwest-to-southeast and separated by an empty space. 1019
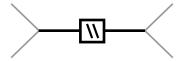
Figure 4.47: *The Process Description glyph for* omitted process.

Glyph: *Uncertain process* 1020
SBO Term: SBO:0000396 ! uncertain process. 1021
Node: An *uncertain process* is represented by a *process* which square box contains a question mark. 1022
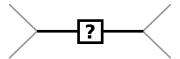
Figure 4.48: *The Process Description glyph for an* uncertain process.

Glyph: *Association* 1023
SBO Term: SBO:0000177 ! non-covalent binding. 1024
Node: An *association* between several entities is represented by a filled disc linked to two connectors, 1025
   small arcs attached on point separated by 180 degrees. The consumption (Section 4.5.34) and 1026
   production (Section 4.5.34) arcs are linked to the extremities of those connectors. 1027
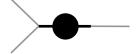

Figure 4.49: *The Process Description glyph for* association.

The example in Figure 4.50 illustrates the association of cyclin and CDC2 kinase into the Matura- 1028
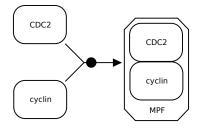tion Promoting Factor. 1029

Figure 4.50: *Association of cyclin and CDC2 kinase into the Maturation Promoting Factor.*

Figure 4.51 gives an example illustrating the association of a pentameric macromolecule (a nicotinic acetylcholine receptor) with a simple chemical (the local anesthetic chlorpromazin) in an unnamed complex.
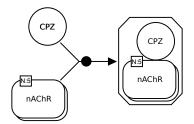

Figure 4.51: *The association of a pentameric macromolecule with a simple chemical in an unnamed complex.*

Glyph: *Dissociation*
SBO Term: SBO:0000180 ! dissociation.
Node: A *dissociation* between several entities is represented by two concentric circles. A simple empty disc could be, in some cases, confused with the *catalysis* (section Section 4.5.35). Moreover, the existence of two circles reminds the dissociation, by contrast with the filled disc of the *association* (Section 4.5.25).
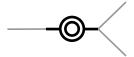

Figure 4.52: *The Process Description glyph for* dissociation.

The example in Figure 4.53 illustrates the dissociation of the small and large ribosomal subunits from a messenger RNA.
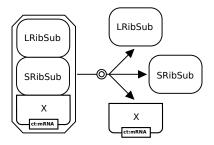

Figure 4.53: *Dissociation of the small and large ribosomal subunits from a messenger RNA.*

Changes from Previous Version 1041

Although the NonStoichiometricProcess was not explicitly defined in the previous version the seman- 1042
tics and glyphs are unchanged. 1043

4.5.26 Compartment 1044

Figure 4.54: *The UML definition of the Compartment showing how it containment of EntityPoolN-ode.*

The Compartment is a logical or physical structure that contains entity pool nodes. An EntityPoolN- 1045
ode (see section 4.5.9) can only belong to one compartment. Therefore, the "same" biochemical 1046
species located in two different compartments are in fact two different pools. 1047

Generalisation 1048

- SBGNGlyph (see section 4.5.3) 1049

Attributes 1050

name: string **(R)** The name of the compartment. 1051

natures: cv(*) A set of controlled vocabularies[25] that describes a characteristic of the compartment. 1052
⚠ Zero, one or more values may be set, but each one must belong to a different controlled vocab- 1053
See footnote. ulary. 1054

Associations 1055

epns:EntityPoolNode (*) The EntityPoolNodes contained by this compartment. 1056

4.5.27 Logical Identity 1057

Logical Key: 1058
- owning_map 1059
- name 1060

Rules and Constraints 1061

- name must not be used by another instance of Container contained by the same instance of Map. 1062

- epns must contain a unique set of EntityPoolNodes. See section 4.7 for the definition of Entity- 1063
  PoolNode uniqueness. 1064

Notation 1065

Glyph: *Compartment* 1066
SBO Term: SBO:0000290 ! physical compartment 1067

---

[25]This reconciles the use of the Unit of Information to represent the nature of an EPN by using to present similar infor-mation for the compartment. This is consistent with previous usage, but not with the usage of the UofI for annotation.

Container: A compartment is represented by a surface enclosed in a continuous border or located ₁₀₆₈
between continuous borders. These borders should be noticeably thicker than the borders of ₁₀₆₉
the EPNs. A compartment can take **any** geometry. A compartment must always be entirely ₁₀₇₀
enclosed. ₁₀₇₁

Label: The identification of the compartment is carried by an unbordered box containing a string of ₁₀₇₂
characters. The characters can be distributed on several lines to improve readability, although ₁₀₇₃
this is not mandatory. The label box can be attached anywhere in the container box. Note that ₁₀₇₄
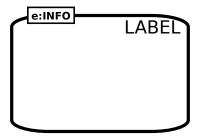the label can spill-over from the container box. ₁₀₇₅

Figure 4.55: *The Process Description glyph for* compartment.

To allow more aesthetically pleasing and understandable maps, compartments are allowed to ₁₀₇₆
overlap each other visually, but it must be kept in mind that this does not mean the top compartment ₁₀₇₇
contains part of the bottom compartment. Figure 4.56 shows two semantically equivalent placement ₁₀₇₈
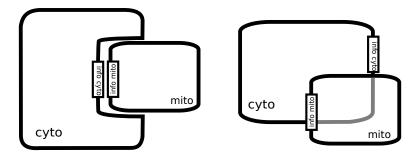of compartments: ₁₀₇₉

Figure 4.56: *Overlapped compartments are permitted, but the overlap does not imply containment.*

Overlapped (hidden) part of the compartment should not contain any object which could be ₁₀₈₀
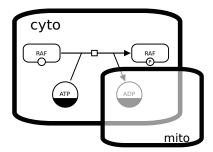covered by an overlapping compartment. Figure 4.57 illustrates the problem using an incorrect map. ₁₀₈₁

Figure 4.57: *Example of an **incorrect** map. Overlapped compartments must not obscure other objects.*

Changes from Previous Version 1082

The use Compartment has a set of natures, which previously were less well specified and handled as 1083
notation provided by the *unit of information*. In some cases, where the CVs used are not distinct or 1084
if the *unit of information* contains arbitrary text as annotation then maps containing these features 1085
will be invalid according to the current specification. 1086
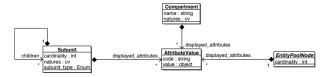
4.5.28 AttributeValue 1087

Figure 4.58: *The UML definition of the AttributeValue and its usage by other classes.*

⚠
See footnote.

The AttributeValue[26] is used to present the values of certain attributes held by other SBGN ele- 1088
ments. It is typically contained and owned by the class containing the attribute (or its descendants). 1089
It contains two values, one is a code to indicate the attribute that is defined and the other is the value 1090
itself. The code and the presentation format of the value are defined by the SBGN element that con- 1091
tains the AttributeValue, currently Compartment (see section 4.5.26), EntityPoolNode (see section 4.5.9), 1092
and Subunit (see section 4.5.19). 1093

Generalisation 1094

- AuxiliaryUnit (see section 4.5.4) 1095

Attributes 1096

code: string **(R)**   The code indicating the attribute that is being presented. 1097
value: object **(R)**   The value of the attribute. The format of the value is determined by the class holding 1098
    the attribute. 1099

Associations 1100

No additional associations. 1101

Rules and Constraints 1102

No additional rules and constraints. 1103

Notation 1104

For historical reasons the AttributeValue is represented graphically by the glyph *Unit of Information*. 1105

Glyph: *Unit of information*   When representing biological entities, it is often necessary to convey 1106
some abstract information about the entity's function that cannot (or does not need to) be easily 1107
related to its structure. The *unit of information* is a decoration that can be used in this situation to 1108
add information to a glyph. Some example uses include: characterizing a logical part of an entity 1109
such as a functional domain (a binding domain, a catalytic site, a promoter, etc.), or the information 1110
encoded in the entity (an exon, an open reading frame, etc.). A *unit of information* can also convey 1111
information about the physical environment, or the specific type of biological entity it is decorating. 1112
SBO Term: Not applicable. 1113

---

[26]A new concept, that modifies the behaviour of the Unit of Information in previous versions with the need to use it to
present the nature and cardinality of an EPN. The glyph retains its original name, but the class has been names to reflect it
purpose.

Container: A unit of information is represented by a rectangle. The long side of the rectangle should 1114 be oriented parallel to the border of the *EPN* being annotated by the *unit of information*. The 1115 center of the bounding box of a *state of information* should be located on the mid-line of the 1116 border of the *EPN*. 1117

Label: A *unit of information* is identified by a label placed in an unbordered box containing a string 1118 of characters. The characters can be distributed on several lines to improve readability, al- 1119 though this is not mandatory. The label box must be attached to the center of the container. 1120 The label may spill outside of the container. 1121
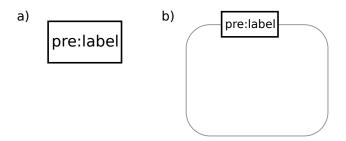

Figure 4.59: *The Process Description glyph for* unit of information. *(a) The glyph. (b) An example of its usage with a* macromolecule.

Changes from Previous Version 1122

There was no definition of the AttributeValue in the previous version of this specification. However, 1123 the *Unit of Information* did exist although its semantics have been changed. It no longer can hold 1124 arbitrary annotation but must display an attribute value and observe the constraints set out by the 1125 definition of the class owning the attribute. 1126

Since the use of the *Unit of Information* has been deprecated, it is recommended that Annotation 1127 (see section 4.5.30) and the *Annotation* glyph is used instead. 1128

4.5.29 StateVariable 1129

Many biological entities such as molecules can exist in different *states*, meaning different physical or 1130 informational configurations. These states can arise for a variety of reasons. For example, macro- 1131 molecules can be subject to post-synthesis modifications, wherein residues of the macromolecules 1132 (amino acids, nucleosides, or glucid residues) are modified through covalent linkage to other chem- 1133 icals. Other examples of states are alternative conformations as in the closed/open/desensitized 1134 conformations of a transmembrane channel, and the active/inactive forms of an enzyme. 1135

In the Process Description language these states are defined by the StateVariableDefinitions associ- 1136 ated with the EntityType, but the specific values of the variables are define by the StateVariable (figure 1137 4.60) associated with the EntityPoolNode. For every StateVariableDefinition associated with an Entity- 1138 Type there should be a corresponding StateVariable associated with the instance of EntityPoolNode 1139 using that type. This enforces one of the fundamental rules of the language that once a state variable 1140 has been displayed for a given entity type, then it must always be displayed. 1141
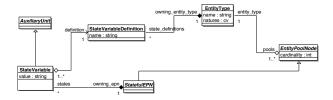

Figure 4.60: *The UML definition of the StateVariable showing its relationship to StatefulEPN and StateVariableDefinition.*

Generalisation ₁₁₄₂

- AuxiliaryUnit (see section 4.5.4) ₁₁₄₃

Attributes ₁₁₄₄

value: string **(R)** The value of the state variable. This is optional, but cannot be an empty string, ₁₁₄₅ should start with a non-space character and end with a non-space character. It should also ₁₁₄₆ not include the '@' character[27]. ₁₁₄₇

⚠ See footnote.

Associations ₁₁₄₈

owning_epn:StatefulEPN (1) The stateful EPN that owns the state variable. ₁₁₄₉
definition:StateVariableDefinition (1) The definition of this state variable[28]. ₁₁₅₀

⚠ See footnote.

Rules and Constraints ₁₁₅₁

No additional rules and constraints. ₁₁₅₂

Notation ₁₁₅₃

Glyph: *State variable* ₁₁₅₄

SBO Term: Not applicable. ₁₁₅₅

Container: A *state variable* is represented by a "stadium" container, that is two hemicercles of same ₁₁₅₆ radius joined by parallel segments, as shown in Figure 4.61. The parallel segment axis should ₁₁₅₇ be tangent to the border of the glyph of the *EPN* being modified by the *state variable*. The ₁₁₅₈ center of the bounding box of a *state variable* should be located on the mid-line of the border ₁₁₅₉ of the *EPN*. In previous versions of this specification the *state variable* was represented by an ₁₁₆₀ ellipse. This symbols is now **deprecated** in favour of the stadium symbol described above. New ₁₁₆₁ Process Description maps should not use the ellipse symbol. ₁₁₆₂

Label: An unbordered box containing a string indicating the contents of the StateVariable. The style ₁₁₆₃ of labeling of *State Variables* encouraged by SBGN Process Description Level 1 is to combine ₁₁₆₄ a prefix representing the value of the variable with a suffix representing the variable's name. ₁₁₆₅ Prefix and suffix should be separated by the symbol '@', X@Y thus meaning *value X* AT *variable* ₁₁₆₆ *Y*. If name is undefined then only the value should be displayed and the '@' character omitted. ₁₁₆₇ If both the name and value are undefined then the label should be empty (i.e., an empty string). ₁₁₆₈ The label of a *state variable* should, if possible, be displayed within the boundary of the glyph. ₁₁₆₉ In earlier versions of the SBGN specification it was permitted to separate the name and value ₁₁₇₀ into two unlabelled boxes and display the name box outside the *state variable* glyph. This is ₁₁₇₁ now **deprecated** and new Process Description maps should not use this notation. ₁₁₇₂
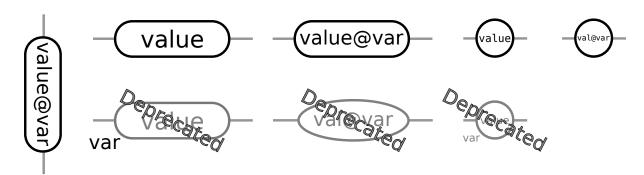

Figure 4.61: *Examples of the Process Description glyph for* state variable. *Included are the older forms of glyph, which are now deprecated.*

---

[27] Not specified is previous spec, but seems essential if we are to avoid ambiguity.
[28] New concept. It aims to ensure that all state variables are defined in all EPNS of the same type.

A *state variable* does not necessarily have to be Boolean-valued. For example, an ion channel can possess several conductance states; a receptor can be inactive, active and desensitized; and so on. As another example, a *state variable* "ubiquitin" could also carry numerical values corresponding to the number of ubiquitin molecules present in the tail. However, in all cases, a *state variable* on an EPN can only take *one* defined value. Further, an EPN's *state variable* should always be displayed and always set to a value. An "empty" *state variable* is a *state variable* that is set to the value "unset", it is not a *state variable* with no value. Note that the value "unset" is *not* synonymous to "any value" or "unknown value".

Changes from Previous Version

The StateVariable class was not explicitly defined in previous versions of the specification, however the *state variable* was. Some aspects of its notation have been deprecated and these are detailed above (section 4.5.29).

4.5.30 Annotation

In SBGN Process Description Level 1 there are cases where the language does not capture everything the author wishes to convey. This may be additional experimental detail or descriptions of mechanisms that cannot be described full by the Process Description language. In this case the language provides the Annotation. This contains text and is associated with a particular glyph in a map. Importantly, it is purely "decoration" and does alter the meaning the map.

Generalisation

- AuxiliaryUnit (see section 4.5.4)

Attributes

annotation_text: string **(R)** The text of the annotation. The text is mandatory and cannot be empty or just spaces.

Associations

annotated_glyph:SBGNGlyph (1) The instance of SBGNGlyph that is being annotated[29].

Rules and Constraints

No additional rules and constraints.

Notation

Glyph: *Annotation*
SBO Term: SBO:NEW
Container: An *annotation* is represented by a rectangular container with a folded corner, as illustrated in Figure 4.62. This container is linked to the annotated element via a callout (see figure 4.63. The callout should overlap with the object it is annotating.
Label: An *annotation* contains information placed in an unbordered box containing a string of characters. The characters can be distributed on several lines to improve readability, although this is not mandatory. The label box must be attached to the center of the container. The label may spill outside of the container.

---

[29]Note that as a result of this association only glyphs and **not** auxiliary items may be annotated by instances of Annotation

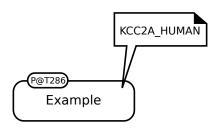**Figure 4.62:** *The Process Description glyph for* annotation.

Figure 4.63: *Example of* annotations *adding information to the description of the trans-phosphorylation of CaMKII. Note that three different types of links are used between annotation nodes and annotated elements. However, it is recommended to use a consistent scheme whithin a map.*

Changes from Previous Version 1210

This is a new language element and an not previous versions of the Process Description language. 1211

4.5.31 CrossReference 1212
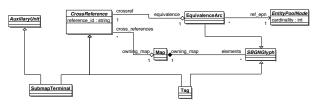
Figure 4.64: *The UML definition of the Crossreference showing its subclasses Tag and SubmapTerminal and its association with other elements in the Process Description language.*

⚠ See footnote.

CrossReference[30] handles links or relationships between elements of a map and sub-map. At 1213 present there is only one reference glyph, *tag*, which can be used in a map refered to by a *submap* 1214 (Section 4.5.23) or as an auxilary unit on the *submap*. The *clone marker* can also provide additional 1215 reference mechanisms and is discussed below (Section 4.5.38). 1216

Generalisation 1217

None 1218

Attributes 1219

reference_id: string **(R)** a string that identifies the cross-reference. The string cannot start and end in 1220 white space and cannot be empty. 1221

---

[30]A new concept requird for consistency between tags ans submap terminals.

Associations 1222

equivalence:EquivalenceArc (1) The equivalence arc that links this class to the referenced element. 1223

Rules and Constraints 1224

- Two or more instances of CrossReference with the same reference_id value are pointing to the 1225
same element. 1226

- The above rules applies within a Process Description map's namespace (see section 4.8). 1227

Changes from Previous Version 1228

Not defined in the previous version. 1229

4.5.32 SubmapTerminal 1230

⚠
See footnote.

A SubmapTerminal[31] (figure 4.30) is a named reference that is part of a SubmapNode (see section 1231
4.5.23). It provides the reference that is the link to a tag in the submap that the SubmapNode refers to. 1232

Generalisation 1233

- AuxiliaryUnit (see section 4.5.4) 1234

- CrossReference (see section 4.5.31) 1235

Attributes 1236

No additional attributes. 1237

Associations 1238

No additional associations. 1239

Rules and Constraints 1240

No additional rules and constraints. 1241

Notation 1242

Glyph: *Submap Terminal* 1243
SBO Term: Not applicable. 1244
Container: A *tag* is represented by a rectangle fused to an empty arrowhead. The flat edge opposite 1245
the arrowhead should be aligned to the edge of the *Submap* glyph and the connecting should 1246
connect to the middle of this face (see figure 4.65). 1247
Label: A *tag* is identified by a label placed in an unbordered box containing a string of characters. 1248
The characters can be distributed on several lines to improve readability, although this is not 1249
mandatory. The label box must be attached to the center of the container. The label may spill 1250
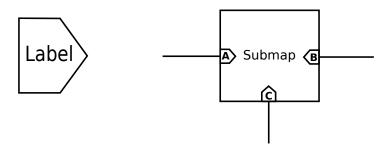outside of the container. 1251

Figure 4.65: *The Process Description glyph for* Submap Terminal. *This shows the basic glyph and its correct usage within a* Submap *glyph.*

---

[31]Not properly defined in previous versions of the spec. The rules described here are consistent with accepted usage.

Changes from Previous Version 1252

Clarified that the tag does not link a Compartment, but only instances of EntityPoolNode. 1253

4.5.33 Tag 1254

A Tag is a named handle, or reference, to another EntityPoolNode. *Tags* are used to identify those 1255
elements in *submaps* (Section 4.5.23). 1256

Generalisation 1257

- SBGNGlyph (see section 4.5.3) 1258

- CrossReference (see section 4.5.31) 1259

Attributes 1260

No additional attributes. 1261

Associations 1262

No additional associations. 1263

Rules and Constraints 1264

- All values of reference_id must be unique within an instance of Map. 1265

Notation 1266

Glyph: *Tag* 1267
SBO Term: Not applicable. 1268
Container: A *tag* is represented by a rectangle fused to an empty arrowhead, as illustrated in Fig- 1269
ure 4.66. The symbol should be linked to one and only one edge (i.e., it should reference only 1270
one EPN or compartment). 1271
Label: A *tag* is identified by a label placed in an unbordered box containing a string of characters. 1272
The characters can be distributed on several lines to improve readability, although this is not 1273
mandatory. The label box must be attached to the center of the container. The label may spill 1274
outside of the container. 1275

Figure 4.66: *The Process Description glyph for* tag.

Changes from Previous Version 1276

Clarified that the tag does not link a Compartment, but only instances of EntityPoolNode. 1277

4.5.34 FluxArc 1278

The FluxArc permits a quantity of entities to flow through the arc and in doing so connects a stoichio- 1279
metric process (StoichiometricProcess (see section 4.5.25)) and an EPN (EntityPoolNode (see section 1280
4.5.9)). The FluxArc has a stoichiometry which is used to indicate the stoichiometry of a process. It is 1281
required to eliminate ambiguity when the exact composition, or the number of copies, of the inputs 1282
or outputs to a reaction are ambiguous from the map (see figure 4.71 for an example). The FluxArc 1283
has three forms determined by its flux_type: 1284

consumption Links an entity pool to a process that will consumed it (an input to the process). 1285
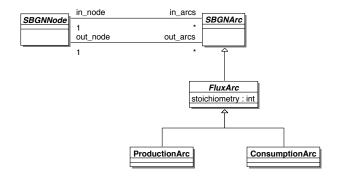
Figure 4.67: *The UML definition of the FluxArc and its subclasses.*

production Links an entity pool to a process that will be produced by a process (an output from the process). It may also links entities on the "right-hand-side" of a reversible process (see section 4.5.25 for more details about the reversible process).

reversible Links a process to an entity pool that is the "left-hand-side" of a process (see section 4.5.25).

Generalisation

- SBGNArc (see section 4.5.6)

Attributes

stoichiometry: int **(R)** The stoichiometry of this FluxArc (see section 4.5.34). This must be a non-zero positive integer. flux_typeenumR The type of the flux arc. One of the following: consumption, production and reversible.

No additional attributes.

Associations

No additional associations.

Rules and Constraints

⚠
See footnote.
⚠
See footnote.

- if the stoichiometry is not displayed then it is assumes to be 1[32]

- If the stoichiometry > 1 then the stoichiometry must be displayed.[33]

- if flux_type = 'consumption' or 'reversible' then:

  - The in_node must be an instance of EntityPoolNode (see section 4.5.9).
  - The out_node must be an instance of StoichiometricProcess (see section 4.5.25).

- if flux_type = 'production' then:

  - The in_node must be an instance of StoichiometricProcess (see section 4.5.25).
  - The out_node must be an instance of EntityPoolNode (see section 4.5.9).

Notation

The FluxArc is represented by three glyphs depending on its flux_type:

consumption *consumption arc.*

production *production arc.*

⚠
See footnote.

reversible *reversible arc*[34].   1313

    These are defined below. In addition the stoichiometry is displayed by the *stoichiometry label*. Its   1314
appearance and layout in relation to the flux arc is also described here.   1315

Glyph: *Consumption*   1316
SBO Term: SBO:0000394 ! consumption.   1317
End point: No particular symbol is used to represent a consumption.   1318
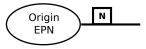
Figure 4.68: *The Process Description glyph for* consumption.

Glyph: *Production*   1319
SBO Term: SBO:0000393 ! production.   1320
End point: The target extremity of a *production* carries a filled arrowhead.   1321

Figure 4.69: *The Process Description glyph for* production.

Glyph: *Reversible*   1322
SBO Term: SBO:0000393 ! production.   1323
End point: The origin extremity of a *reversible* carries a filled arrowhead.   1324

Figure 4.70: *The Process Description glyph for* reversible.

Glyph: *Stoichiometry Label*   The stoichiometry label is part of the *consumption arc* and *production*   1325
arc glyphs see below (sections 4.5.34 and 4.5.34). However, as their use is common to all subclasses   1326
of FluxArc their presentation is described here.   1327

---

[32]The previous version said that the stoichiometry was unspecified, but this was before the stoichiometric process was specified. In addition since cardinality is defined explicitly the stoichiometry can often be implied.

[33]See above comment.

[34]The reversible arc is introduced here to fix syntax problems in previous spec. The change is visually neutral on previous Process Description maps.
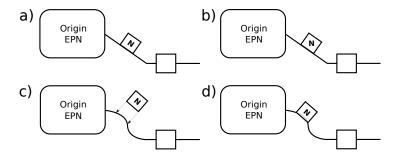
Figure 4.71: *Examples of stoichiometry label layout. In figure (a) the label is aligned with the stoichiometry box, while in (b) the label is aligned with the orientation of the map: these are both simple cases where the arc is a straight line. In cases where the arc is curved, the corners at the base of the label are anchored to point on the arc (c) and the label is drawn over the arc (d). Note that in (d) the covered part of the arc is shown for clarity, but normally the box is opaque and so the arc is not visible.*

The label is a node that must be drawn above the flux arc. This node is attached to the arc where 1328
it intersects the arc with its bottom corners (see figure 4.71.)[35]. 1329

SBO Term: None 1330

Container: A rectangle with a draw edge. 1331

Label: A number that should remain within the container and be of a normal font, i.e., not bold or 1332
italic. 1333

Changes from Previous Version 1334

The *reversible arc* has been added to ensure the correct syntax is observed for a reversible process[36] 1335
— using the *production arc* for this was syntactically forbidden in the previous version of the spec- 1336
ification. There is no change to semantics from previous version, but layout rules for placement of 1337
stoichiometry label have been clarified. 1338
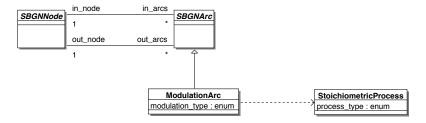
4.5.35 ModulationArc 1339

Figure 4.72: *The UML definition of the ModulationArc. The class interacts with subclasses of StoichiometricProcess.*

The ModulationArc (figure 4.72) affects the flux of a process represented by the target process. Such 1340
a modulation can affect the process **positively or negatively**, or even both ways depending on the 1341
conditions, for instance the concentration of the intervening participants. The permitted values for 1342
process_type are described in the following table: 1343

---

[35]These layout rules were agreed at COMBINE 2010, but have not been voted on.

[36]New glyph. We need it to ensure the correct syntax as we have a directed graph and we need to prevent the production arc pointing towards the process. It looks the same as the previous situation.

modulation	A general modulation where the exact nature of the modulation is not specified or not known. Modulation can be used when one does not know the precise direction of the effect.
stimulation	A stimulation affects **positively** the flux of a process represented by the target process. This stimulation can be for instance a catalysis or a positive allosteric regulation. Note that *catalysis* exists independently in SBGN, see Section 4.5.35.
catalysis	A particular case of stimulation, where the effector affects positively the flux of a process represented by the target process. The positive effect on the process is due to the lowering of the activation energy of a reaction.
inhibition	An inhibition **negatively** affects the flux of a process represented by the target process. This inhibition can be for instance a competitive inhibition or an allosteric inhibition.
necessary_stim	A necessary stimulation, is one that is necessary for a process to take place. A process modulated by a necessary stimulation can only occur when this necessary stimulation is active.

As discussed in Chapter 2, it is implied, but not defined explicitly that the process has a rate at which it converts its LHS EPNs to its RHS EPNs (and vice-versa in the case of a reversible process). This concept is important in understanding how the Process Description language describes process modulation[37].

⚠ See footnote.

1. A *process* with no modulations has an underlying "basal rate" which describes the rate at which it converts inputs to outputs.

2. A *modulation* changes the basal rate in an unspecified fashion.

3. A *stimulation* is a modulation that increases the basal rate.

4. An *inhibition* is a modulation that decreases the basal rate.

5. The above types of modulation, when assigned to the same process, are combined and have a multiplicative effect on the basal rate of the process.

6. Modulators that do not interact with each other in the above manner, should be drawn as modulating different process nodes. Their effect is therefore additive.

Generalisation

- EntityPoolNode (see section 4.5.9)

Attributes

No additional attributes.

Associations

states:StateVariable (*) The state variables associated with this EPN.

Rules and Constraints

- At most one *necessary stimulation* can be assigned to a process node. Two *necessary stimulations* would imply an implicit AND or OR operator. For clarity only one *necessary stimulation* can be assigned to a *process*, and such combinations must be explicitly expressed using *logical operators*[38].

⚠ See footnote.

---

[37]This is taken from the grammar section of the previous specification.
[38]Taken from the grammar section of the previous specification.

- At most one *catalysis* can be assigned to a *process*. Modulation by a catalysis arc implies that the exact biochemical mechanism underlying the process is known. In this context two *catalysis* cannot be assigned to the same process node as they represent independent reactions. Other EPNs can be assigned to the same process as a catalysis, such as modulators, stimulators, and inhibitors, and will have a multiplicative modulation on the reaction rate defined by the catalysis[39]. 1369 1370 1371 1372 1373 1374

⚠
See footnote.

Notation 1375

The ModulationArc is represented by a number of glyphs depending on its modulation_type. The table below defines what glyph is used for each type. 1376 1377

Type	Glyph
modulation	*Modulation*
stimulation	*Stimulation*
inhibition	*Inhibition*
necessary_stim	*Necessary Stimulation*

1378

Glyph: *Modulation* 1379
SBO Term: SBO:0000168 ! control. 1380
End point: The target extremity of a *modulation* carries an empty diamond. 1381
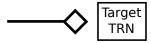

Figure 4.73: *The Process Description glyph for* modulation.

Figure 4.74 represents the effect of nicotine on the process between closed and open states of a nicotinic acetylcholine receptor. High concentrations of nicotine open the receptor while low concentrations can desensitize it without opening. 1382 1383 1384
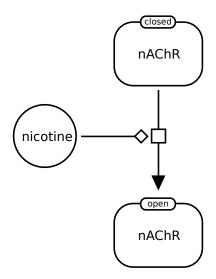

Figure 4.74: *Modulation of nicotinic receptor opening by nicotine.*

---

[39]Taken from the grammar section of the previous specification.

Glyph: *Stimulation* 1385

SBO Term: SBO:0000170 ! stimulation. 1386

End point: The target extremity of a *stimulation* carries an empty arrowhead. 1387

Figure 4.75: *The Process Description glyph for* stimulation.

Glyph: *Catalysis* 1388

SBO Term: SBO:0000172 ! catalysis. 1389

Node: The target extremity of a *catalysis* carries an empty circle. 1390

Figure 4.76: *The Process Description glyph for* catalysis.

Glyph: *Inhibition* 1391

SBO Term: SBO:0000169 ! inhibition. 1392

Node: The target extremity of an *inhibition* carries a bar perpendicular to the arc. 1393
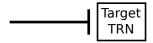
Figure 4.77: *The Process Description glyph for* inhibition.

Glyph: *Necessary stimulation* 1394

SBO Term: SBO:0000171 ! necessary stimulation. 1395

Node: The target extremity of a *necessary stimulation* carries an open arrow (to remind that it is a *stimulation*) coming after a larger vertical bar. 1396 1397

Figure 4.78: *The Process Description glyph for* Necessary Stimulation.

Examples The example in Figure 4.79 below describes the transcription of a gene X, that is the cre- 1398
ation of a messenger RNA X triggered by the gene X. The creation of the protein X is then triggered 1399
by the mRNA X. (Note that the same example could be represented using the gene as reactant and 1400
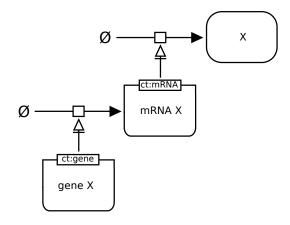product, although it is semantically different.) 1401

Figure 4.79: *The creation of a messenger RNA X triggered by the gene X.*

The example in Figure 4.80 below describes the transport of calcium ions out of the endoplasmic 1402
reticulum. Without IP3 receptor, there is not calcium flux, therefore, one cannot use a *stimulation*. 1403
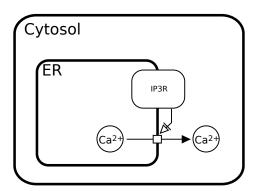The Necessary Stimulation instead represents this absolute stimulation. 1404

Figure 4.80: *The transport of calcium ions out of the endoplasmic reticulum into the cytosol. Note that IP3R crosses both compartment boundaries. This is allowed, but the Macromolecule should only belong to one of the compartments see section D.1 for more discussion of this issue.*

Changes from Previous Version 1405

The definition of ModulationArc did not exist in the previous version but there has been no changes to 1406
the glyphs and glyph semantics in this version. 1407

4.5.36 LogicArc 1408

The LogicArc (figure 4.81) takes a quantity from either a LogicalOperator (see section 4.5.24) or an 1409
EntityPoolNode (see section 4.5.9) and converts it into a Boolean output, which serves as an input for 1410
a LogicalOperator (see section 4.5.24). How this is done is not defined, but one could imagine that 1411
when a threshold value of the quantity is exceeded the output is True. 1412

Generalisation 1413

- SBGNArc (see section 4.5.6) 1414

Attributes 1415

No additional attributes. 1416

Associations 1417
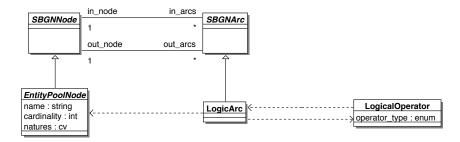
No additional associations. 1418

Figure 4.81: *The UML definition of the LogicArc and its context.*

Rules and Constraints 1419

- The in_node must be an instance of EntityPoolNode or LogicalOperator. 1420

- The out_node must be an instance of LogicalOperator. 1421

Notation 1422

Glyph: *Logic arc* *Logic arc* is used to represent the fact that an entity influences the outcome of a 1423
logic operator. 1424
SBO Term: SBO:0000398 ! logical relationship. 1425
End point: No particular symbol is used to represent a logic arc. 1426

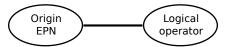**Figure 4.82:** *The Process Description glyph for* logic arc.

Changes from Previous Version 1427

No changes from the previous version. 1428

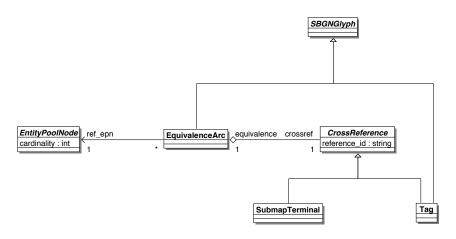### 4.5.37 EquivalenceArc 1429

Figure 4.83: *The UML definition of the EquivalenceArc and its context.*

EquivalenceArc (figure 4.83) is the arc used to link a cross-reference to an EPN in another Process 1430
Description map (represented by CrossReference (see section 4.5.31)) with an EPN (EntityPoolNode 1431
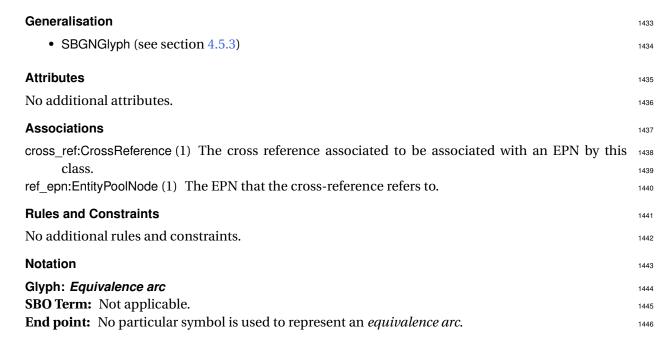(see section 4.5.9)) in this map. 1432

Generalisation   1433

- SBGNGlyph (see section 4.5.3)   1434

Attributes   1435

No additional attributes.   1436

Associations   1437

cross_ref:CrossReference (1) The cross reference associated to be associated with an EPN by this   1438
class.   1439

ref_epn:EntityPoolNode (1) The EPN that the cross-reference refers to.   1440

Rules and Constraints   1441

No additional rules and constraints.   1442

Notation   1443

Glyph: *Equivalence arc*   1444
SBO Term: Not applicable.   1445
End point: No particular symbol is used to represent an *equivalence arc.*   1446

Figure 4.84: *The Process Description glyph for* Equivalence arc.

Changes from Previous Version   1447

The relationship of EquivalenceArc to the SubmapTerminal (see section 4.5.32) was unclear in previous   1448
versions of the specification and has been clarified here.   1449

4.5.38 CloneMarker   1450
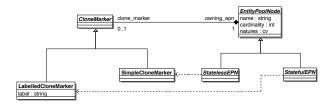

Figure 4.85: *The UML definition of the StateVariable showing its relationship to StatefulEPN, Com-
plex and Subunit.*

If an EntityPoolNode (see section 4.5.9) is duplicated on a map, it is necessary to indicate this fact   1451
by the CloneMarker auxiliary unit (figure 4.85). The purpose of this marker is to provide the reader   1452
with a visual indication that this node has been cloned, and that at least one other occurrence of the   1453
EntityPoolNode can be found in the map (or in a submap; see Section 4.5.23). The clone marker takes   1454
two forms, simple and labeled, depending on whether the node being cloned can carry state variables   1455
(i.e., whether it is a stateful EPN). Note that an EntityPoolNode belongs to a single compartment. If   1456
two classes named "X" are located in two different compartments, such as ATP in cytosol and ATP in   1457
mitochondrial lumen, they represent different Entity Pools, and therefore do not need to be marked   1458
as cloned.   1459

Generalisation 1460

- AuxiliaryUnit (see section 4.5.4) 1461

Attributes 1462

No additional attributes. 1463

Associations 1464

owning_epn:EntityPoolNode (1) The EPN that holds this clone marker. 1465

Rules and Constraints 1466

No additional rules and constraints. 1467

Changes from Previous Version 1468

Not defined in previous version. 1469

4.5.39 SimpleCloneMarker 1470

The SimpleCloneMarker (figure 4.85) is the unlabelled subclass CloneMarker. All duplicated instances 1471
of StatelessEPN must contain an instance of this class. 1472

Generalisation 1473

- CloneMarker (see section 4.5.38) 1474

Attributes 1475

No additional attributes. 1476

Associations 1477

No additional associations. 1478

Rules and Constraints 1479

- Only subclasses of StatefulEPN (see section 4.5.15) can contain labelled clone markers. 1480

Notation 1481

Simple clone marker 1482

SBO Term: Not applicable. 1483

Container: The simple (unlabeled) *clone marker* is a portion of the surface of an *EPN* that has been 1484
modified visually through the use of a different shade, texture, or color. Figure 4.86 illustrates 1485
this. The *clone marker* occupies the lower part of the *EPN*. The filled area must be smaller than 1486
the unfilled one. 1487

Label: Not applicable. 1488
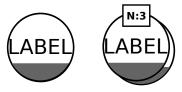
Figure 4.86: *The Process Description glyph for* simple clone marker *applied to a* simple chemical
and a multimer *of* simple chemicals.

Figure 4.87 contains an example in which we illustrate the use of *clone markers* to clone the 1489
species ATP and ADP participating in different reactions. This example also demonstrates the chief 1490
drawbacks of using clones: it leads to a kind of dissociation of the overall network and multiplies 1491

the number of nodes required, requiring more work on the part of the reader to interpret the result. 1492
Sometimes these disadvantages are offset in larger maps by a reduction in the overall number of line 1493
crossings, but not always. In general, we advise that cloning should be used sparingly. 1494
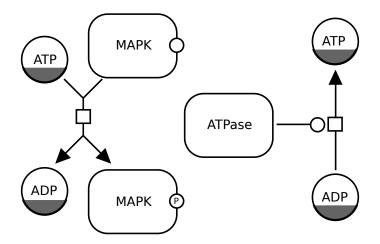

Figure 4.87: *An example of using cloning, here for the species ATP and ADP.*

Changes from Previous Version 1495

No change from previous version. 1496

4.5.40 LabelledClonerMarker 1497

Unlike the classrefSimpleCloneMarker, the LabeledCloneMarker (figure 4.85) includes (unsurprisingly, 1498
given its name) an identifying label that can be used to identify equivalent clones elsewhere in the 1499
map. This is particularly useful for subclasses of StatefulEPN (see section 4.5.15), because these can 1500
have a large number of state variables displayed and therefore may be difficult to visually identify as 1501
being identical. 1502

Generalisation 1503

- CloneMarker (see section 4.5.38) 1504

Attributes 1505

label: string **(R)** The label that identified the clone. This label must start and end with an alphanu- 1506
meric character, and cannot contain white space. 1507

Associations 1508

No additional associations. 1509

Rules and Constraints 1510

- At least two or more instanced of a LabelledCloneMarker with the same label must exist in this 1511
same in a given Map (see section 4.5.2). 1512

- Only subclasses of StatefulEPN (see section 4.5.15) can contain labelled clone markers. 1513

Notation 1514

Labeled clone marker 1515
SBO Term: Not applicable. 1516
Container: The labeled *clone marker* is a portion of the surface of an *EPN* that has been modified 1517
visually through the use of a different shade, texture, or color. The *clone marker* occupies the 1518

lower part of the EPN glyph. The filled area must be smaller than the unfilled one, but the be 1519
large enough to have a height larger than the *clone marker*'s label (cf below). 1520

Label: A *clone marker* is identified by a label placed in an unbordered box containing a string of 1521
characters. The characters can be distributed on several lines to improve readability, although 1522
this is not mandatory. The label box must be attached to the center of the container. The label 1523
may spill outside of the container (the portion of the surface of the EPN that has been modified 1524
visually). The font color of the label and the color of the clone marker should contrast with one 1525
another. The label on a *labeled clone marker* is mandatory. 1526

Figure 4.88: *The Process Description glyph for* labeled clone marker *applied to a* macromolecule,
a nucleic acid feature *and a* multimer *of* macromolecules.

Changes from Previous Version 1527

No changes from previous version. 1528

4.6 Controlled vocabularies 1529

Some classes in the SBGN Process Description language can contain particular kinds of textual anno- 1530
tation conveying information relevant to the class. Examples are the natures of an EntityPoolNode (see 1531
section 4.5.9) or Compartment (see section 4.5.26) or the value of the StateVariable (see section 4.5.29). 1532
The values held by these attributes can be taken from controlled vocabularies defined below. When 1533
displayed in some cases is mandatory to prefix a code indicating the type of controlled vocabulary 1534
used. This is in order to make it clear what the information is that the value referes to: for example 1535
'mt' indicates that a value 'rna' is describing the material RNA. 1536

In the rest of this section, we describe the controlled vocabularies (CVs) used in SBGN Process 1537
Description Level 1. They cover the following categories of information: an EPN's material type, an 1538
EPN's conceptual type, covalent modifications on macromolecules and the physical characteristics. 1539
These controlled vocabularies are *closed* in the sense that only the values defined for each CV can 1540
be used in a valid Process Description map and also closed because only the controlled vocabularies 1541
⚠ defined in this specification can be used in a valid Process Description map [40]. We understand that 1542
See footnote. this is of necessity restrictive, but in closing these definitions it means we can be clear about the 1543
meaning of all CV terms used in the specification. Updates to the CV terms and the CVs used are 1544
welcome and we encourage any changes or additions to be submitted as a tracker item at the address 1545
given on the front page of this specification. 1546

4.6.1 Entity pool node material types 1547

The material type of an EPN indicates its chemical structure. A list of common material types is 1548
shown in Table 4.2, but others are possible. The values are to be taken from the Systems Biology On- 1549
tology (http://www.ebi.ac.uk/sbo/), specifically from the branch having identifier SBO:0000240 1550
(*material entity* under *physical entity representation*). The labels are defined by SBGN Process De- 1551
scription Level 1. 1552

---

[40]The previous version was a bit vaque but implied that the CVs were open. This has been changed after discussion on
the list.

Name	Label	SBO term
Non-macromolecular ion	`mt:ion`	`SBO:0000327`
Non-macromolecular radical	`mt:rad`	`SBO:0000328`
Ribonucleic acid	`mt:rna`	`SBO:0000250`
Deoxribonucleic acid	`mt:dna`	`SBO:0000251`
Protein	`mt:prot`	`SBO:0000297`
Polysaccharide	`mt:psac`	`SBO:0000249`

Table 4.2: *A sample of values from the* material types *controlled vocabulary (Section 4.6.1).*

The material types are in contrast to the *conceptual types* (see below). The distinction is that material types are about physical composition, while conceptual types are about roles. For example, a strand of RNA is a physical artefact, but its use as messenger RNA is a role.

4.6.2 Entity pool node conceptual types

An EPN's *conceptual type* indicates its function within the context of a given Process Description. A list of common conceptual types is shown in Table 4.3, but others are possible. The values are to be taken from the Systems Biology Ontology (http://www.ebi.ac.uk/sbo/), specifically from the branch having identifier `SBO:0000241` (*functional entity* under *physical entity representation*). The labels are defined by SBGN Process Description Level 1.

Name	Label	SBO term
Gene	`ct:gene`	`SBO:0000243`
Transcription start site	`ct:tss`	`SBO:0000329`
Gene coding region	`ct:coding`	`SBO:0000335`
Gene regulatory region	`ct:grr`	`SBO:0000369`
Messenger RNA	`ct:mRNA`	`SBO:0000278`

Table 4.3: *A sample of values from the* conceptual types *vocabulary (Section 4.6.2).*

4.6.3 Macromolecule covalent modifications

A common reason for the introduction of state variables (Section 4.5.29) on an entity is to allow access to the configuration of possible covalent modification sites on that entity. For instance, a macromolecule may have one or more sites where a phosphate group may be attached; this change in the site's configuration (i.e., being either phosphorylated or not) may factor into whether, and how, the entity can participate in different processes. Being able to describe such modifications in a consistent fashion is the motivation for the existence of SBGN's covalent modifications controlled vocabulary.

Table 4.4 lists a number of common types of covalent modifications. The most common values are defined by the Systems Biology Ontology in the branch having identifier `SBO:0000210` (*addition of a chemical group* under *interaction→process→biochemical or transport reaction→ biochemical reaction→conversion*). The labels shown in Table 4.4 are defined by SBGN Process Description Level 1; for all other kinds of modifications not listed here, the author of a Process Description must create a new label (and should also describe the meaning of the label in a legend or text accompanying the map).

Name	Label	SBO term
Acetylation	Ac	SBO:0000215
Glycosylation	G	SBO:0000217
Hydroxylation	OH	SBO:0000233
Methylation	Me	SBO:0000214
Myristoylation	My	SBO:0000219
Palmytoylation	Pa	SBO:0000218
Phosphorylation	P	SBO:0000216
Prenylation	Pr	SBO:0000221
Protonation	H	SBO:0000212
Sulfation	S	SBO:0000220
Ubiquitination	Ub	SBO:0000224

Table 4.4: *A sample of values from the* covalent modifications *vocabulary (Section 4.6.3).*

4.6.4 Physical characteristics

SBGN Process Description Level 1 defines a special unit of information for describing certain common physical characteristics. Table 4.5 lists the particular values defined by SBGN Process Description Level 1.

Name	Label	SBO term
Temperature	pc:T	SBO:0000147
Voltage	pc:V	SBO:0000259
pH	pc:pH	SBO:0000304

Table 4.5: *A sample of values from the* physical characteristics *vocabulary (Section 4.6.4).*

4.7 Entity Pool Node Identity and Cloning

All elements in an SBGN Process Description map have an implicit identify defined by the id attribute in SBGNElement (see section 4.5.1) so in that respect all elements drawn are unique. However, in some cases it is possible to draw two or elements that define identical information in the map. In some clases, such as the Compartment this is prohibited, but in others (EntityPoolNode) it is not, but requires special decoration (the clone marker) to indicate that the information is replicated on the map. We define this type of identity and "logical identity" and the attributes that so discriminate between elements as the "logical key"[41].

⚠ See footnote.

Clearly it is therefore important that we define the logical key for all the elements in the map and that we do this for the class definitions (section 4.5). The conventions used for this is as follows:

- If no logical key is specified then instance identity applies (see above).

- If the logical key is defined then this should be applied when determining equality between two elements.

- If an element can be replicated and two elements are determined to be logically identical, then instance identity should be used to determine uniqueness.

---

[41]This is recapitulation of the rules as defined in the previous version of the specification and as it is understood in discussions.

- If an element can be replicated if marked as a clone, then the above rule applies. If the element is not marked as cloned, then the only the logical key can be used when determining uniqueness. 1595 1596 1597

Figure 4.89 illustrates how the above rules are applied in practise in the Process Description map. They also affect our understanding when reading a map. For example since all processes are unique, two *Process* glyphs connected to the same input and output indicates that the same "reaction" can be carried out by two distinct processes, with potentially different mechanisms. 1598 1599 1600 1601
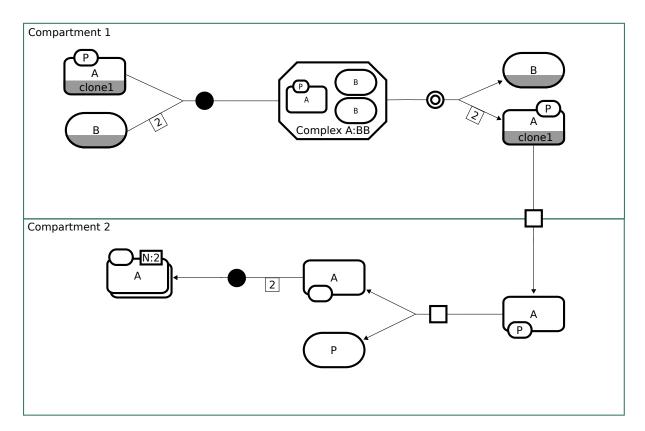
Figure 4.89: *This fictitious example illustrates the correct cloning semantics in the Process Description language. The diagram shows the association of the phosphorylated macromolecule A (A[P]) and simple chemical B into a complex. Two copies of B are consumed and these are shown as separate subunits in the complex (note that no clone marker is used). The complex then dissociates into the same components that formed it which requires an appropriate clone marker to be applied. A[P] is translocated into compartment 2 where A[P] is not cloned as it does not have a duplicate. The unphosphorylated from of A dimerised, but again this does not require a clone marker as these are regarded as separate entity pools.*

4.8 Map and Submap Linking 1602

The submap is a visual device that allows the detail of an Process Description map to be exported into another Process Description map and replaced by a SubmapNode (see section 4.5.23), which acts as a place-holder. In the case of an SBGN description that is made available through a software tool, the content of a SubmapNode may be available to the tool. A user could then ask the tool to expand it, for instance by clicking on the *submap node* glyph. The tool might then expand and show its submap within the same map, or it might display it as a separate map. In the case of an SBGN description made available in a book or a website, the content of the *submap node* may be available on another page, possibly accessible via a hyperlink.[42] 1603 1604 1605 1606 1607 1608 1609 1610

⚠
See footnote.

---

[42] Currently we can't name a map so linking between supermaps and submaps is problematic.

As you can see the rules relating to the submap concept range wider than those applying to the 1611
SubmapNode so we will detail these rules here. The concepts and nomenclature here can get confus- 1612
ing, so we will use the following nomenclature in the rest of this section[43]: 1613

⚠ See footnote.

supermap The map that owns the submap and so contains the submap glyph referring to it. 1614

submap The map that contains the detail summarised by the submap glyph in the super-map. 1615

reference_id the value displayed by the tag and submap terminal. These must be the same to indi- 1616
cate equivalence between map and submap. 1617

Figure 4.90 represents a *submap node* that transforms glucose into fructose-6-phosphate. The 1618
submap node carries five terminals, four linked to EPNs and one linked to a *compartment*. The latter 1619
is particularly important in the case of EPNs present only in a *compartment* enclosed in a *submap* 1620
node, and that are not linked to terminals themselves. Note that the terminals do not define a "direc- 1621
tion", such as input or output. The flux of the reactions is determined by the context. 1622
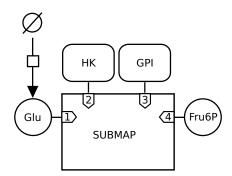
Figure 4.90: *Example of a submap with contents elided.*

The map in Figure 4.91 represents an unfolded version of a *submap*. Here, anything outside the 1623
submap node has disappeared, and the internal *tags* are not linked to the corresponding external 1624
submap terminals.[44] 1625

⚠ See footnote.
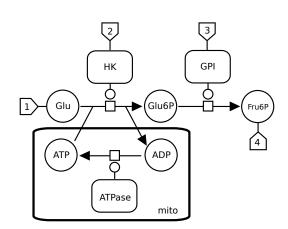
Figure 4.91: *Example of an unfolded submap. The unfolded submap corresponds to the folded submap of Figure 4.90.*

---

[43]This nomenclature has not been used previously. The previous nomenclature was a bit confusing and didn't capture the hierarchy of maps well.

[44]This example in the previous version fo the spec included compartment tagging. This has been removed, but the example may need refinement as a result.

4.8.1 Namespace and the relationship between maps 1626

The following rules apply to the use of the SubmapNode and submaps in the Process Description [1627] language [45]. [1628]

- Submaps can contain SubmapNode instances so submaps can contain submaps - there is no [1629] limit on the depth of this. [1630]

- Submaps share the same namespace as their supermap. Therefore submaps of submaps share [1631] the top-most map's namespace. [1632]

- Compartment's can be replicated across submaps and they all refer to the same compartment [1633] if they share the same name. [1634]

- Subclasses of EntityPoolNode replicated in both map and its immediate submap must be as- [1635] sociated with a subclass of CrossReference (SubmapTerminal for the supermap and Tag for the [1636] submap). [1637]

- Subclasses of EntityPoolNode may be replicated between submaps separated by one or more [1638] other submaps, i.e. those not affected by the above rule. [1639]

- All reference_ids used must be unique within a given map namespace. [1640]

- submap folding: all tagged nodes in submap merge with those linked to terminals in super- [1641] map. If either node is cloned then all equivalent nodes in merged map must share the appro- [1642] priate clone marker. [1643]

- For a valid mapping between subclasses of EntityPoolNode in a supermap and submap to exist, [1644] the EntityPoolNode subclasses must be logically identical and the reference_id in their associated [1645] CrossReference must be identical. identical. [1646]

- A SubmapNode which do not have a corresponding submap is valid. [1647]

- Submaps without a corresponding supermap are invalid[46]. [1648]

- Subclasses of EntityPoolNode with the same name must share the same EntityType. [1649]

4.8.2 Compartment spanning 1650

An *EPN* cannot *belong* to more than one *compartment*. However, an EPN can be *drawn* over more [1651] than one *compartment*. In such cases, the decision on which is the owning *compartment* is deferred [1652] to the drawing tool or the author. A *complex* may contain EPNs which belong to different *compart-* [1653] *ments* and in this way a *complex* can be used to describe entities that span more than one compart- [1654] ment. [1655]

This restriction makes it impossible to represent in a semantically correct way a macromolecule [1656] that spans more then one compartment — for example a receptor protein. It is clearly desirable to [1657] be able to show a macromolecule in a manner that the biologist expects (i.e. spanning from the [1658] outside through the membrane to the inside). Therefore, the author is recommended to draw the [1659] macromolecule across compartment boundaries, but the underlying SBGN semantic model will as- [1660] sign it to only one. The assignment to a *compartment* may be decided by the software drawing tool [1661] or the author. Note that this has implications for auto-layout algorithms as they will only be able to [1662] treat such *entity pool nodes* as contained within a *compartment* and will have no way of knowing a [1663] macromolecule spans a compartment. [1664]

---

[45]These rules are taken from the previous version of the spec, but rules about the depth of submap 'recursion', the unique- ness of reference_id values and cloning between maps are not

[46]We need to sort out mapping between super and sub maps to get this working. I propose naming the maps

The current solution is consistent with other Systems Biology representations such as SBML and BioPAX. For more information about the problems representing membrane spanning proteins and the rationale behind the current solution see Section D.

Chapter 5

Layout Rules for a Process Description

5.1 Introduction

The previous chapters describe the appearance and meaning of SBGN Process Description Level 1
components. Here we describe rules governing the visual appearence and asthetics of the Process
Description language. The components of a Process Description have to be placed in a meaningful
way – a random distribution with spaghetti-like connections will most likely hide the information
encoded in the underlying model, whereas an elegant placement of the objects, giving a congenial
appearance of the maps, may reveal new insights. The arrangement of components in a map is called
a *layout*.

SBGN Process Descriptions should be easily recognisable not only by the glyphs used, but also
by the general style of the layout. However, the arrangement of the components is a complex art in
itself, and there is no simple rule which can be applied to all cases. Therefore this section provides
rules for the layout of process description maps, divided into two categories:

1. requirements, i. e. rules which **must** be fulfilled by a layout, and

2. recommendations, i. e. rules which **should** be followed if possible.

In addition, we provide a list of additional suggestions which may help in producing aesthetically
more pleasant layouts, possibly easier to understand.

Those layout rules are independent of the method used to produce the map, and apply to both
manually drawn maps as well as maps produced by an automatic layout algorithm. The rules do not
deal with interactive aspects (e. g. the effect of zooming). Further information about automatic net-
work layout (graph drawing) can be found, for example, in the books of Di Battista and co-authors [3]
and Kaufmann and Wagner [4].

Please note that the color of objects do not carry any meaning in SBGN. Although one can use
colors to emphasize part of a map or encode additional information, the meaning of the map should
not depend on the colors. Furthermore, objects can have different sizes and size is also meaningless
in SBGN. For example, a process node may be larger than a protein node. Also the meaning of a graph
should be conserved upon scaling as far as possible.

5.2 Requirements

Requirements are rules which **must** be fulfilled by a layout to produce a valid Process Description
map.

5.2.1 Node-node overlaps

Nodes are only allowed to overlap in two cases when they are allowed to contain other nodes — as
described in Chapter 4. Otherwise, nodes are not allowed to overlap (Figure 5.1). This includes the

touching of nodes. Touching is not allowed apart from the case where it has a specific meaning, e.g. two macromolecules touching each other within a complex because they form the complex. 1702 1703
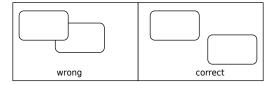

Figure 5.1: *Nodes must not overlap.*

5.2.2 Node-edge crossing 1704

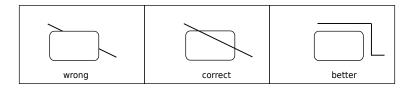Edges must be drawn on the top of a the node (Figure 5.2). See also recommendation Section 5.3.1. 1705

Figure 5.2: *If an edge crosses a node, the edge must be drawn on top of the node.*

5.2.3 Node border-edge overlaps 1706

Edges are not allowed to overlap the border lines of nodes (Figure 5.3). 1707
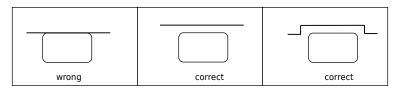
Figure 5.3: *Edges must not overlap node borders.*

5.2.4 Edge-edge overlaps 1708

Edges are not allowed to overlap (Figure 5.4). This includes touching of edges. Furthermore, an edge is neither allowed to cross itself nor to cross a boundary of node more than twice or other edges more than once. 1709 1710 1711

5.2.5 Node orientation 1712

Nodes have to be drawn horizontally or vertically, any other rotation of elements is not allowed (Figure 5.5). 1713 1714

5.2.6 Node-edge connection 1715

1. The arcs linking the square glyph of a *process* to the *consumption* and *production arcs* are attached to the center of opposite sides (Figure 5.6). 1716 1717
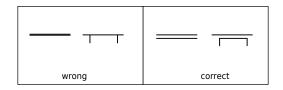
Figure 5.4: *Edges must not overlap.*
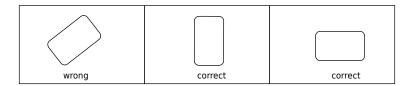
Figure 5.5: *The node orientation must be horizontally or vertically.*

2. The modulatory arcs are attached to the other two sides, but not necessarily all to the center, as several modifiers can affect the same process node.

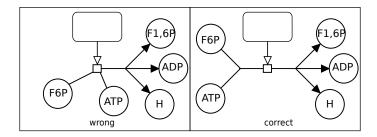**Figure 5.6:** *Arcs between a* process *and the* consumption *and* production arcs *must be attached to the center of opposite sides, modulatory arcs must be attached to the other two sides.*

5.2.7 Node labels

At least a part of the label (unbordered box containing a string of characters) has to be placed inside the node it belongs to. Node labels are not allowed to overlap other nodes or other labels (this includes touching of other nodes or labels).

5.2.8 Edge labels

Edge labels are not allowed to overlap nodes. This includes touching of nodes.

5.2.9 Compartments

If a process has all participants in the same compartment the process node and all edges/arcs should be drawn in this compartment. If a process has participants in at least two different compartments, the process node has to be either in a compartment where the process has at least one participant or in the empty space.

5.3 Recommendations

Recommendations are rules which should be followed if possible and generally should improve the clarity of the diagram.

5.3.1 Node-edge crossing 1734

Situations where edges and nodes cross should be avoided. Note that some crossings may be un- 1735
avoidable, e. g. the crossing between an edge and a compartment border or an edge and a complex 1736
(if the edge connects an element inside the complex with something outside). 1737

5.3.2 Labels 1738

Labels should be horizontal. Node labels should be placed completely inside the node if possible. 1739
Edge labels should be placed close to the edge and avoid overlapping the edge as well as other edge 1740
labels. 1741

5.3.3 Avoid edge crossings 1742

The amount of crossings between edges should be minimized. 1743

5.3.4 Branching of *association* and *dissociation* 1744

The branching points of *association* and *dissociation* nodes should be placed closed to the symbol 1745
of the process, if possible at a distance comparable than, or smaller to, the diameter of the symbol 1746
defining the process (Figure 5.7). 1747
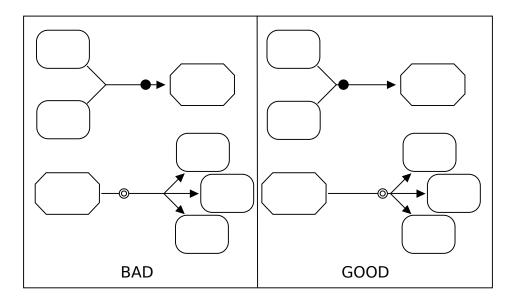
Figure 5.7: *Branching points should be close to association and dissociation symbols.*

5.3.5 Units of information 1748

Units of information should not hide the structure of the corresponding node and should not overlap 1749
other elements (Figure 5.8). 1750

5.4 Additional suggestions 1751

Here is a list of additional layout suggestions which may help improve the asthetics and clarity of 1752
Process Description maps. 1753

- Angle of edge crossings: If edge crossing cannot be avoided then the edges should cross with 1754
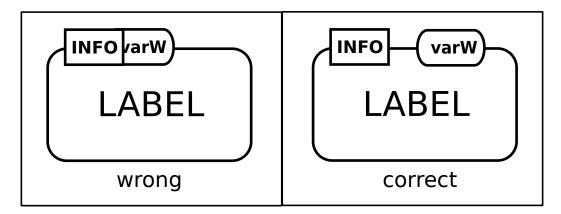 an angle close to 90 degrees. 1755

Figure 5.8: *Units of information should not overlap with any other element.*

- Drawing area and width/height ratio: The drawing should be compact and the ratio between the width and the height of the drawing should be close to 1.
- Edge length: Long edges should be avoided.
- Number of edge bends: Edges should be drawn with as few bends as possible.
- Similar and symmetric parts: Similar parts of a map should be drawn in a similar way, and symmetric parts should be drawn symmetrically.
- Proximity information: Related elements (e. g. nodes connected by a process or all elements within a compartment) should be drawn close together.
- Directional information: Subsequent processes (e. g. a sequence of reactions) should be drawn in one direction (e. g. from top to bottom or from left to right).
- Compartments: It can help clarity to use a different background shade or color for each compartment.

Chapter 6

Acknowledgments

Here we acknowledge those people and organisations the assisted in the develeopment of this and previous releases of the SBGN Process Description language specification. First we specifically acknowledge those who contributed directly to each revision of the specification document, followed by a comprehensive acknowledgement of contributors that attended workshops and forum meetings or in some other way provided input to the standard. Finally, we acknowledge the bodies that provided financial support for the development of the standard.

6.1 Level 1 Release 1.0

The specification of was written by Nicolas Le Novère, Stuart Moodie, Anatoly Sorokin, Michael Hucka, Falk Schreiber, Emek Demir, Huaiyu Mi, Yukiko Matsuoka, Katja Wegner and Hiroaki Kitano. In addition, the specification benefited much from the help of (in alphabetical order) Frank Bergmann, Sarala Dissanayake, Ralph Gauges, Peter Ghazal, Lu Li, and Steven Watterson.

6.2 Level 1 Release 1.1

The specification of SBGN PD Level 1.1 was written by Stuart Moodie and Nicolas Le Novère, with contributions from (in alphabetical order) Frank Bergmann, Sarah Boyd, Emek Demir, Sarala Wimalaratne, Yukiko Matsuoka, Huaiyu Mi, Falk Schreiber, Anatoly Sorokin, Alice Villéger.

6.3 Level 1 Release 1.2

The specification of SBGN PD Level 1.2 was modified by Stuart Moodie, with contributions from (in alphabetical order) Sarah Boyd, Nicolas Le Novère, Huaiyu Mi.

6.4 Level 1 Release 1.3

The specification of SBGN PD Level 1.3 was modified by Stuart Moodie, with contributions from (in alphabetical order), Tobias Czauderna, Nicolas Le Novère, Anatoly Sorokin.

6.5 Comprehensive list of acknowledgements

Here is a more comprehensive list of people who have been actively involved in SBGN development, either by their help designing the languages, their comments on the specification, help with development infrastructure or any other useful input. We intend this list to be complete. We are very sorry if we forgot someone, and would be grateful if you could notify us of any omission.

6.6 Financial Support

Appendix A

Complete examples of Process Description Maps

The following maps present complete examples of SBGN Process Descriptions representing Biological processes. They by no mean exhaust the possibilities of SBGN Process Description Level 1.

Figure A.1 presents an example of metabolic pathway, that examplifies the use of the *EPNs simple chemical*, *macromolecule*, and *clone marker*, the *PNs process*, and the *connecting arcs consumption*, *production* and *catalysis*.
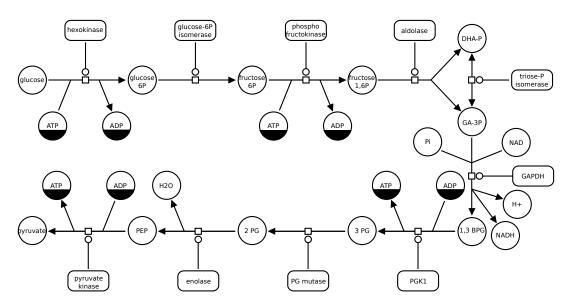

Figure A.1: *Glycolysis. This example illustrates how SBGN can be used to describe metabolic pathways.*

Figure A.2 presents an example of signalling pathway, that examplifies in addition the use of the *EPNs phenotype*, and *state variable*, the *containers complex*, *compartment* and *submap*, the *PNs association*, and the *connecting arcs stimulation*. Note the complex IGF and IGF receptor, located on the boundary of the compartment. This position is only for user convenience. The complex has to belong to a given compartment in SBGN Process Description Level 1.

Figure A.3 is an expanded version of the submap present on the map present in Figure A.2. It shows the use of *tag*.

Figure A.4 introduces an SBGN Process Description that spans several compartments. Note that the compartment "synaptic vesicle" is not **contained** in the compartment "synaptic button" but **overlaps** it. The *simple chemical* "ACh" of the "synaptic vesicle" is not the same *EPN* than the "ACh" of the

77
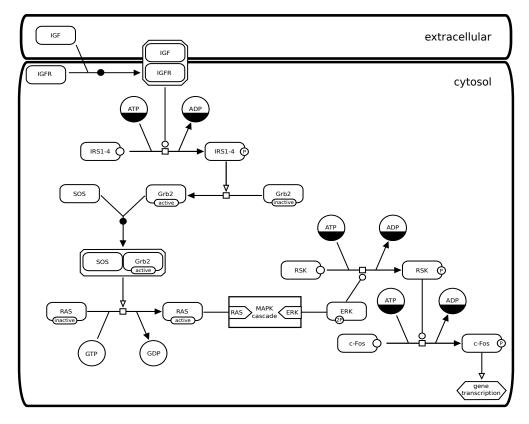
Figure A.2: *Insulin-like Growth Factor (IGF) signalling. This example shows the use of compartments and how details can be hidden by using a submap. The submap is shown on Figure A.3.*

"synaptic button" and of "synaptic cleft". The situation is similar with the compartments "ER" and "muscle cytosol". The map exemplifies the use of the *PN omitted* and *dissociation*, and the *connecting*
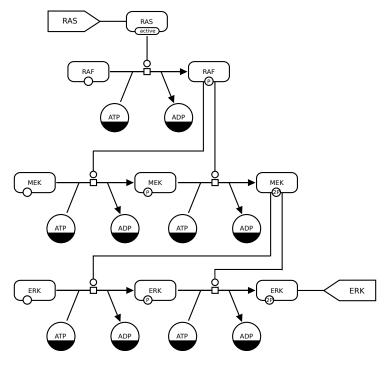
Figure A.3: *A submap of the previous map showing the MAPK cascade.*

arc necessary stimulation.

1839
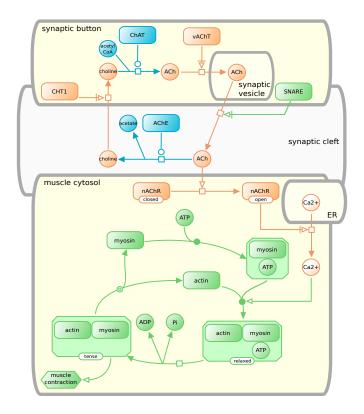
Figure A.4: *Neuronal/Muscle signalling. A description of inter-cellular signalling using SBGN.*

Figure A.5 introduces the use of SBGN Process Description Level 1 to encode gene regulatory networks. It also show the use of the *EPNs Source* and the *logical operator and*.

1840

1841
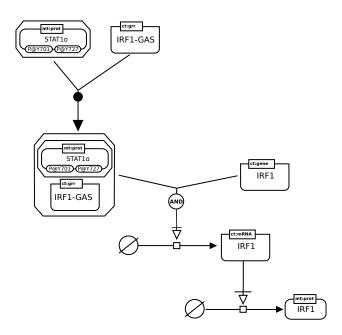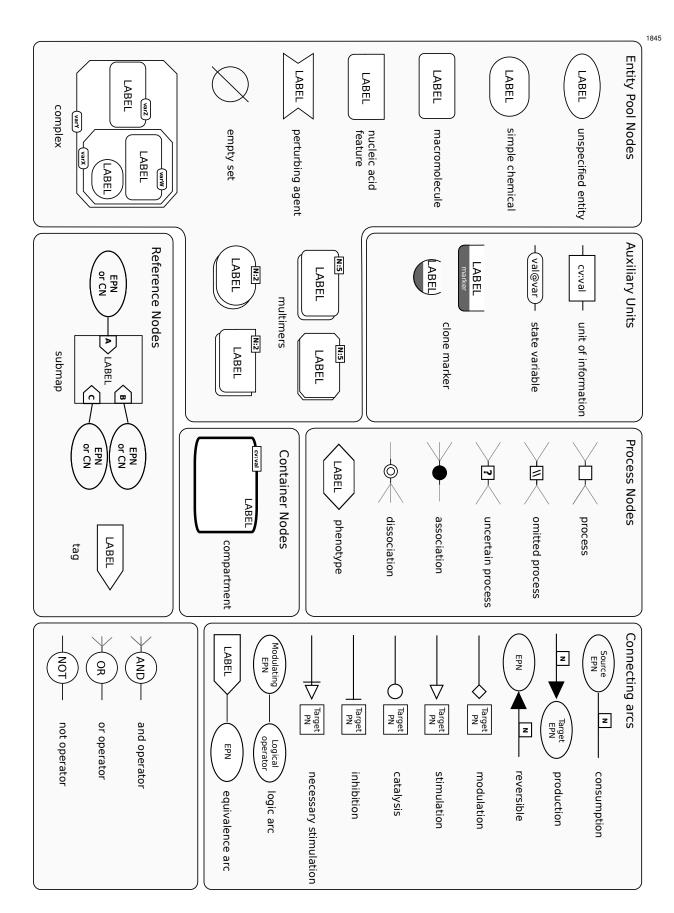
Figure A.5: *Activated STAT1α induction of the IRF1 gene. An example of gene regulation using logical operators.*

Appendix B

Reference card

Print the summary of SBGN symbols on the next page for a quick reference.

1845

Entity Pool Nodes

- LABEL — unspecified entity
- LABEL — simple chemical
- LABEL — macromolecule
- LABEL — nucleic acid feature
- LABEL — perturbing agent
- empty set
- complex (varY, varZ, varX, varW, LABEL, LABEL, LABEL)

Auxiliary Units

- cv:val — unit of information
- val@var — state variable
- LABEL marker — clone marker
- (LABEL)
- LABEL N:2, LABEL N:2 — multimers
- LABEL N:5, LABEL N:5

Process Nodes

- process
- omitted process
- ? uncertain process
- association
- dissociation
- LABEL — phenotype

Container Nodes

- LABEL cv:val — compartment

Reference Nodes

- EPN or CN; LABEL A, B, C; EPN or CN; EPN or CN — submap
- LABEL — tag

Connecting arcs

- Source EPN — N — consumption
- N — production
- Target EPN
- EPN — N — reversible
- Target PN — modulation
- Target PN — stimulation
- Target PN — catalysis
- Target PN — inhibition
- Target PN — necessary stimulation
- Modulating EPN — Logical operator — logic arc
- LABEL — EPN — equivalence arc
- AND — and operator
- OR — or operator
- NOT — not operator

Appendix C

Validation and consistency rules

This section describes criteria that must be met by the diagram (or SBGN-ML diagram description) 1848
to conform SBGN Process Description Level 1 specification. 1849

    All criteria has unique and perennial numbering. In the case when modification of the specifica- 1850
tion makes a rule irrelevant it will be deleted but the number of the rule will not be taken, so there 1851
will be gaps in rule numbering. 1852

C.1 EPN rules

000001. Elements that do not comply SBGN Process Description Level 1 specification are not allowed 1854
      on SBGN Process Description Level 1 map. 1855

000002. All instances of SBGNGlyph (see section 4.5.3) must be unique (see section 4.7). 1856

000003. The set of SBGNNodes linked to this node via a SBGNArc (its adjacent nodes) must be all belong 1857
      to different entity pools (as defined by EntityPoolNode) and cannot include more than one clone 1858
      of the same entity pool. 1859

Appendix D

Issues postponed to future levels

D.1 Multicompartment entities

The problem of entities, such as macromolecules, spanning several compartments proved to be a challenge for the community involved in the development of SBGN Process Description Level 1. It was thus decided to leave it for a future Level. It turns out there is at the moment no obvious solution satisfactory for everyone. Three broad classes of solutions have been identified so far:

- One can systematically locate an *EPN* in a given *compartment*, for instance a transmembrane receptor in a membrane. However, the reactions of this entity with entities represented by *EPN* in other compartments, such as extracellular ligands and second messenger systems, will create artificial transport reactions.

- One can represent the domains of proteins in different compartments by *macromolecules*, and link all those macromolecules in a *complex* spanning several compartments. However, such a representation would be very confusing, implying that the domains are actually different molecules linked through non-covalent bonds.

- On can accept *macromolecules* that span several compartments, and represent domains as *units of information*. Those *units of information* should then be located in given compart- ments. To make a full use of such a representation, one should then start and end connecting arcs on given *units of information*, something prohibited by the current specification.

D.2 Logical combination of state variable values

The value of a *state variable* has to be perfectly defined in SBGN Process Description Level 1. If a state variable can take the alternative values 'A', 'B' and 'C', one cannot attribute it values such as 'non-A', 'A or B', 'any' or 'none'. As a consequence some biochemical processes cannot be easily represented because of the very large number of state to enumerate. The decision to forbid such a Boolean logic lies in the necessity of maintaining truth path all over an SBGN map.

D.3 Non-chemical entity nodes

The current specification cannot represent combinations of events and entities. For instance a vari- able "voltage" cannot be controlled by a difference of concentration between different entities, such as a given ion in both sides of a membrane.

D.4 Generics 1889

SBGN Process Description Level 1 does not provide mechanisms to sub-class *EPNs*. There is no spe- 1890
cific means of specifying that *macromolecules* or *nucleic acid features* X1, X2 and X2 are subclasses of 1891
X. Therefore, any process that applies to all the subtypes of X has to be triplicated. That situation can 1892
easily generate combinatorial explosions of the number of *EPNs* or *PNs*. 1893

D.5 State and transformation of compartments 1894

In SBGN Process Description Level 1 a *compartment* is a stateless entity. It cannot carry *state vari-* 1895
ables, and cannot be subjected to process modifying a state. As a recult, a *compartment* cannot be 1896
transformed, moved, split or merged with another. If one want to represent the transformation of a 1897
compartment, one has to create the start and end compartments, and represent the transport of all 1898
the *EPNs* from one to the other. This is not satisfactory, and should be addressed in the future. 1899

Appendix E

Revision History

E.1   Version 1.0 to Version 1.1

Below are the changes incorporated into Version 1.1 of the SBGN Process Description Level 1 speci-
fication.

Description	Tracker ID
Regarding modulation of reversible processes, changed "should" to "must" be represented by two *process*	
Removed "The connectors and the box move as a rigid entity" in the definition of *process*	
Changed the definition of process node to "represent processes that transform one or several EPNs into *one or several EPNs, identical or different*"	
Changed SBO term of *compartment* From SBO:0000289 (functional compartment) to SBO:0000290 (physical compartment)	
Reoganised classification of glyphs	
Reoganised glyph section to reflect the above changes	
Revised reference card to reflect changes in glyph organisation	
Revised logic operators throughout spec to make sure input and output arcs meet before attaching to the glyph - as with processes.	
Added enumerated rules to grammar section. This is probably not complete, but should help the implementation of semantc validation by software tools. The hope is this will be refined as tools start validating maps.	
Updated UML maps and data dictionary to be consistent with rest of changes to spec.	
Definition of cardinality is ambiguous	2840996
Sink and source are lumped together	2726435
SBO terms are incorrect or missing.	2841261
Compartment description is confusing and contradictory.	2841122
Clone marker fill percentages unhelpful.	2841114
Use of CV for physical charactetistic not clear.	2841085
Definition of Cardinality is ambiguous.	2840996
input to AND on IFN example.	2804326
more SBO terms for *multimers*	2803593
Legend of figure 2.20 is incorrect	2803537
legend of figure 3.2	2802990
	continued on next page

continued from previous page	
Description	Tracker ID
Compartment colouring	2745703
Errors in diag a4.	2664912
Change name of trigger glyph.	2664908
Transition should be renamed process.	2664862
Converting arcs tautological.	2664843
Example invalid.	2545870
consumption and production.	2388317
Should require circles to be distinguishable from ellipses	2219388
Figure 2.53	2162619
Reference card: production	2104471
Figure 2.42 is wrong	2104465
Mistake in the multi-cellular example	2395488
Should not prevent processes having identical in and out	2664933
No description of linking to subunit rules.	2545810
Extensively revised the grammar section. The UML diagrams have been simplified to show glyph taxonomy, and the data dictionary has been pruned to just show glyph identity. The some syntax rules have been moved into semantics and the rules reformulated to make them easier to understand.	
Eliminated duplicate rules in layout section and revised text slightly.	
Phenotype cloning?	2989007
Perturbing agent description	2940021

E.2 Version 1.1 to Version 1.2

Below are the changes incorporated into Version 1.2 of the SBGN Process Description Level 1 specification.

Description	Tracker ID
Perturbing agent description	2940021
Members of complex touching	2849273
PD Reference card error for submap glyph	3029242

E.3 Version 1.2 to Version 1.3

Below are the changes incorporated into Version 1.3 of the SBGN Process Description Level 1 specification.

Description	Tracker ID
Incorrect editor on title page	
Typos in acknowledgements	
Fixed typo in item on catalysis in section on modulation semantics.	
State variables figure 2.6 V1.2	3090543

Bibliography

[1] Object management Group. OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. Available via the World Wide Web at http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/, 2010.

[2] Michael Hucka, Frank T. Bergmann, Stefan Hoops, Sarah M. Keating, Sven Sahle, James C. Schaff, Lucian P. Smith, and Darren J. Wilkinson. The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core. Available via the World Wide Web at http://sbml.org/Documents/Specifications, November 2007.

[3] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, New Jersey, 1998.

[4] M. Kaufmann and D. Wagner. *Drawing Graphs: Methods and Models,* volume 2025 of *Lecture Notes in Computer Science Tutorial.* Springer, 2001.