

ETSI
TECHNICAL
REPORT

ETR 298

September 1996

Source: ETSI TC-MTS

Reference: DTR/MTS-00033-1

ICS: 33.020

Key words: SDL, ASN.1, MSC, methodology

**Methods for Testing and Specification (MTS);
Specification of protocols and services;
Handbook for SDL, ASN.1 and MSC development**

ETSI

European Telecommunications Standards Institute

ETSI Secretariat

Postal address: F-06921 Sophia Antipolis CEDEX - FRANCE

Office address: 650 Route des Lucioles - Sophia Antipolis - Valbonne - FRANCE

X.400: c=fr, a=atlas, p=etsi, s=secretariat - **Internet:** secretariat@etsi.fr

Tel.: +33 92 94 42 00 - Fax: +33 93 65 47 16

Copyright Notification: No part may be reproduced except as authorized by written permission. The copyright and the foregoing restriction extend to reproduction in all media.

© European Telecommunications Standards Institute 1996. All rights reserved.

Contents

Foreword	5
1 Scope	7
2 References	7
3 Definitions and abbreviations	8
3.1 Definitions	8
3.1.1 External definitions	8
3.1.2 Internal definitions	8
3.2 Abbreviations	8
4 Introduction.....	9
5 Formalization of SDL (with ASN.1 and MSC).....	9
5.1 Criteria to start formalization	9
5.2 Formalization steps.....	10
5.3 Structure steps (S-steps).....	10
5.3.1 Step S:1 Boundary and environment of the system	10
5.3.2 Step S:2 Discrete system communications.....	11
5.3.3 Step S:3 System parts.....	12
5.3.4 Step S:4 Communication paths between parts	13
5.3.5 Step S:5 Associating signals to communication paths.....	13
5.3.6 Step S:6 Information hiding and sub-structuring	14
5.3.7 Step S:7 Block constituents.....	15
5.3.8 Step S:8 Local signals in a block.....	15
5.4 Behaviour steps (B-steps)	15
5.4.1 Step B:1 Set of signals to a process	16
5.4.2 Step B:2 Skeleton processes	17
5.4.3 Step B:3 Informal processes	17
5.4.4 Step B:4 Complete processes.....	18
5.5 Data steps (D-steps).....	19
5.5.1 Step D:1 Signal parameters	19
5.5.2 Step D:2 Process and procedure parameters.....	20
5.5.3 Step D:3 Signal variables	20
5.5.4 Step D:4 Formal transitions.....	21
5.5.5 Step D:5 Output and create parameters	21
5.5.6 Step D:6 Data signatures	22
5.5.7 Step D:7 Informal data description	23
5.5.8 Step D:8 Formal data description.....	24
5.5.9 Step D:9 Complete data formalization.....	24
5.6 Results of formalization	24
5.7 Criteria for adequate formalization.....	25
6 Examples.....	25
6.1 Presentation of the PUMI example	25
6.2 PUMI example: the stepwise formalization.....	26
6.2.1 Structure-steps (S-steps)	26
6.2.1.1 Step S:1 Boundary and environment of the system	26
6.2.1.2 Step S:2 Discrete system communications	27
6.2.1.3 Step S:3 System parts.....	28
6.2.1.4 Step S:4 Communication paths between parts	29
6.2.1.5 Step S:5 Associating signals to communication paths	29
6.2.1.6 Step S:6 Information hiding and sub structuring.....	30
6.2.1.7 Step S:7 Block constituents.....	30
6.2.1.8 Step S:8 Local signals in a block.....	34
6.2.2 Behaviour-steps (B-steps).....	35

6.2.2.1	Step B:1 Set of signals to a process	35
6.2.2.2	Step B:2 Skeleton processes	36
6.2.2.3	Step B:3 Informal processes.....	38
6.2.2.4	Step B:4 Complete processes	42
6.2.3	Data-steps (D-steps)	43
6.2.3.1	Step D:1 Signal parameters.....	43
6.2.3.2	Step D:2 Process and procedure parameters	50
6.2.3.3	Step D:3 Signal variables.....	51
6.2.3.4	Step D:4 Formal transitions	53
6.2.3.5	Step D:5 Output and create parameters.....	54
6.2.3.6	Step D:6 Data signatures	54
6.2.3.7	Step D:7 Informal data description.....	54
6.2.3.8	Step D:8 Formal data description	55
6.2.3.9	Step D:9 Complete data formalization	55
Annex A:	Overview of ETS 300 414	56
A.1	Introduction	56
A.2	Normative interfaces.....	57
A.3	Selection of SDL concepts.....	57
A.4	Selection of Message Sequence Chart concepts.....	58
A.5	Selection of ASN.1 concepts	60
Annex B:	Overview of methodology	61
B.1	The methodology activities	61
B.2	The requirement collection activity	62
B.3	The classification activity	63
B.4	The draft design activity.....	64
B.5	The formalization activity	65
B.6	The derivation of a validation model activity	66
B.7	The documentation activity	66
Annex C (informative):	List of rules of ETS 300 414 [1]	68
Annex D:	List of rules in formalization.....	71
Annex E :	Allowed symbols.....	73
Annex F:	Bibliography.....	76
History	77

Foreword

This ETSI Technical Report (ETR) has been produced by the Methods for Testing and Specification (MTS) Technical Committee of the European Telecommunications Standards Institute (ETSI).

This ETR is the result of the preliminary MTS studies for the production of a comprehensive methodology for the specification of ETSI protocols and services when using SDL. Such studies were performed by the PT60 Phase 1 and 2 in 1993 - 1994 with TC-MTS supervision. With respect to the initial ambitions, the present ETR concentrates on giving guidance on the SDL formalization phases, which were felt the more coherent urgent and important after the publication of ETS 300 414 [1].

ETRs are informative documents resulting from ETSI studies which are not appropriate for European Telecommunication Standard (ETS) or Interim European Telecommunication Standard (I-ETS) status. An ETR may be used to publish material which is either of an informative nature, relating to the use or the application of ETSs or I-ETSs, which is immature and not yet suitable for formal adoption as an ETS or an I-ETS.

Blank page

1 Scope

The purpose of this handbook is to provide guidance to rapporteurs and Project Team experts in applying some methodological steps when developing functional specifications within European Telecommunications Standards (ETSS). The handbook can be used by experienced standards developers as well as those who do not have advanced knowledge of SDL.

ETSS specifying services or protocols should use the ITU Specification and Description Language, SDL, defined in ITU-T Recommendation Z.100 [3] and Message Sequence Charts defined in ITU-T Recommendation Z.120 [6] to specify behaviour. The Abstract Syntax Notation One formalism defined in ITU-T Recommendations X.208 [9] and X.680 [16] may be used in combination with SDL to specify data in compliance with rules defined in ITU-T Recommendation Z.105 [4].

This handbook is based on the work presented in ETS 300 414 [1].

2 References

For the purposes of this ETR, the following references apply:

- [1] ETS 300 414 (1995): "Methods for Testing and Specification (MTS); Use of SDL in European Telecommunications Standards Rules for testability and facilitating validation".
- [2] CEN/CENELEC Internal Regulations (1991): Part 3 "Rules for the drafting and presentation of European Standards (PNE-Rules)".
- [3] ITU-T Recommendation Z.100 (1994): "CCITT Specification and Description Language (SDL)".
- [4] ITU-T Recommendation Z.105 (1994): "SDL combined with ASN.1".
- [5] ITU-T Recommendation Z.110 (1989): "Criteria for the use and applicability of formal description techniques".
- [6] ITU-T Recommendation Z.120 (1993): "Message Sequence Chart".
- [7] ITU-T Recommendation I.130 (1988): "Method for the characterization of telecommunication services supported by an ISDN and network capabilities of an ISDN".
- [8] ITU-T Recommendation Q.65 (1988): "Stage 2 of the method for the characterization of services supported by an ISDN".
- [9] ITU-T Recommendation X.208 (1988): "Specification of Abstract Syntax Notation One (ASN.1)".
- [10] ITU-T Recommendation X.209 (1988): "Specification of basic encoding rules for Abstract Syntax Notation One (ASN.1)".
- [11] ITU-T Recommendation X.290: "OSI conformance testing methodology and framework for protocol Recommendations for CCITT applications; General concepts".
- [12] ITU-T Recommendation X.291: "OSI conformance testing methodology and framework for protocol Recommendations for CCITT applications; Abstract test suite specification".
- [13] ITU-T Recommendation X.292: "OSI conformance testing methodology and framework for protocol Recommendations for CCITT applications; The Tree and Tabular Combined Notation (TTCN)".

- [14] ITU-T Recommendation X.293: "OSI conformance testing methodology and framework for protocol Recommendations for CCITT applications; Test realisation".
- [15] ITU-T Recommendation X.294: "OSI conformance testing methodology and framework for protocol Recommendations for CCITT applications; Requirements on test laboratories and clients for the conformance assessment process".
- [16] ITU-T Recommendation X.680 (1993): "Information technology - Abstract Syntax Notation One (ASN.1) - Specification of basic notation".

3 Definitions and abbreviations

3.1 Definitions

3.1.1 External definitions

This technical report uses the following terms defined in other documents:

Abstract Syntax Notation One	(ITU-T Recommendation X.208 [9]).
Message Sequence Chart	(ITU-T Recommendation Z.120 [6]).
Specification and Description Language	(ITU-T Recommendation Z.100 [3]).
Tree and Tabular Combined Notation	(ITU-T Recommendation X.292 [13]).

3.1.2 Internal definitions

For the purposes of this ETR, the following definitions apply:

application domain: The field of activity in which the system under specification will operate.

documentation: The activity of creating an ETS from the formal SDL description and other information generated during application of this methodology, in accordance with certain standards for format and style.

formalization: The stepwise activity in which a formal SDL description of a system is produced.

formal SDL description: An SDL description which conforms to ITU-T Recommendation Z.100 [3] and does not contain any SDL "informal text" and which, thus, can be interpreted by automatic tools.

informal SDL description: An SDL description which conforms to ITU-T Recommendation Z.100 [3] but which includes some SDL "informal text". and which, thus, cannot be interpreted by automatic tools.

validation: The process, with associated methods, procedures and tools, by which an evaluation is made that a standard can be fully implemented, conforms to rules for standards, and satisfies the purpose expressed in the record of requirements on which the standard is based; and that an implementation that conforms to the standard has the functionality expressed in the record of requirements on which the standard is based.

validation model: A detailed version of a specification, possibly including parts of its environment, that is used to perform formal validation.

3.2 Abbreviations

For the purposes of this ETR, the following abbreviations apply:

ASN.1	Abstract Syntax Notation One [9]
MSC	Message Sequence Chart [6]
SDL	Specification and Description Language [3]
TTCN	Tree and Tabular Combined Notation [13]

4 Introduction

The need to validate deliverables and the associated need to ensure the testability of specifications are the driving factors in developing methodologies to be used for the production of ETSs. ITU-T Recommendation Z.100 [3] has been identified as a suitable specification language which also fulfils requirements for uniform and comprehensible documentation based on a standard.

ETS 300 414 [1] identifies instructions, rules and guidelines for the development of SDL specifications within ETSI to ensure that they are testable and can be validated.

TC-MTS and in particular Project Team 60 described a methodology which covers the complete development process for ETSs using SDL. It is described as a set of coupled activities which can be combined and which range from informal specification, through specification design and finally to formal specification. The use of SDL is supplemented by the incorporation of two other formalisms, ITU-T Recommendation X.208 [9] and ITU-T Recommendation Z.120 [6].

This ETR is a handbook for rapporteurs and other writers of deliverables. It describes a "hands-on" process based on ETS 300 414 [1] and other TC-MTS work. It includes action steps to identify and resolve specification problems and high-risk issues early in the formalization of a specification. This process is described in clause 5.

Overviews of ETS 300 414 [1] and other TC-MTS investigations are provided in annexes A and B respectively.

Summaries of all rules applied in the formalization process are listed in annexes C and D.

Graphical symbols allowed in ITU-T Recommendation Z.100 [3] and ITU-T Recommendation Z.120 [6] formalisms are described in annex E.

5 Formalization of SDL (with ASN.1 and MSC)

The purpose of stepwise formalization is to produce a formal specification of a system to be used as the basis for an ETS and for validation. ETSI supports the use of ITU-T Recommendation Z.100 [3] as the main formalism for describing behaviour with ITU-T Recommendation Z.120 [6] to specify message flow relationships and ITU-T Recommendation X.208 [9] to describe the data contents of messages. Figure 1 describes the input and output of the formalization activity.

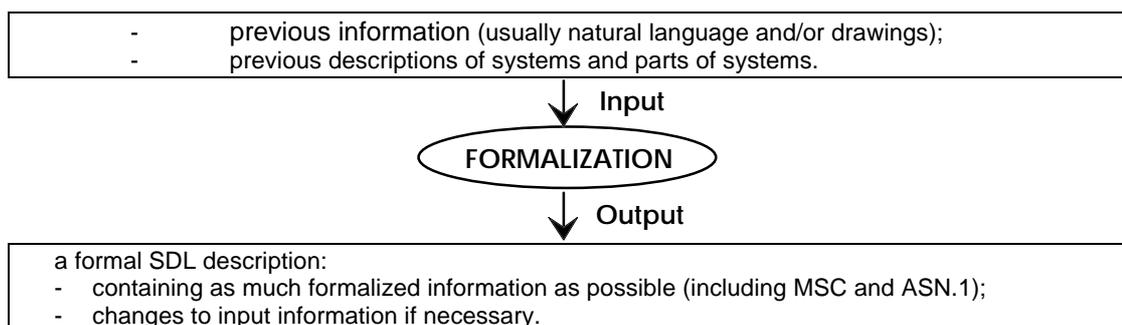


Figure 1: Input and output of formalization

5.1 Criteria to start formalization

Before formalization can start, the following information should exist:

- 1) a description of the entities that are to be modelled in ITU-T Recommendation Z.100 [3] (Functional Entity descriptions, information flows and/or MSCs, service elements, ASN.1 data descriptions, action descriptions, etc.);
- 2) a description of the concepts and names for the main elements of the system;
- 3) a description of each of the normative interfaces.

5.2 Formalization steps

The formalization process is divided into steps as follows:

- Structure steps (S-steps);
- Behaviour steps (B-steps);
- Data steps (D-steps).

In the descriptions of these steps there are subclauses on instructions, rules and guidelines.

Parts of the input information may already be in ITU-T Recommendation Z.100 [3], ITU-T Recommendation X.208 [9] and ITU-T Recommendation Z.120 [6]. In such cases the formalization may be simplified as these parts need only to be checked (and changed if needed) to the language rules and the rules given for each step below.

Before the individual steps are described it is worth considering how they can be applied.

The steps are presented in a sequence which can be followed in strict order. However, this is not always necessary. Depending on the user's experience in SDL modelling, the complexity of the application domain and the quality and completeness of the input information, the application of particular steps can proceed in any order and can be interleaved. The important aspect to note is that all steps need to be completed to achieve the final result. Formalization is an iterative process and it is to be expected that some steps will have to be repeated and intermediate models changed, improved and refined.

During the application of these steps it is important that a record is maintained of any questions raised and decisions taken. An important reason for this is the need for elements of SDL descriptions to be traced back to requirements.

To simplify the description of the Structure, Behaviour and Data steps, certain keywords from the SDL syntax have been used in the text. These are printed in **bold** so that they may be easily identified. The formal definitions of these are in ITU-T Recommendation Z.100 [3] annex B.

5.3 Structure steps (S-steps)

The purpose of the structure steps (S-steps) is to define in ITU-T Recommendation Z.100 [3] the external and internal interfaces of the system and to partition the system into SDL blocks.

5.3.1 Step S:1 Boundary and environment of the system

Instructions:

- 1) identify the boundaries between the system to be described and its environment;
- 2) find a suitable name for the system;
- 3) draw an SDL system diagram with the identified name and explain the system and its relation to the environment informally in a comment within the SDL system diagram.

Rules:

Rule 1 The system boundary defines what is going to be specified (described). Communicating entities within a system have to be specified (described) irrespective of whether they are normative or informative. Entities outside the boundaries are assumed to exist, but cannot be described in SDL. The communication is possible only by means of exchanged discrete messages. The message in SDL is called **signal**;

Rule 2 The unit for time data should be recorded in a comment in the system diagram.

Guidelines:

- 1) the SDL system may be a closed system with no external interfaces. In this case, the SDL system describes the behaviour that is the subject of an ETS and the entities that communicate with it. These entities are informative. The normative interfaces for the ETS may need to be internal interfaces of the SDL system;
- 2) if MSCs have already been produced, identify the axes corresponding to entities considered to belong to the system and, separately, the axes for entities that belong to the external environment. The latter then have no formal description in the SDL specification (or are provided only as informative parts of the system);
- 3) if the system has any timers, the unit of time needs to be defined. Normally, one unit equals 1 millisecond or one unit equals 1 second are suitable values. It is also useful to define SDL **synonyms** for useful multipliers such as "seconds" and "minutes".

Example:

- subclause 6.2.1.1 illustrates this step.

5.3.2 Step S:2 Discrete system communications

Instructions:

- 1) identify entities outside of the system whose communication with a system is a topic of specification;
- 2) identify the information flow in terms of discrete messages to be communicated between the system and each outside communicating entity;
- 3) model these messages by **signals** defined in the system;
- 4) state the relation between each **signal** and entities external to the system;
- 5) state the purpose for each **signal** in a comment with the **signal** definition;
- 6) place related (often all) **signals** for one entity in one direction into a **signallist**;
- 7) include the **signal** definitions and **signallists** in the system diagram.

Rules:

Rule 3 The textual definitions of a diagram should be placed in text symbols inside diagrams.

NOTE: Placing the SDL text in a text symbol on a diagram clearly identifies the text as SDL and also identifies the SDL diagram where the text belongs.

Rule 4 Each type of definition (for example: **signal** definitions, **signallist** definitions, data definitions, etc.) should be placed in a different text symbol. If textual definitions occupy more than 50 % of a diagram, it should have separate pages for each type of definition.

Guidelines:

- 1) the number of **signals** will be smaller if related events are communicated with one **signal** with parameters (for example: one can introduce one **signal** for each error cause or one **signal** for all of them with a parameter which specifies which error cause occurred);
- 2) when a **signal** with a parameter is introduced, identify or name a corresponding sort of data. The sort of data should correspond to a named ASN.1 or SDL type;

- 3) the **signal** definitions and **signallist** definitions are usually too large to include in a text symbol on the first SDL page of the system diagram. Additional pages should be added to the system diagram to contain these textual descriptions. Lengthy statements about the relationship between **signals** and external entities, or descriptions that contain informal drawings should not be placed in the SDL diagram. Such statements should be referenced from the SDL. Shorter statements should be placed with the appropriate SDL definitions;
- 4) to assist documentation, **signallist** definitions are placed before the definition of **signals** used in the **signallist**. Data definitions are placed after **signal** definitions. The definitions are grouped by placing related definitions in one text symbol.

Example:

- subclause 6.2.1.2 illustrates this step.

5.3.3 Step S:3 System parts

Instructions:

- 1) identify the main parts within the system and draw them as blocks in the system;
- 2) find a suitable name for each block and describe the block and its relation to its environment (its enclosing structure) informally in a comment within the block.

NOTE: Steps S:3 to S:6 are used repeatedly when a block is decomposed into further blocks. When the steps are re-applied, the "system" being considered is actually an SDL block, and the word "system" should be considered replaced by the phrase "enclosing block". The difference between a block and the system is that the **signals (signallists and data)** used to communicate with the surrounding structure are defined *outside* a block but *inside* a system.

Rules:

Rule 5 There should be no more than five blocks at the system level (or directly enclosed within a block).

Rule 6 A definition should have the smallest scope that includes all uses of the defined item.

Rule 7 If a block (process or procedure) is informative and is not part of an enclosed informative block (process or procedure), it should have the annotation "informative" in the diagram referencing it or in its referenced diagram or in both places.

Guidelines:

- 1) blocks are entities that contain a behaviour aspect with internal states;
- 2) blocks delimit visibility. For this reason, **signals**, sorts and types that are used only within a block should be defined within that block, so that information is hidden from higher levels and therefore makes these levels easier to understand. This principle also applies for steps S:6 and S:7 and is expressed as a general principle for information hiding in rule 6;
- 3) a block is "informative" if it has no behaviour that is to be normative. The purpose of an informative block is to make the system complete, so that the function of the system can be:
 - understood by its use of and interaction with these informal parts;
 - executed and analysed, leading to a better quality specification and supporting validation.
 A block can be informative only if all the enclosed blocks, processes and procedures (including remote procedures) are informative. Once a block (process or procedure) is marked informative it is implied that all the enclosed blocks, processes and procedures are informative and it is not necessary to mark these blocks, processes and procedures as "informative".

NOTE: "Informative" is not the same as "informal". In the formal SDL description both informative and normative parts should be formal (that is, expressed formally).

- 4) when there are a large number of blocks directly enclosed within the system (or a block) some of them can be grouped together and encapsulated in a block as in step S:6.

Example:

- subclause 6.2.1.3 illustrates this step.

5.3.4 Step S:4 Communication paths between parts

Instructions:

- 1) identify the channels needed between blocks and the boundary of the diagram and between blocks within the diagram;
- 2) for each channel, identify the direction(s) of communication;
- 3) associate a **signallist** with each direction of the channel;
- 4) choose a **signallist** name related to the function and usage of the communication.

Rules:

Rule 8 There should be only one channel between two blocks.

Rule 9 Every channel that is normative should have the comment "normative" attached.

Guidelines:

- 1) the **signallists** identified in step S:2 correspond to one or more channels leading to a point on the system boundary. What from the outside should be regarded as one interface, may be more than one channel coming from different blocks. Also there can be more than one **signallist** per channel. Each point on the boundary represents the communication with a different external entity. There can be one or more channels connecting such a point to the blocks in the SDL system diagram. Realistic modelling requirements such as independent interfaces are taken into consideration;
- 2) blocks are connected to each other and/or to the environment by channels according to the flows of information. The system may contain a single block or more than one but there are no good reasons for a block diagram to contain a single block;
- 3) typical communication cases represented in MSC help to identify the channels;
- 4) special effects that depend on the delay or non delay of each channel should not normally be used. If only one channel is used between two blocks then **signals** sent to one block arrive in the same order at the other block. Channels should be assumed to have a delay, unless there are requirements to communicate without delay;
- 5) if the communication on a channel needs to have specific messages with a specific format and encoding, the channel is "normative". Channels that are internal to the system and do not identify a particular interface for product testing or other purposes are usually informative. The communication on informative channels contributes to the behaviour of the system, but there could be an alternative system with different channels or communication which has the same behaviour.

Example:

- subclause 6.2.1.4 illustrates this step.

5.3.5 Step S:5 Associating signals to communication paths

Instructions:

- 1) for each **signallist** name, identify the appropriate **signals**;
- 2) name and define each new **signal**.

Rules:

Rule 10 No more than three **signals** (or **signallists**) should be listed in a **signallist** symbol, instead use a **signallist** attached to the channel. In principle it is better to use **signallists** consistently.

Rule 11 The **signallists**, **signals** and data used in all communication of a system should be defined in one (or more) text symbol(s) separate from other definitions.

Guidelines:

- 1) although SDL allows the **signal** list symbol (that is [...]) associated with a channel to contain explicit lists of **signal** names, this is not recommended. A list of **signals** is usually needed in more than one place, and it is easier to modify the list if it is defined once in a **signallist**;
- 2) consider whether to redefine the **signallist** associated with a channel at higher levels, making use of a new **signallist**. This can improve the structure and clarity of the SDL. The **signallist** definition needs to be within the highest level diagram in which it is used;
- 3) if there are new **signals** used for communication between blocks, they need to be defined in the unit (system or block) that contains communicating blocks.

Example:

- subclause 6.2.1.5 illustrates this step.

5.3.6 Step S:6 Information hiding and sub-structuring

Instructions:

- 1) consider each block within the current diagram in turn and decide whether the block should contain blocks or processes;
- 2) if the block is to contain blocks, recursively apply the sequence of steps S:3 to S:6 to the block, regarding it as a (sub) system on its own and introducing new required **signals**, blocks, channels and **signallist**;
- 3) otherwise, apply step S:7 to divide the block into processes.

Rules:

Rule 12 The diagrams should be nested by reference rather than direct enclosure.

Rule 13 The number of channels from each block should be no more than five.

Guidelines:

- 1) if the system is large, some blocks can be considered a system on their own, and can be further partitioned according to the rules given for the system. This results in nesting of blocks. Each block can then be elaborated as described above. An entity that has behaviour as the main aspect and is not further partitioned is a good candidate to be a process; therefore, the directly enclosing entity is a block. An entity that contains further partitioned entities is usually a block to be partitioned;
- 2) it may not be clear whether the contents of a block should be blocks or processes, in which case a division into blocks is initially attempted. If it is difficult to partition a block further, it should probably be a process;
- 3) recursive application of this step results in a number of levels. Even if the system is complex there should not be more than three levels of blocks. For example, if each level has 3 to 4 blocks with 2 to 5 processes in each leaf block, then there will be over 100 processes in the whole SDL system;
- 4) if there are more than five obvious process definitions within a block, then the block should be divided into two or more blocks with fewer processes. The interactions in MSC between axes corresponding to processes can help to identify natural "clusters" of processes for each unpartitioned block;
- 5) a block tree diagram should be drawn.

Example:

- subclause 6.2.1.6 illustrates this step.

5.3.7 Step S:7 Block constituents

Instructions:

- 1) identify the separate entities with behaviour for the block that has been chosen in step S:6 to be divided into processes and define these entities as the processes of the block;
- 2) find a suitable name for each process and describe it and its relation to its environment (the enclosing block) informally in a comment within the process reference;
- 3) for each process define its initial and maximum number of instances;
- 4) use **signal** routes to connect the process sets to channels at the block boundary.

Rules:

Rule 14 For each block, at least one process should have its initial number of instances greater than zero, so that it can create other instances in the block.

Rule 15 The number of process definitions in each block should be no more than five.

Guidelines:

- 1) the **signal** routes within the block can be derived in a similar way to the channels in steps S:3 and S:4.
- 2) in the case where a process has been encapsulated in a "dummy" block, the unpartitioned block contains a process description of just one type and the relation to its environment is derived from the external interface of the block.

Example:

- subclause 6.2.1.7 illustrates this step.

5.3.8 Step S:8 Local signals in a block

Instructions:

- 1) identify the **signals** between the processes local to the block;
- 2) define at the block level any of these **signals** that are additional (not defined external to the block);
- 3) identify any imported and exported procedure of processes in the block and the corresponding remote procedure definition, and define these at the block (or block type) level.

Guideline:

- 1) In the special case with a "dummy" block, there will be no additional **signals** at the block level.

NOTE: A "dummy block" is the one containing only one process.

Example:

- subclause 6.2.1.8 illustrates this step.

5.4 Behaviour steps (B-steps)

These steps are to describe the behaviour of components in terms of communication, but without providing a formal definition of the data covered by data steps. This subclause describes these steps for an SDL process, but both these steps and the data steps are also generally appropriate for defining the behaviour of an SDL procedure.

5.4.1 Step B:1 Set of signals to a process

Instructions:

- 1) Identify the input alphabet of the process, called the signalset of the process, which is done by identifying any:
 - **signals** that can be received by the process;
 - exported procedure of the process. This is a case where the process acts as a server in a client-server model where the corresponding **signal** is implicit but the remote procedure name can be considered as part of the alphabet;
 - timer expirations are received in the same manner as ordinary **signals**. Timers belong to a process and must be declared in text symbols of the process. The duration to which a timer is set should be given a symbolic name using a **synonym** or a variable (of duration type).

Guidelines:

- 1) although the signalset can usually be simply derived from the enclosing block diagram, the purpose of this step is to review the signalset considering only the current process. If it is decided to change the set of **signals** from other processes, the corresponding block diagrams must be changed. The processes that send the **signals** will need to be updated (probably at some other time), if they have already been formalized. One source of **signals** not shown on the block diagram is the process sending **signals** to itself or other instances of the same process definition;
- 2) the signalset is used when deciding the behaviour of the process in each state in step B:4. It is suggested that a record is kept of the signalset if this cannot be derived automatically from tools used for SDL;
- 3) for a procedure the **signals** of the enclosing process are used, but new **signals** may be identified, that need to be added to the process signalset (identified in this step but for the enclosing process).

Example:

- subclause 6.2.2.1 illustrates this step.

5.4.2 Step B:2 Skeleton processes

Instructions:

- 1) produce MSC for at least the "typical" use cases. This is very important both for identifying **signals** and describing basic behaviour;
- 2) produce a skeleton process by mapping from MSC considering only "typical" uses:
 - 2.1 starting from the start symbol of the process, build a tree of states by considering "normal" state changes of the process;
 - 2.2 build the process tree by branching at each state based on each input that is consumed but not ignored in the MSC and following each by a transition (including outputs and creates) to different states;
 - 2.3 identify a state as different from other states if it has a different set of **signals** that it consumes or saves or if it has a different response to a **signal**;
 - 2.4 include time supervision (set and reset) and the corresponding timer input;
 - 2.5 as the tree is drawn, identify where the process returns to the same state and make the tree into a graph.
- 3) draw this graph as a process diagram;
- 4) determine whether the process has two or more disjoint sets of interfaces for different behaviour parts of the process that can be interleaved, then — if the behaviours are independent — divide the process into multiple processes.

NOTE: If the behaviours are coupled by the use of common data, dividing the process into processes requires one or more remote procedures to access the data.

Rules:

Rule 16 Spontaneous transition should not be used in normative parts of the standard.

Rule 17 The MSC should either be correct traces of the handling of messages by the SDL system or be clearly annotated to indicate how and why they differ from the SDL behaviour.

Guidelines:

- 1) two processes with N and M states respectively, when combined as one process has $N \times M$ states. Splitting such a process therefore produces two much simpler processes that are easier to understand;
- 2) when a process (or procedure) is informative, there may be spontaneous transitions starting with **none** that model user behaviour or other unpredictable events.

Example:

- subclause 6.2.2.2 illustrates this step.

5.4.3 Step B:3 Informal processes

Instructions:

- 1) identify combinations of use cases;
- 2) identify what information the process stores and consider whether this is implicit in the process states or whether internal data is needed;
- 3) use this information to define the actions of each process;
- 4) add tasks or procedures and possibly more decisions in transitions, but use only informal-text in tasks and decisions;
- 5) if a procedure is used, give it a meaningful name and (later) use steps B:1 to D:9 to define it.

Guidelines:

- 1) use existing MSC and generate new MSC to identify combinations of uses;
- 2) decide whether to use a task or procedure to describe the information processing in a transition, although this may be changed later. Use a procedure if it is anticipated that the information is needed from another process or if the task is likely to be complex or to depend on several stored data values; otherwise choose a task. If a procedure is chosen it may be appropriate to consider the parameters (one or more parameters with defined data sorts);
- 3) sometimes it is obvious that the same actions are done in several places in the process. These actions can be collected and made into a procedure;
- 4) if procedure has states, it should be kept in mind that only **signals** mentioned in the state are dealt with explicitly, all other **signals** to a process that has called a procedure may be implicitly consumed if not mentioned in a save symbol.

Example:

- subclause 6.2.2.3 illustrates this step.

5.4.4 Step B:4 Complete processes

Instructions:

- 1) identify at each state and for each item in the signalset (**signal** or remote procedure) whether the **signal** (or remote procedure call) is input by the process or saved;
- 2) if the item is input, determine the transition taken (it may be an implicit transition back to the same state);
- 3) continue instruction 1 and instruction 2 until there are no states at the end of a transition that have not been considered so that all items in the signalset have been considered for every state;
- 4) analyse first each process and then combinations up to the whole SDL system to check for unwanted properties and redesign to avoid them if necessary.

Rules:

Rule 18 All states in a process should be reachable from the start of the process.

Rule 19 A procedure that is exported by a process (to be used as a remote procedure) should not be saved in every state of that process.

Rule 20 Each **signal** received by the process should have at least one input leading to a non empty transition.

Guidelines:

- 1) a state **signal** matrix (in ITU-T Recommendation Z.100 [3] appendix I.9.4) can be used to check the action for each **signal** in each state. As a new state is identified, the matrix is extended. This matrix can also help identify when two of the defined states lead to the same behaviour and can be combined, or two **signals** have the same next states. In these cases it may be possible to reduce the number of **signals** or states;
- 2) a state overview diagram (in ITU-T Recommendation Z.100 [3] appendix I.9.3) can be drawn to get an overview of the behaviour of the process. If the process is not normally intended to terminate then usually every state should be reachable from every other state, but this need not be the case as there may be initial states for the start up of the process and final states for termination;

- 3) there is a limit to how much analysis is practical on a single process. More interesting results from analysis are obtained as more processes are "complete" can be analysed in combination in a block or as the whole system. The unwanted properties that can be detected (such as unreachable states, deadlock and live-lock) will depend on the complexity of the system and the tools available;
- 4) for analysis it can be assumed that every decision contains the SDL **any** value construct.

NOTE: Analysis of anything other than a simple process is difficult without tools to generate the state space and then check it. Even tools have difficulty with large numbers of processes or a very complex process. This is therefore not a trivial step, but analysis at this stage saves a lot of wasted time and effort if redesign is found necessary and is one of the major benefits of using SDL.

Example:

- subclause 6.2.2.4 illustrates this step.

5.5 Data steps (D-steps)

The purpose of the data steps is to provide a formal definition of the data used in the processes. Without formal data any decision in the behaviour and operators used in expressions have uncertain results.

The first two steps (D:1 to D:2) concern interface values. The next three steps (D:3 to D:5) concern variables within processes. The remaining steps (D:6 to D:9) concern the formal definition of new sorts of data. These last steps are needed if new sorts of data have new operators. For this reason D:6 to D:9 are only occasionally necessary and should be unnecessary if ITU-T Recommendation X.208 [9] has been used to define data.

Step D:1 is applied to the system as a whole, whereas steps D:2 to D:9 can be applied to each process or procedure.

The sorts of data used should be defined using SDL combined with ASN.1 as defined in ITU-T Recommendation Z.105 [4].

5.5.1 Step D:1 Signal parameters

Instructions:

- 1) identify the values to be conveyed by **signals**, starting with **signals** at the system level;
- 2) look for predefined sorts of data to represent the identified values;
- 3) extend the **signal** definitions with the sort of data;
- 4) identify and define new sorts of data if necessary;
- 5) timer expiration **signals** may have parameters like any other **signal**. Sort of parameters must be defined when timers are defined.

Rules:

Rule 21 If bit tables have been used to define data then these should be converted to ITU-T Recommendation X.208 [9] or ITU-T Recommendation Z.100 [3].

Rule 22 The names of existing sorts of data should not be used for new sorts of data even if they have different scopes.

NOTE: If the sorts of data have the same scope, SDL language rules do not allow the names to be the same.

Guideline:

- 1) ITU-T Recommendation X.208 [9] provides a formal description of the data used for **signals**, and therefore the corresponding sort of data can be used directly. If ASN.1 has not been used, a decision must be made whether to define the sorts of data in ASN.1 or ITU-T Recommendation Z.100 [3]. ASN.1 should be used if the encoding of data is important or if the data is on an interface with the environment. SDL may be used if no particular encoding is required and if the sort of data is used only within the SDL system.

Example:

- subclause 6.2.3.1 illustrates this step.

5.5.2 Step D:2 Process and procedure parameters

Instructions:

- 1) identify the sorts of data needed for parameters of the process (or procedure);
- 2) state the role of each parameter in a comment in the heading of the process (or procedure);
- 3) procedure parameters can be defined as IN or IN/OUT. IN should be used to pass values from the calling process to the called procedure. IN/OUT are used to pass values in both directions.

Rule:

Rule 23 A process parameter should not contain the PId of the process that has created it because this value is returned by **parent** operator.

Guidelines:

- 1) within a process, a parameter is treated as a variable. The only difference between a process parameter and a variable with a default value is that a process parameter can receive a different value for each instance of the process. Typically this value is the identity of some other entity such as a PId or an equipment number;
- 2) process parameters are often needed to establish information necessary for addressing of **signals**. The creator of the process sends as a parameter the information about the requesting process, so that the created process knows whom to send a **signal** if needed.

Example:

- subclause 6.2.3.2 illustrates this step.

5.5.3 Step D:3 Signal variables

Instructions:

- 1) add parameters to inputs according to **signal** definitions;
- 2) define variables as required to receive the input values;
- 3) state the role of each variable in a comment.

Rule:

Rule 24 A **signal** parameter should not contain the PId of the process that has sent the **signal** because this is value is returned by **sender** operator.

Guidelines:

- 1) check that each parameter defined in a **signal** definition is used in at least one input or is required for communication with the environment;
- 2) with each **signal** reception the value returned by **sender** operator may change. Sometimes it may be necessary to save the PId of the communicating partner.

Example:

- subclause 6.2.3.3 illustrates this step.

5.5.4 Step D:4 Formal transitions

Instructions:

- 1) replace the informal text in tasks, decisions and answers with formal assignments, formal expressions, formal range expressions and procedure calls and identify any new operators used in the formal expressions to be added to the sorts of data in step D:6;
- 2) define additional variables and synonyms as required;
- 3) define additional procedures as required;
- 4) add parameters to procedure calls according to procedure definitions.

Rule:

Rule 25 A value or decision that is non-deterministic (using **any**) should always have a comment attached explaining how the choice is made.

Guidelines:

- 1) the previous informal text in symbols may be useful as comments attached to the symbols;
- 2) if the value of a function used in an expression depends only on the actual parameters, there is a choice of making the function an operator or a procedure. If the result depends on other data, it must be a procedure. If the function behaviour depends on data from another process, it may be appropriate to make it a remote procedure or decide how this information is obtained. To obtain the information may require additional parameters to existing **signals** and storing this information or communication in the procedure, possibly with additional **signals**;
- 3) the definition of a procedure is treated in a similar way to a process through steps B:1 to D:9, including the possibilities of having a procedure called internally and a procedure nested within the procedure;
- 4) when the normative behaviour is independent of some data in informative processes or procedures, the **any** construct can be used to create random values for this data or random decisions in informative processes or procedures. In this way the informative parts can model situations where data and behaviour are unpredictable. Spontaneous transitions are used if the time ("when") something happens is unpredictable, and any value is used if what happens is unpredictable;
- 5) the PId of the newly created process is available in the creating process as this value is returned by **offspring** operator. In most cases this value must be saved because it would be lost as new instances get created. Also, the value returned can be NULL if creation was not successful;
- 6) synonyms are a means of giving a symbolic name to values used in various places (parameters, timers, decisions etc.). Using values directly should be strictly avoided;
- 7) in the process that creates other process instances it may be necessary to maintain the list of created processes PIds.

Example:

- subclause 6.2.3.4 illustrates this step.

5.5.5 Step D:5 Output and create parameters

Instructions:

- 1) add expressions to outputs using introduced variables and synonyms;
- 2) add addressing information to every output;
- 3) add actual parameters to create actions.

Rules:

Rule 26 If a parameter is omitted in an output, there should not be a corresponding input that expects a value for this parameter.

Rule 27 Addressing information should always be defined in an output.

Guidelines:

- 1) check that each parameter sent in an output is used in at least one possible input or is required for communication with the environment. As compared with the check that a **signal** definition parameter is used in step D:3, this is a check that a **signal** instance parameter may be used;
- 2) addressing should be done using the keyword **TO** if **signal** is sent to a particular process instance identified with its PId given after the keyword. The PId value can be obtained from some variable of the type PId where it was previously stored or by applying operators **sender**, **parent**, **offspring** or **self**. These operators return the PId of the process instance from which a **signal** was received, the PId of the creator that created this process instance, the PId of the latest process instance created or the PId of the process instance itself, respectively. The operator **self** is used for sending **signals** to its own input queue, either from the body of the process or from its procedures. Instead of an PId expression a process name may be mentioned, but only if the process has one and only one instance, otherwise the **signal** is delivered to an unspecified process instance;
- 3) instead of specifying addresses using a keyword **TO** a keyword **VIA** could be used. This keyword is to be followed by a channel, **signalroute** or **gate** identifier. This restricts the choice of entities that can receive a **signal** and is often used when the communication between two peer entities is being established;
- 4) if expression **VIA ALL** is used **signals** are sent to all possible destinations.

Example:

- subclause 6.2.3.5 illustrates this step.

5.5.6 Step D:6 Data signatures

Instructions:

- 1) identify and define all the sorts of data and values (synonyms) which need to be defined;
- 2) if some values are identified as dependent on the actual system installation, express them by using external synonyms;
- 3) for each required sort of data create a **newtype** or **syntype** with a meaningful name or define an ASN.1 type;
- 4) identify any new operators that need to be defined;

NOTE: If there are no new operators the formalization process is complete.

- 5) for the **newtypes** with new operators, list the signatures (the literals and the operators with parameter sorts) and identify (in a comment) a set of operators and literals that can be used to represent all possible values (the "constructors").

Guidelines:

- 1) external synonyms can be used to provide dimensions for the number of processes, the number of data items in an array, etc. An external declared **synonym** will get its value at system start-up and the transition options expression will be tested at system creation time;
- 2) it is recommended that option transitions and selects be avoided. If it is really necessary to provide a choice in an ETSI specification, some guidelines for specifying optional functionality and alternative behaviour are given in ETS 300 414 [1] annex C clauses 4 and 5;

- 3) inherit as much as possible from ITU-T Recommendation Z.100 [3] predefined data, ASN.1 "useful types" and ITU-T Recommendation Z.105 [4] predefined data. The objective is to avoid having to define the behaviour for new operators. If new operators are defined, they should be defined using an operator definition. The introduction of new operators can be avoided by:
 - using the ASN.1 definitions as defined in ITU-T Recommendation Z.105 [4];
 - using struct for records;
 - using a predefined generator such as array or string;
 - using a syntype if the set of data values is intended to be a subset of and compatible with a sort of data;
 - giving a sort a new name with syntype if the purpose is to introduce a more meaningful name;
 - giving a sort a new name with newtype which inherits all if the purpose is to have a sort of data with the same properties as an existing sort of data, but not compatible with the existing sort;
 - using a user defined generator that avoids defining axioms by using other generators.
- 4) the operators are usually listed in the newtype for the sort of data corresponding to the result of the operator. Sometimes an operator produces a value of a sort of data defined in a different context, for example an integer or a Boolean. In this case the operator is listed under the newtype of (one of) the parameter sorts of data;
- 5) a set of operators and literals that can be used to represent all possible values is a set of constructors of the sort of data. These constructors are used in later steps if (and only if) it is necessary to define operator properties using SDL axioms. The set of constructors is not usually unique, nor is it always obvious. The chosen set should be just sufficient to define all values so that if one constructor is deleted then the set of values is different. The choice can be difficult;
- 6) although this step produces a formal SDL description, if new operators are introduced, the functionality may not be what was intended, because the user defined sorts of data for these operators may have "too many" values. Step D:7 corrects this deficiency, but makes interpretation depend on informal text. Steps D:8 and D:9 make the description formal.

Example:

- subclause 6.2.3.6 illustrates this step.

5.5.7 Step D:7 Informal data description

Instruction:

- 1) add informal axioms in the form of informal text to the newtype definitions.

Guidelines:

- 1) the names of the operators should correspond to their function and therefore assist the description of the function;
- 2) although the result is correct SDL, only the static properties can be completely analysed. The reason is that the dynamic properties depend on the interpretation of the axioms, which can only be done informally. However, in an actual support environment, some additional features or assumptions can make this level of description sufficient.

Example:

- subclause 6.2.3.7 illustrates this step.

5.5.8 Step D:8 Formal data description

Instructions:

- 1) for each operator signature add an operator definition in the form of an operator diagram if possible;
NOTE: If all the operators can be defined in this way, the formalization is complete.
- 2) if some of the operators cannot be defined by operator diagrams, formalize the axioms by replacing informal axioms with formal ones that state the essential properties of the operators;
- 3) use the text of informal axioms as comments to the formal axioms.

Rule:

Rule 28 SDL axioms should not be used to define operator properties.

Guideline:

- 1) Operators can be defined algorithmically using an operator diagram or axiomatically. The operator diagram (algorithmic) form is recommended because it is easier to understand.

Example:

- subclause 6.2.3.8 illustrates this step.

5.5.9 Step D:9 Complete data formalization

This step should be used only in exceptional circumstances.

Instruction:

- 1) add axioms (or operator definitions) to the data newtype definitions, until they are complete (that is, until all expressions containing non-constructor operators and literals can be rewritten into expressions containing only constructor operators and literals).

Guideline:

- 1) detailed guidelines for this step are given in ITU-T Recommendation Z.100 [3] annex I clause 5.

Example:

- subclause 6.2.3.9 illustrates this step.

5.6 Results of formalization

When formalization has been done, there is a set of results as defined below. These should be reviewed to determine whether formalization can be considered complete according to a checklist of criteria defined in the next subclause.

A complete model provides a complete requirements specification of the system.

Some of the formalization steps are intended to produce an executable model, except possibly for some sorts of data that can usually be made executable by using a support tool interactively or with this data in a programming language. The benefit of an executable model is that by running the model the feasibility and functionality of system can be demonstrated. The disadvantage of this model is that some aspects of the system have to be explicitly defined so that the system is executable. For example, data passed through the system without modification has to be modelled in some explicit way (for example a character string) so that the model can be executed, but for the ETSI specification this may be the abstract concept of a data unit.

The SDL model is executable so that execution of the model can be used in validation and the model can run against conformance tests (using ITU-T Recommendation X.292 [13]) to ensure that these are compatible with the model.

5.7 Criteria for adequate formalization

The result of formalization should be checked against the following criteria:

- 1) the formal SDL description should conform to the rules stated in ETS 300 414 [1], listed also in Annex C of this handbook (which implies conformance to ITU-T Recommendation Z.100 [3], ITU-T Recommendation Z.105 [4], and ITU-T Recommendation Z.120 [6]);
- 2) the SDL should conform to the rules **1** to **28** in this clause;
- 3) it should have been shown (by a thorough design review or audit, and by execution with defined input sequences) that the formal SDL description is a satisfactory model of the functionality of the system described by the collected requirements.

6 Examples

It is more appropriate and cost effective to consider the standard developer education in the formalization process using computer based learning. Thus, the present examples exist in an electronic version and so are all supported by a tool.

The tool may help the standard developer to execute these examples and so to simulate and validate them in order to show him the efficiency of using a formal SDL notation. The simulation may be a motivating activity by which the standard developer can see the examples "living" and how "things" work, detecting how many other "things" can miss.

Incoming Private User Mobility (PUMI) (being developed under ETSI Work Item DE/BTC-01051, "Private Telecommunication Network (PTN); Private User Mobility (PUM) - Call Handling - Additional Network Features - Functional Capabilities and Information Flows") has been chosen as a general example just to illustrate the stepwise formalization process described in clause 5.

6.1 Presentation of the PUMI example

Private User Mobility (PUM) provides Private Integrated Services Network (PISN) users with personal mobility services irrespective of the terminal used within the PISN. It enables each PISN user subscribing to the PUM service to participate in a user-defined set of services and to initiate and receive calls on the basis of a unique, personal PUM number throughout the PISN at any terminal, wired or cordless, irrespective of geographic location. In a PISN supporting PUM there exists no permanent association between PUM users and terminals.

Incoming PUM Call Handling (PUMI) enables calls to be directed to a PUM user within the PISN. As there is no predetermined access for the connection of a PUM user to the PISN, the directing of such calls requires that information regarding the location of the user is available within the PISN.

6.2 PUMI example: the stepwise formalization

6.2.1 Structure-steps (S-steps)

6.2.1.1 Step S:1 Boundary and environment of the system

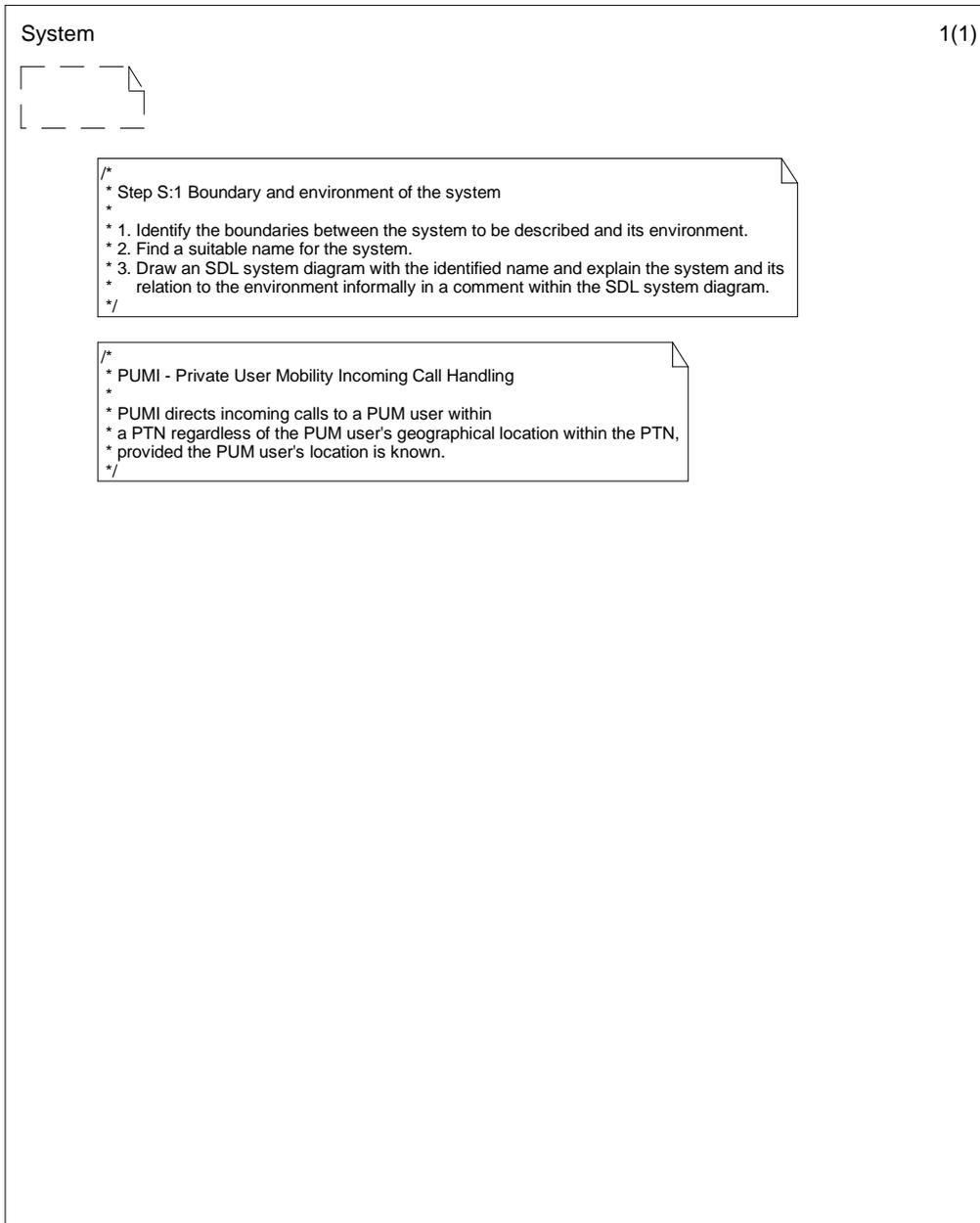


Figure 2: Step S:1 Boundary and environment of the system

6.2.1.2 Step S:2 Discrete system communications

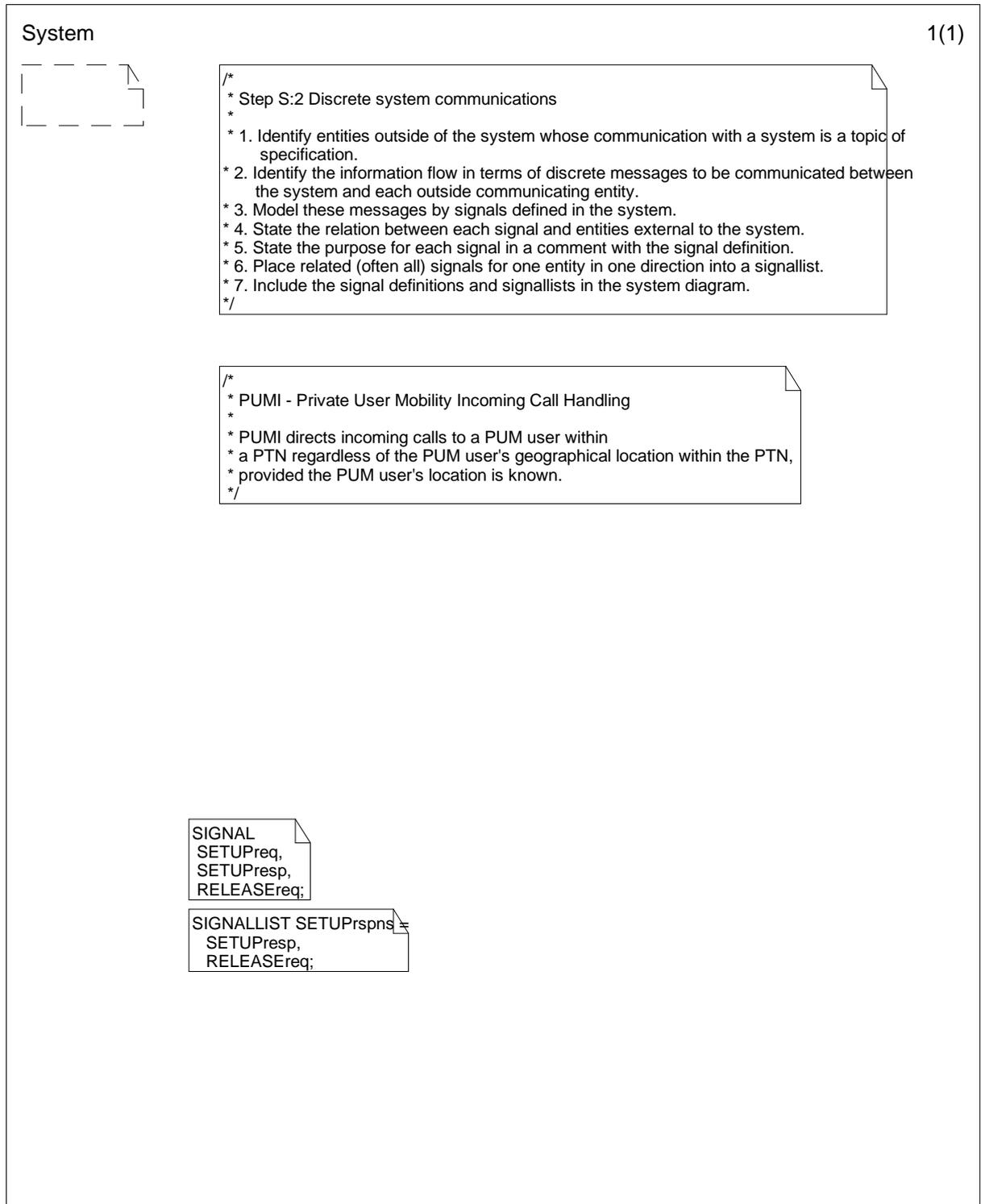


Figure 3: Step S:2 Discrete system communications

6.2.1.3 Step S:3 System parts

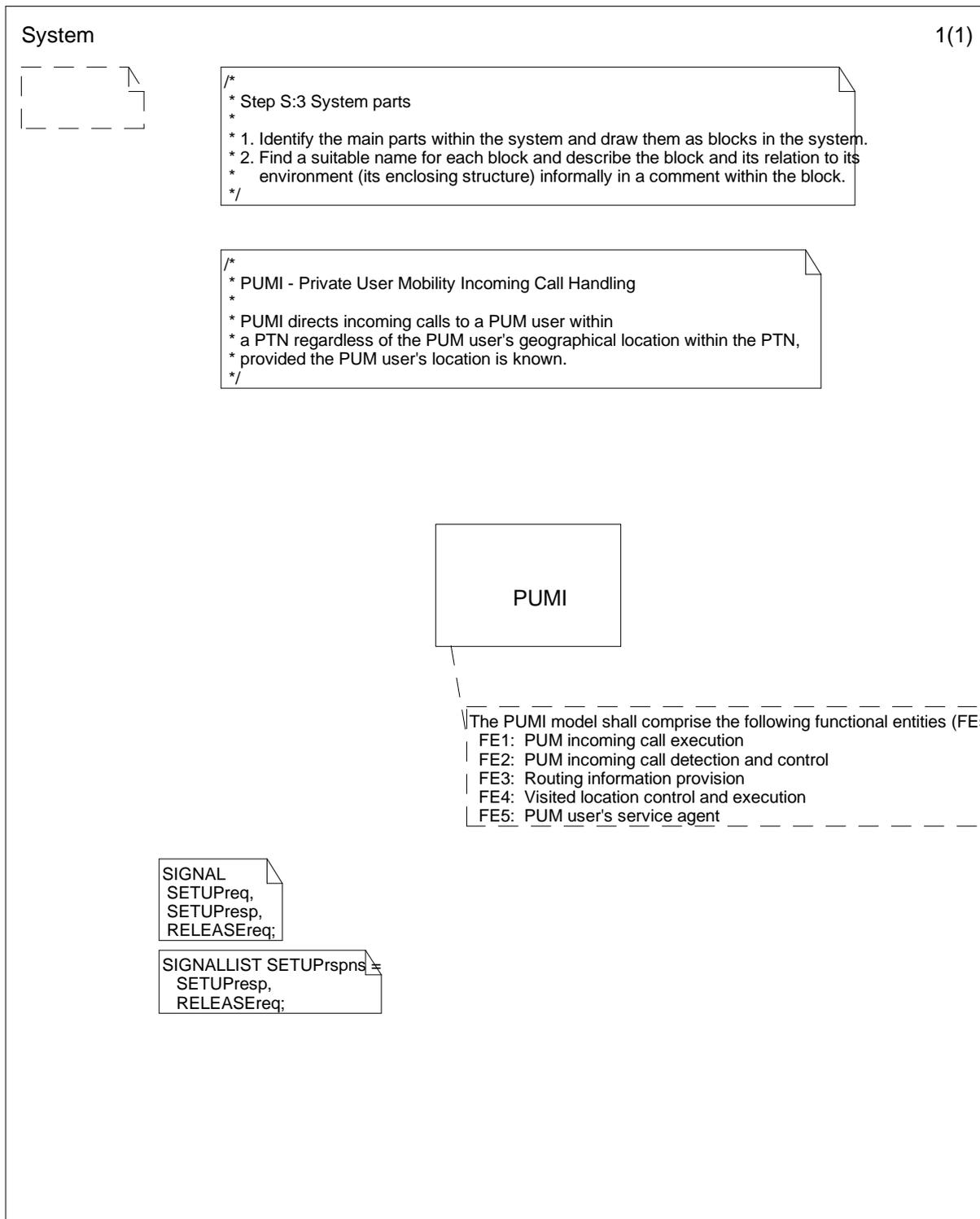


Figure 4: Step S:3 System parts

6.2.1.4 Step S:4 Communication paths between parts

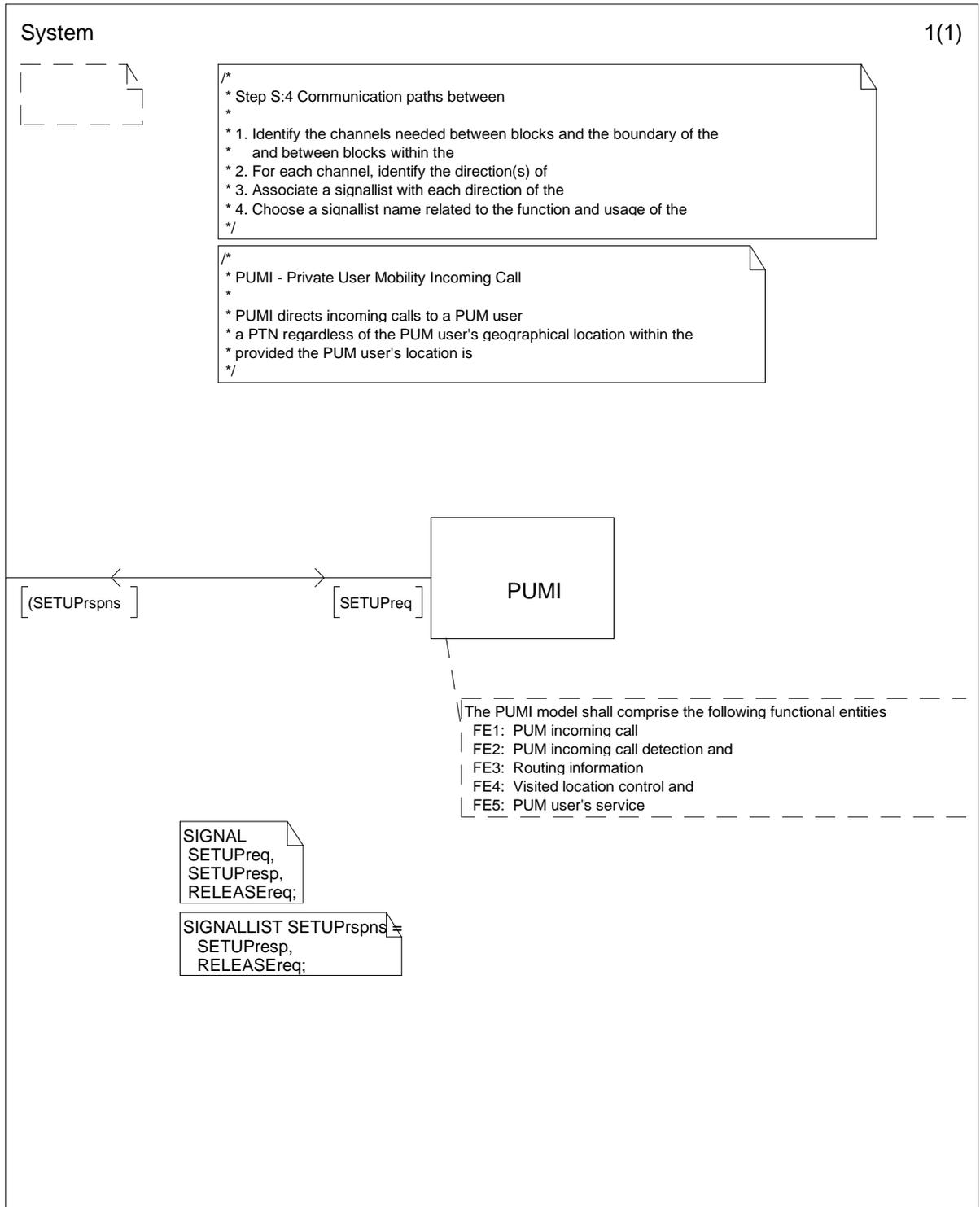


Figure 5: Step S:4 Communication paths between parts

6.2.1.5 Step S:5 Associating signals to communication paths

Step S: 5 is illustrated in subclause 6.2.1.4.

6.2.1.6 Step S:6 Information hiding and sub structuring

This step is not applicable for the PUMI example.

6.2.1.7 Step S:7 Block constituents

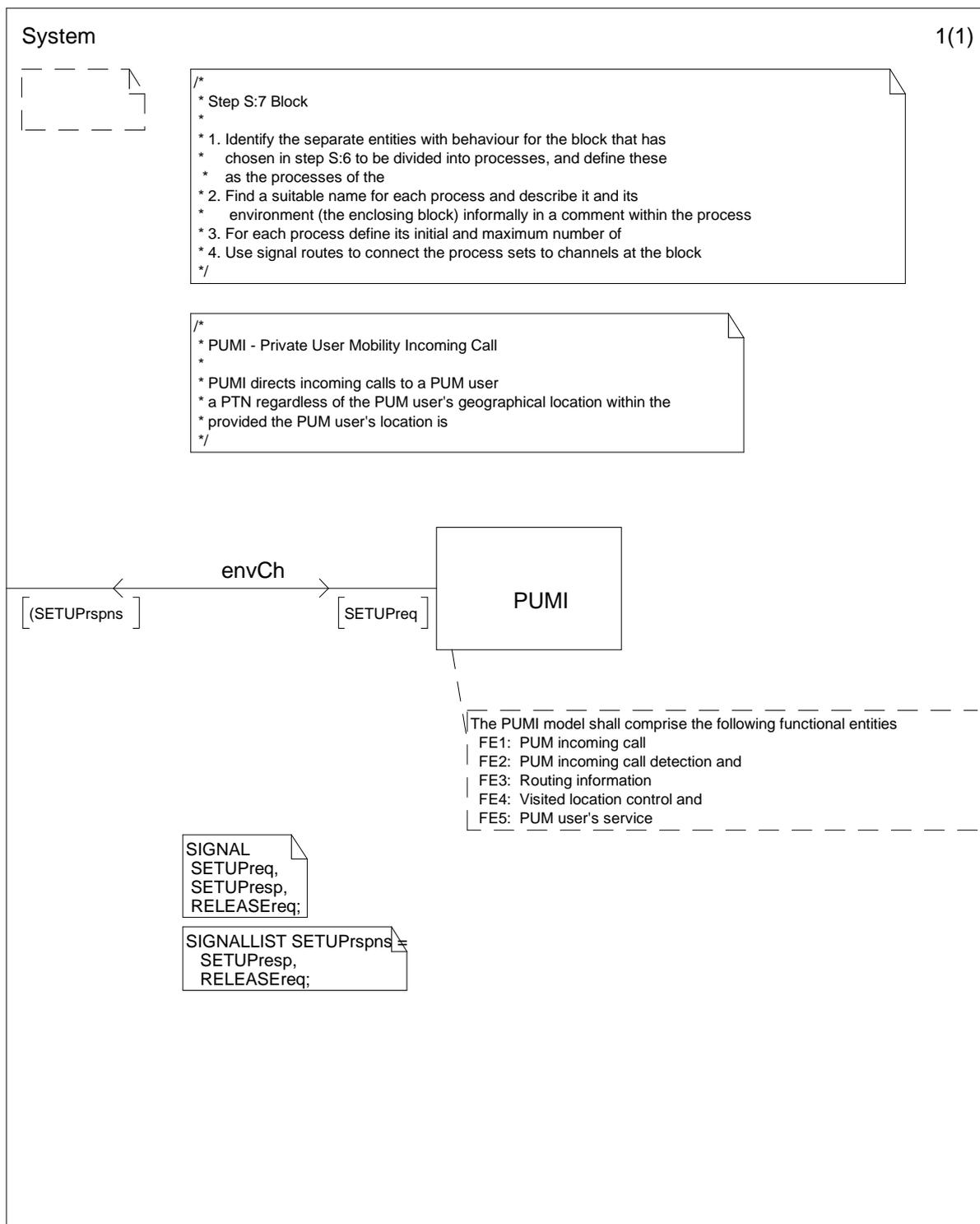


Figure 6: Step S:7 Block constituents

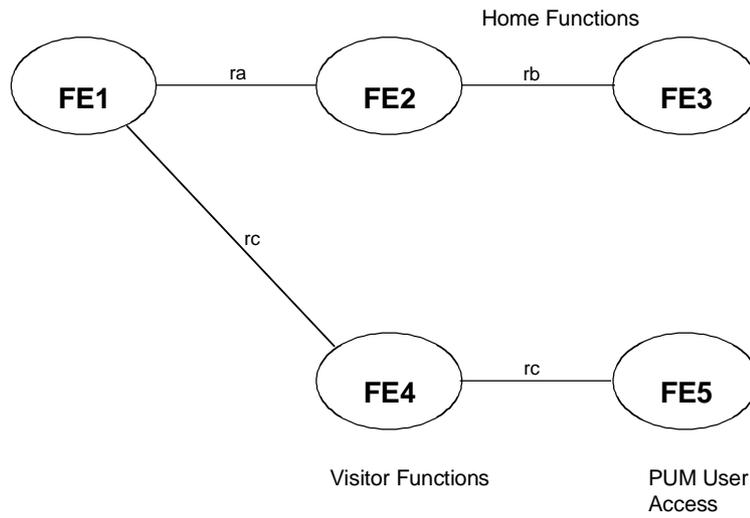


Figure 7: S:7 Functional model for the handling of an incoming call to a PUM user

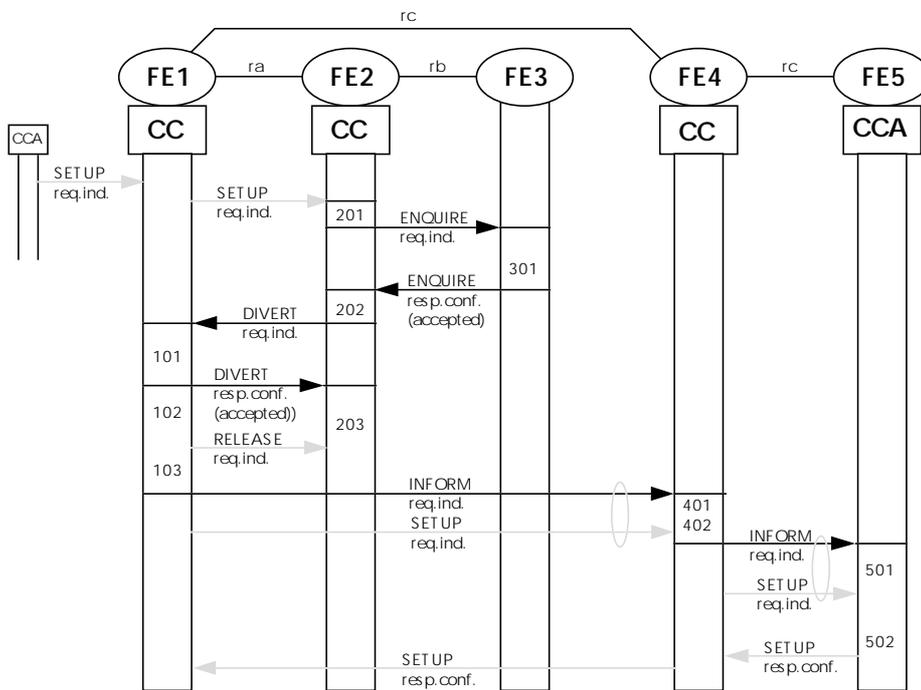


Figure 8: S:7 Information flow sequence for successful PUMI operation

MSC successfulPUMI

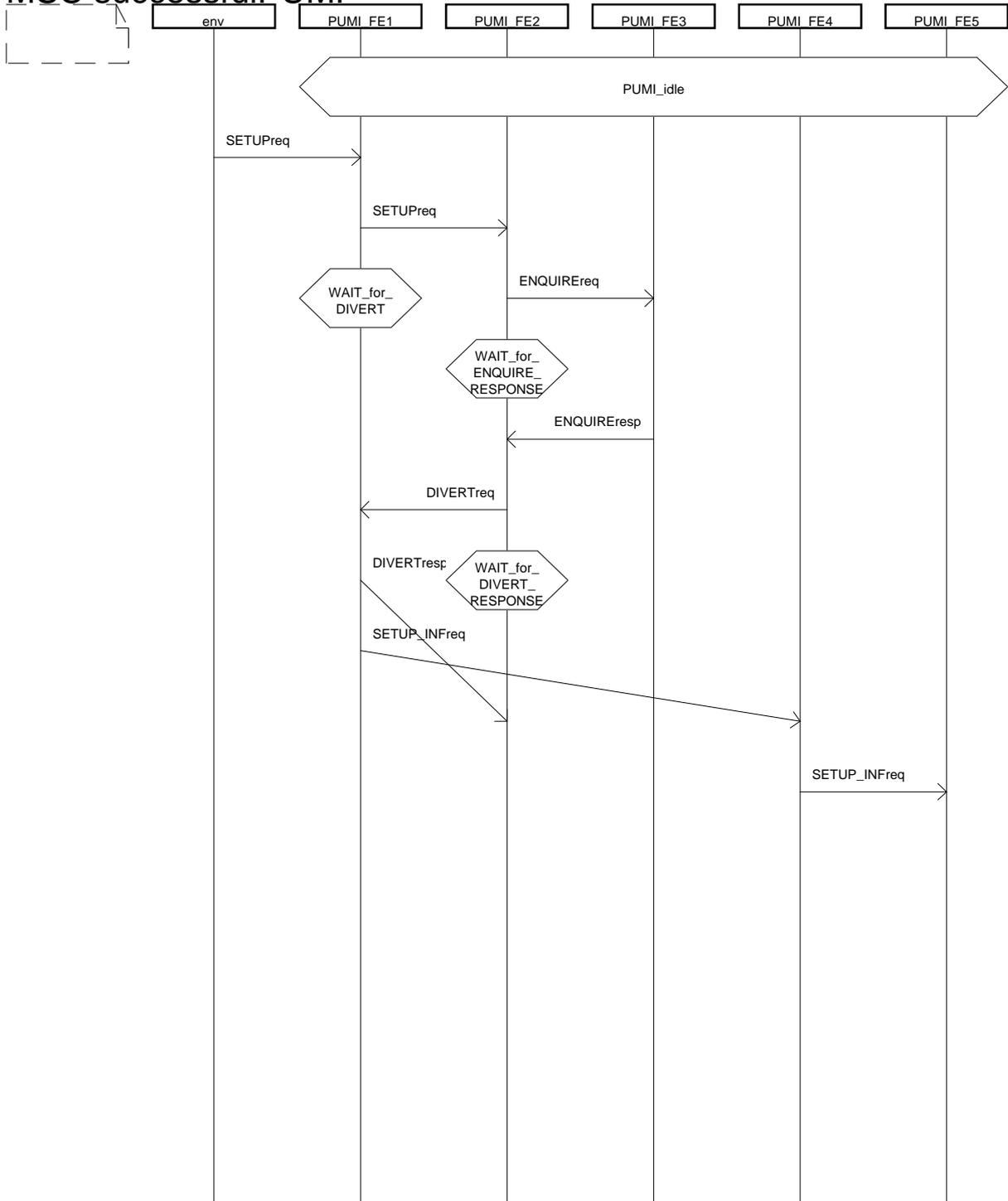


Figure 9: S:7 MSC for successful PUMI operation

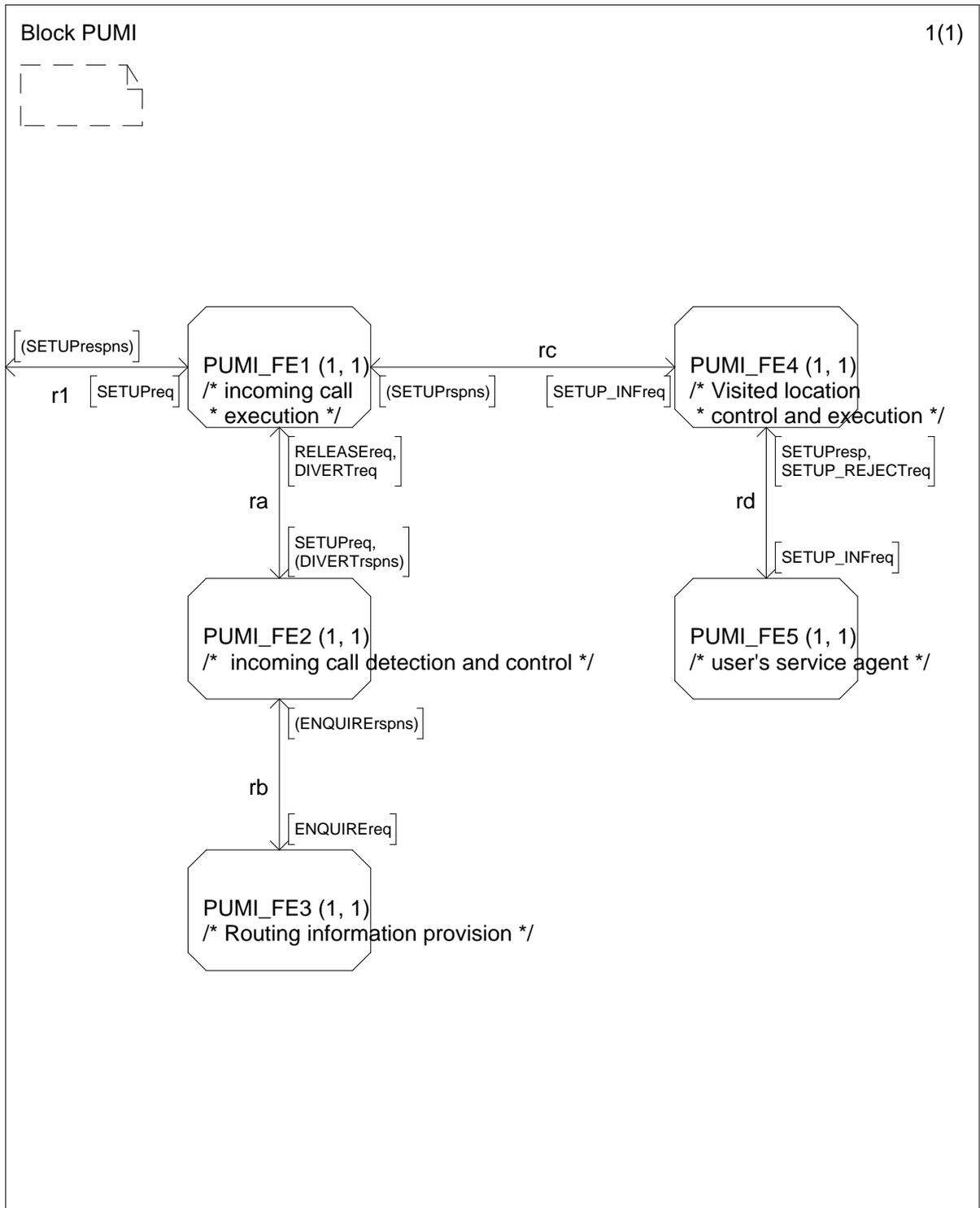


Figure 10: Step S:7 Block PUMI

6.2.1.8 Step S:8 Local signals in a block

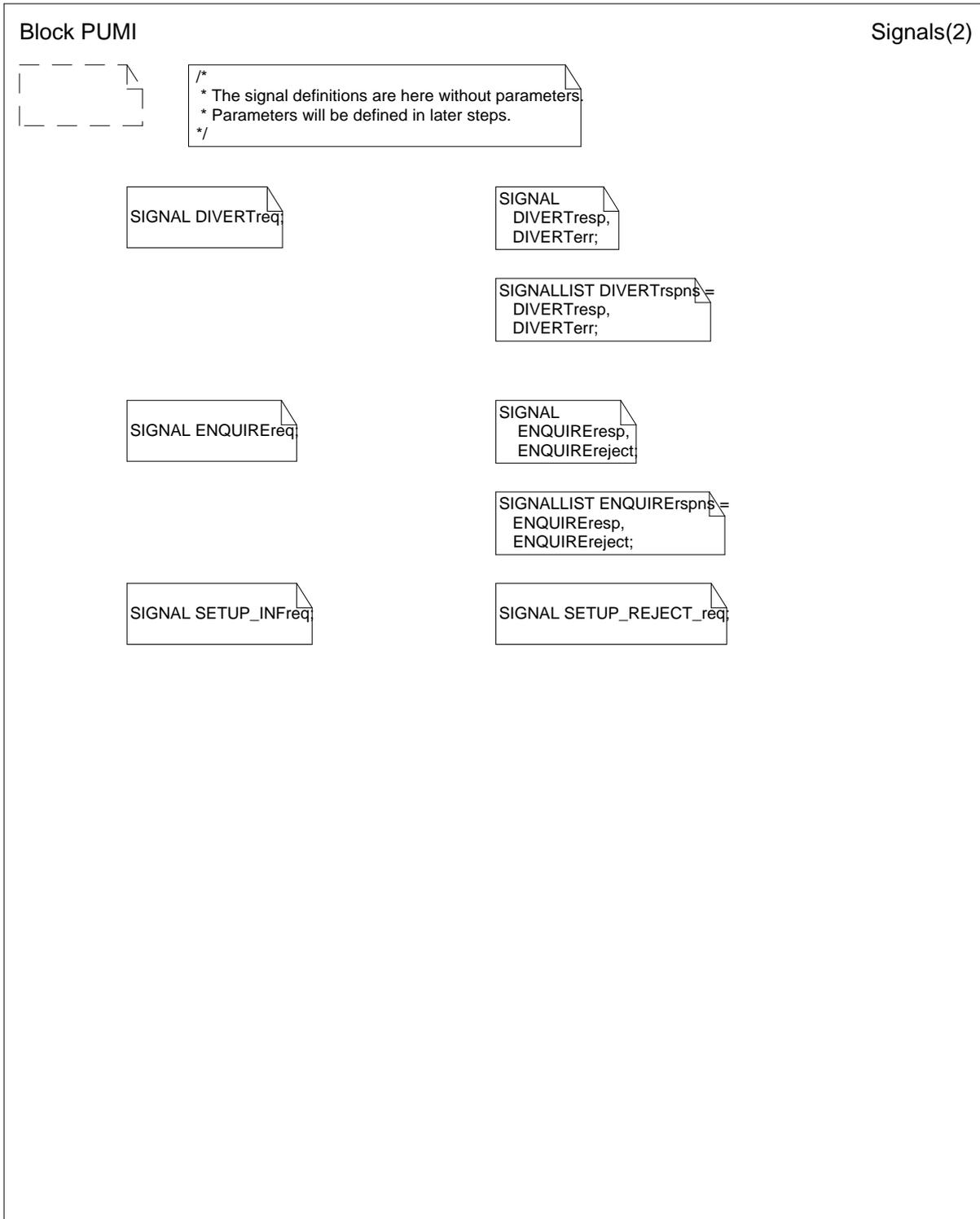


Figure 11: Step S:8 Local signals in a block

6.2.2 Behaviour-steps (B-steps)

6.2.2.1 Step B:1 Set of signals to a process

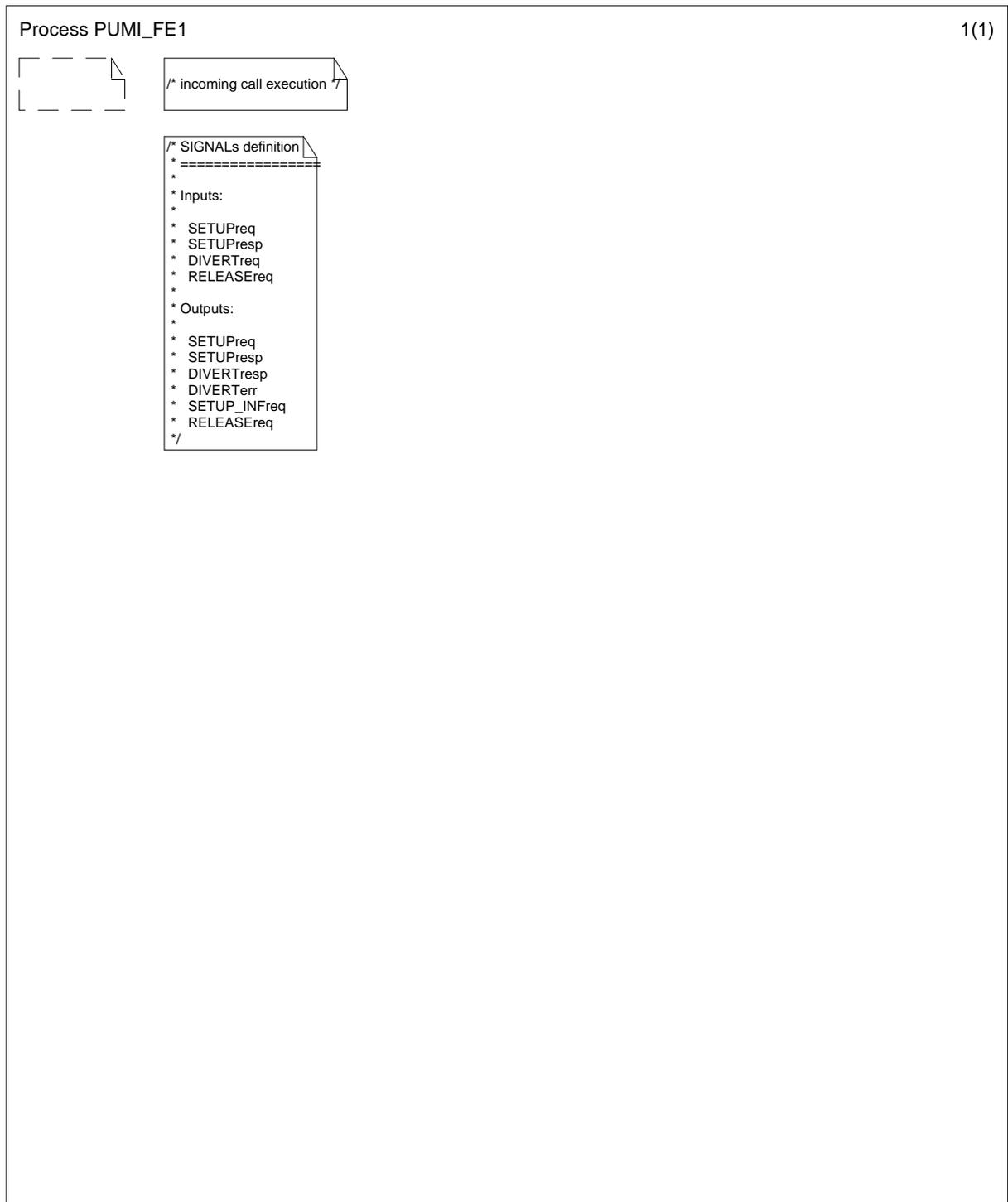


Figure 12: Step B:1 Set of signals to a process

6.2.2.2 Step B:2 Skeleton processes

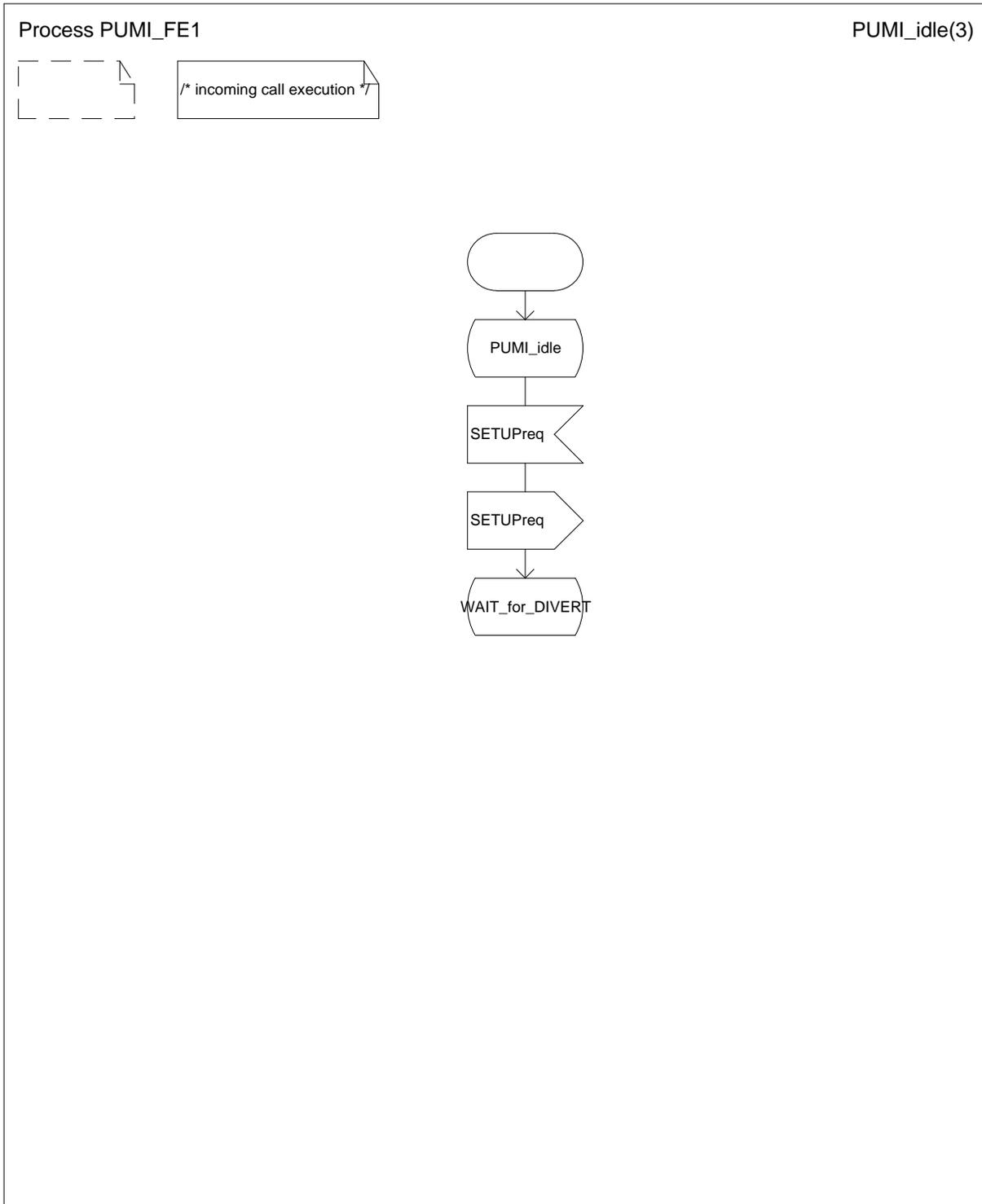


Figure 13: Step B:2 Process PUMI_FE1: PUMI_idle

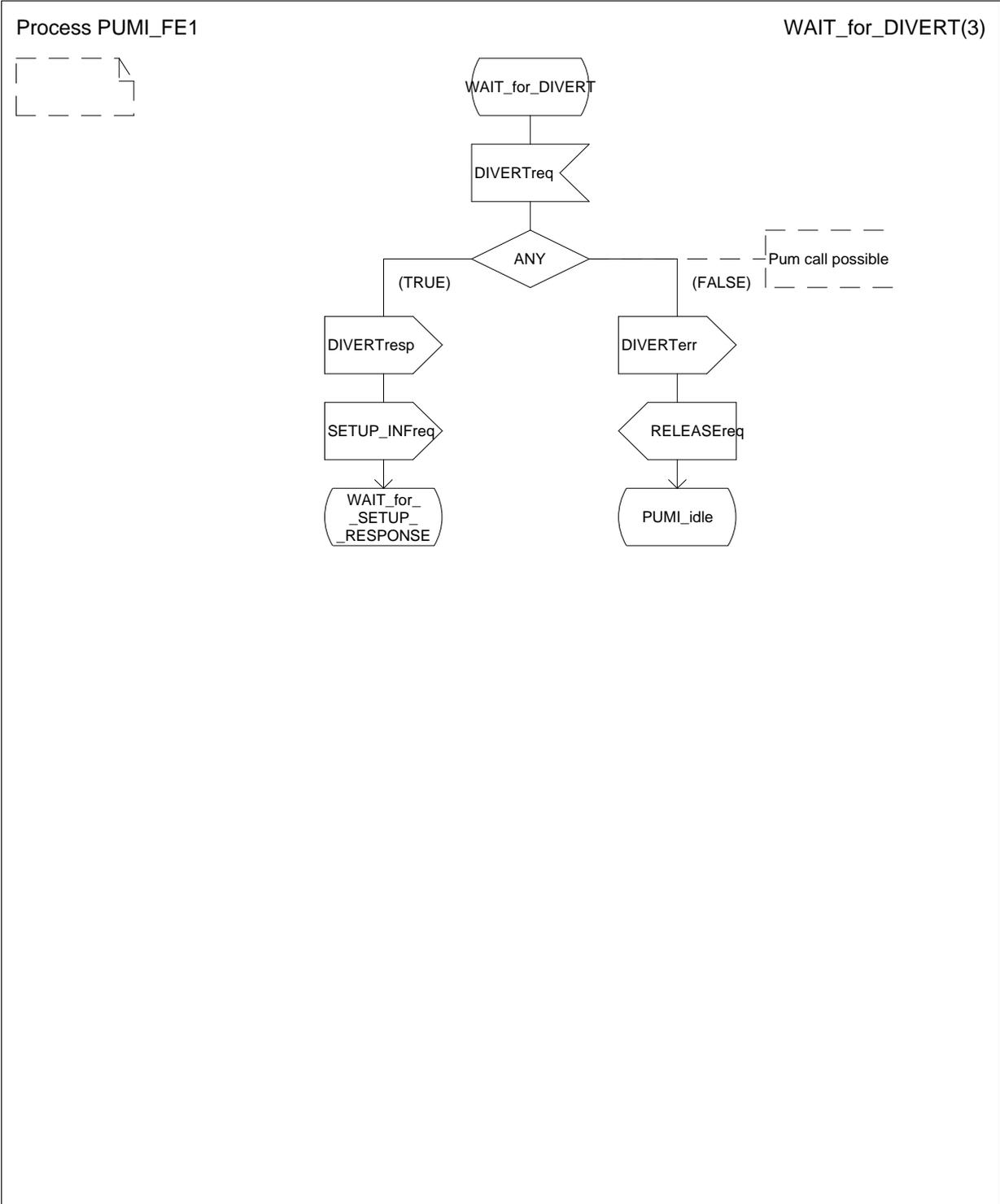


Figure 14: Step B:2 Process PUMI_FE1: WAIT_for_DIVERT

6.2.2.3 Step B:3 Informal processes

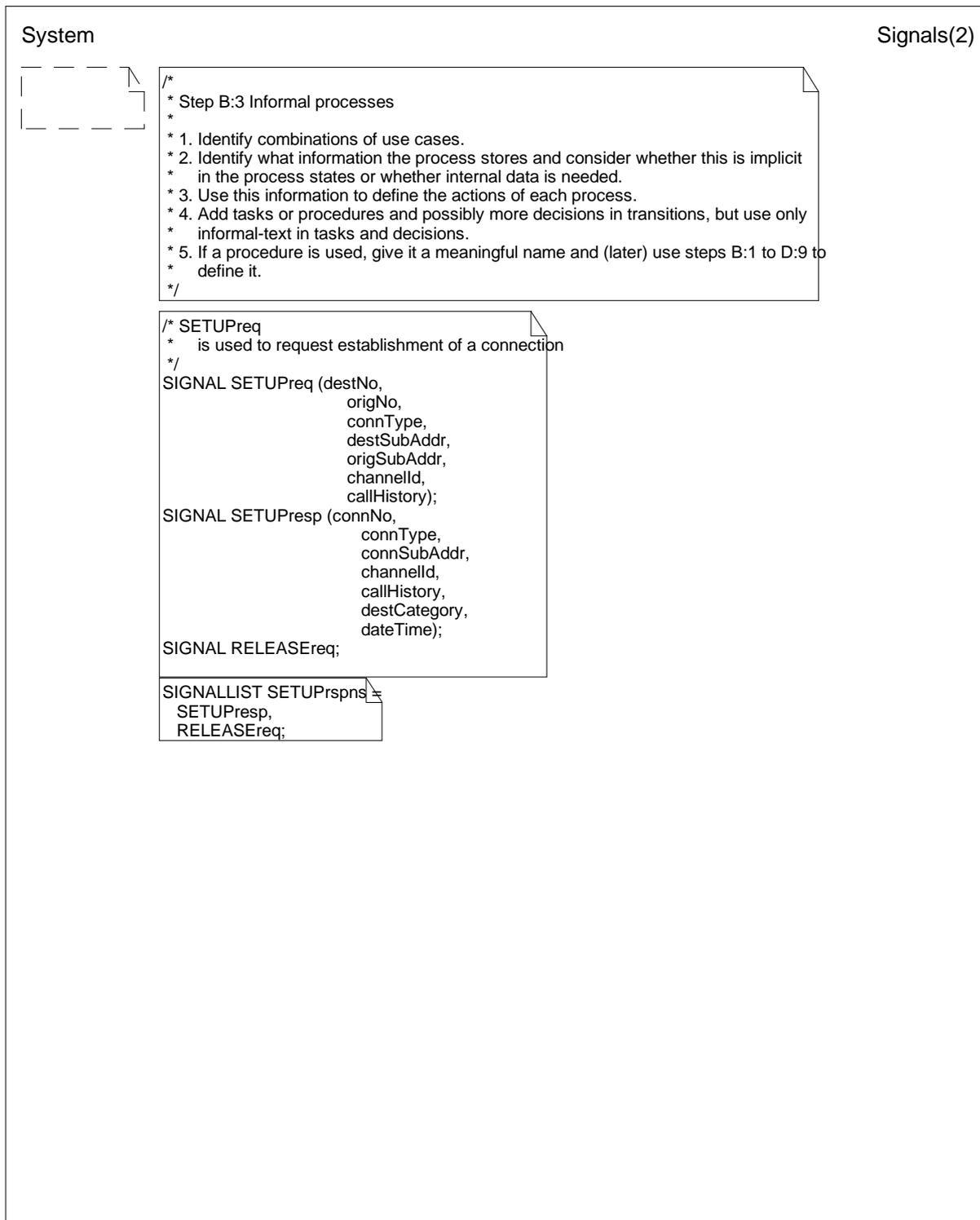


Figure 15: Step B:3 System signals

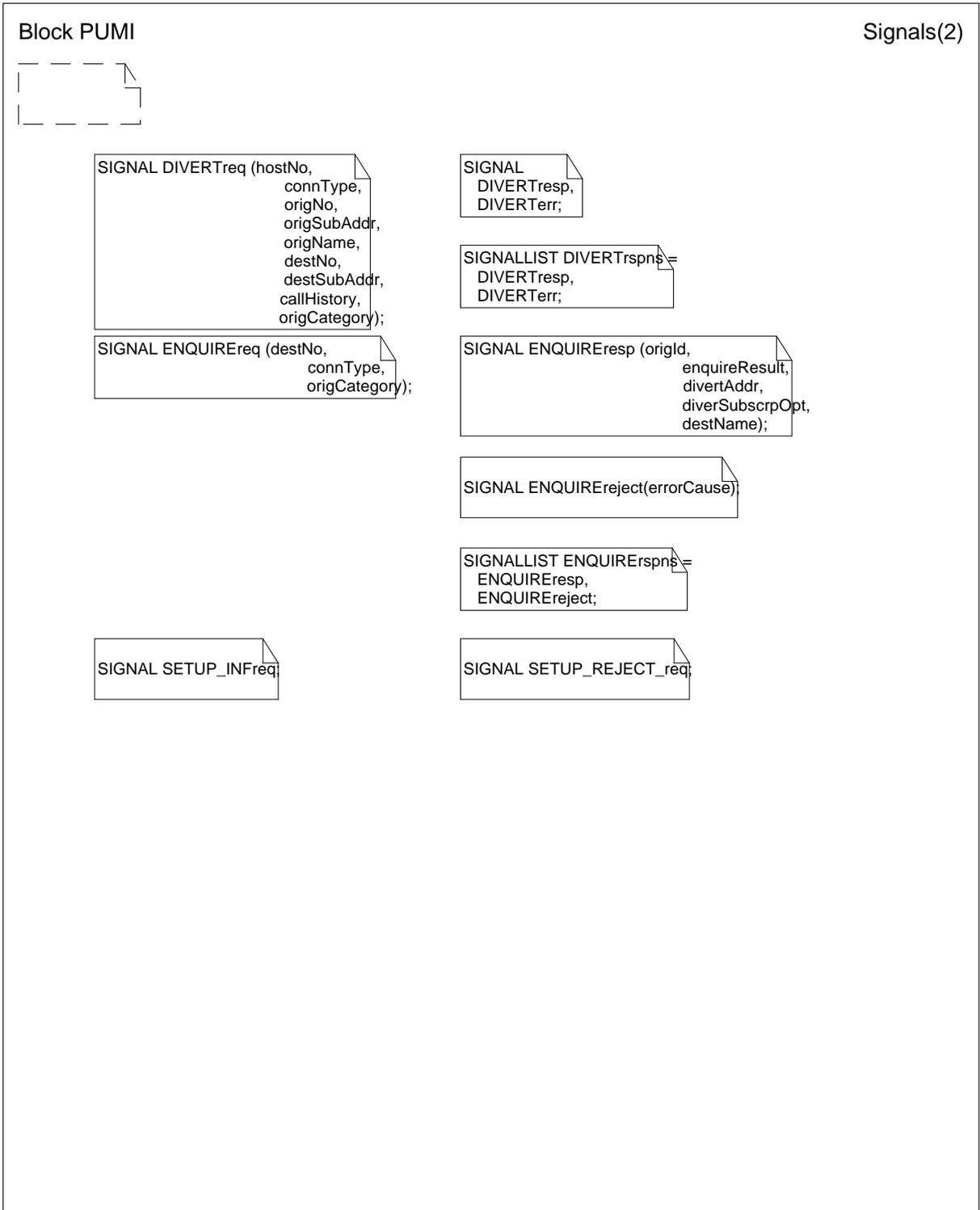


Figure 16: Step B:3 Block signals

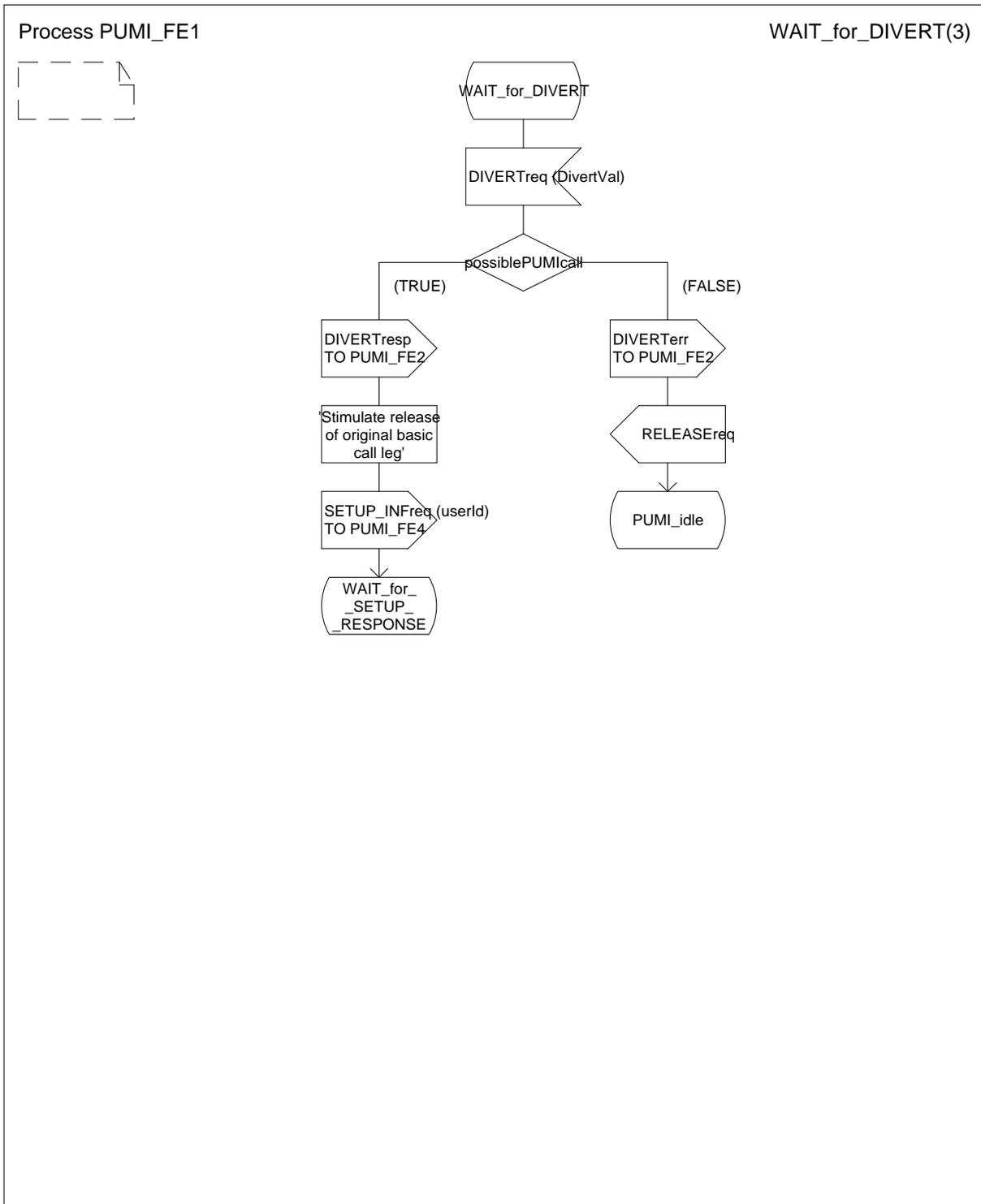


Figure 17: Step B:3 Process PUMI_FE1: WAIT_for_DIVERT

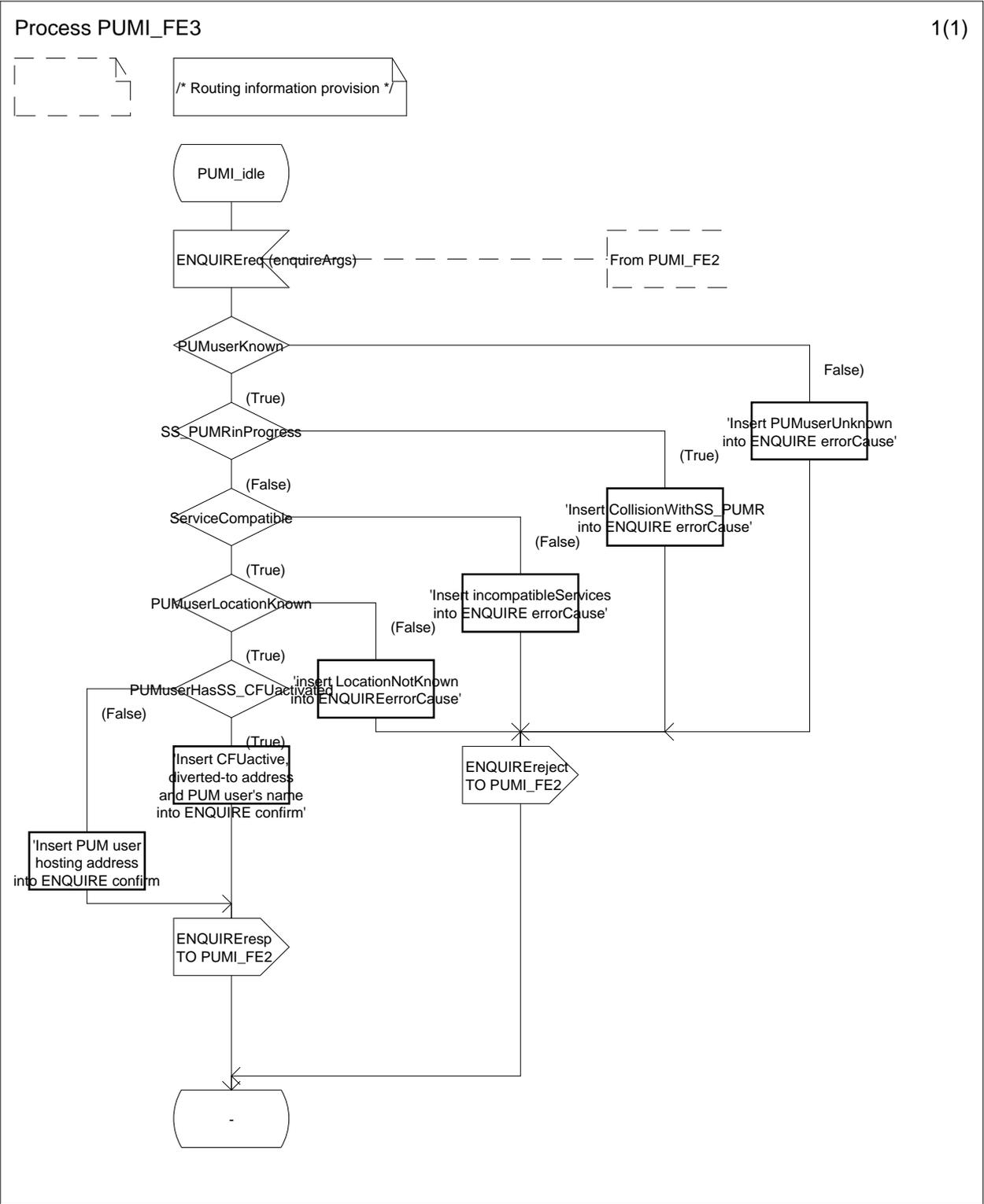


Figure 18: Step B:3 Process PUMI_FE3

6.2.2.4 Step B:4 Complete processes

Step B:4 is illustrated in subclause 6.2.2.3.

6.2.3 Data-steps (D-steps)

6.2.3.1 Step D:1 Signal parameters

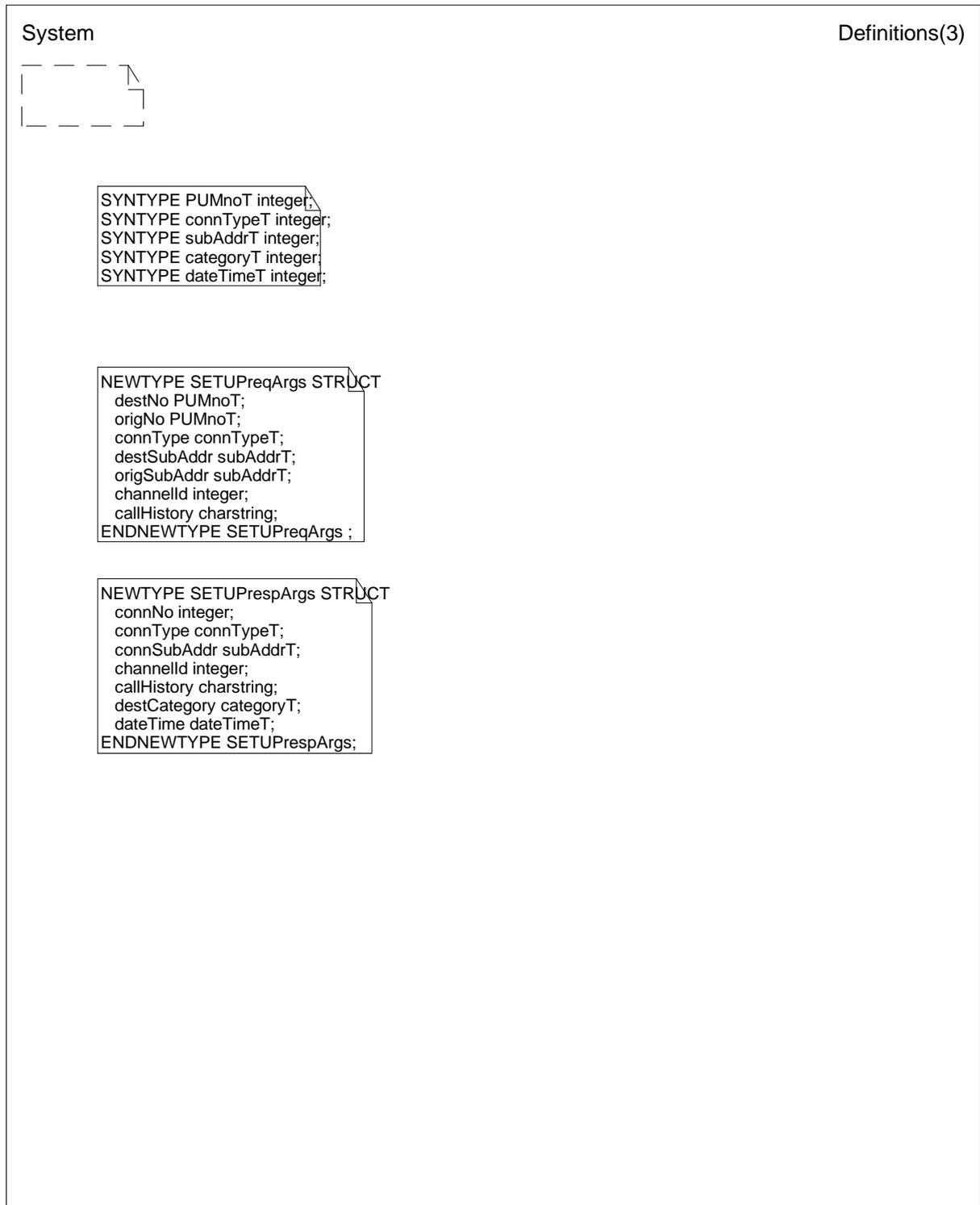


Figure 19: Step D:1 System definitions

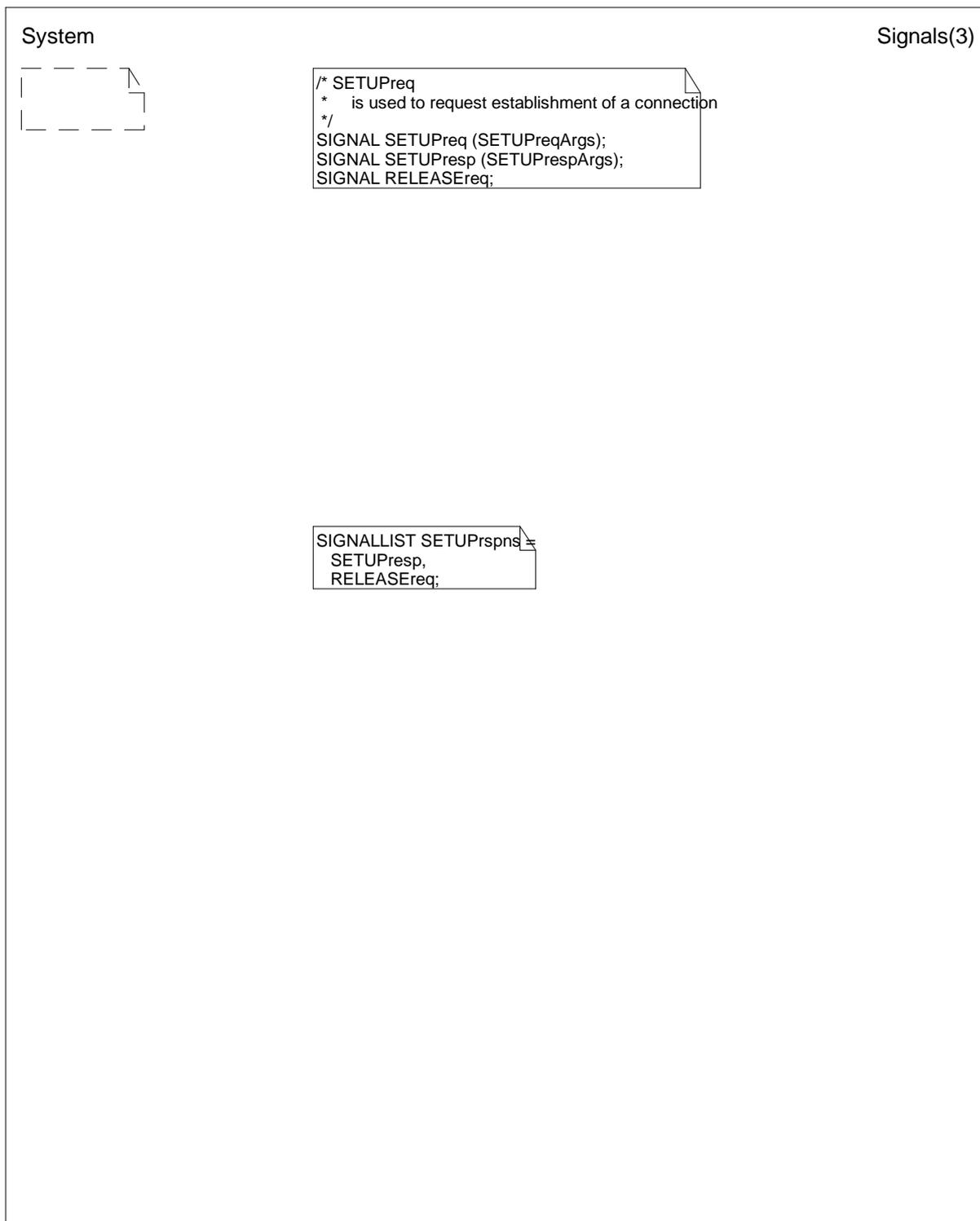


Figure 20: Step D:1 System signals

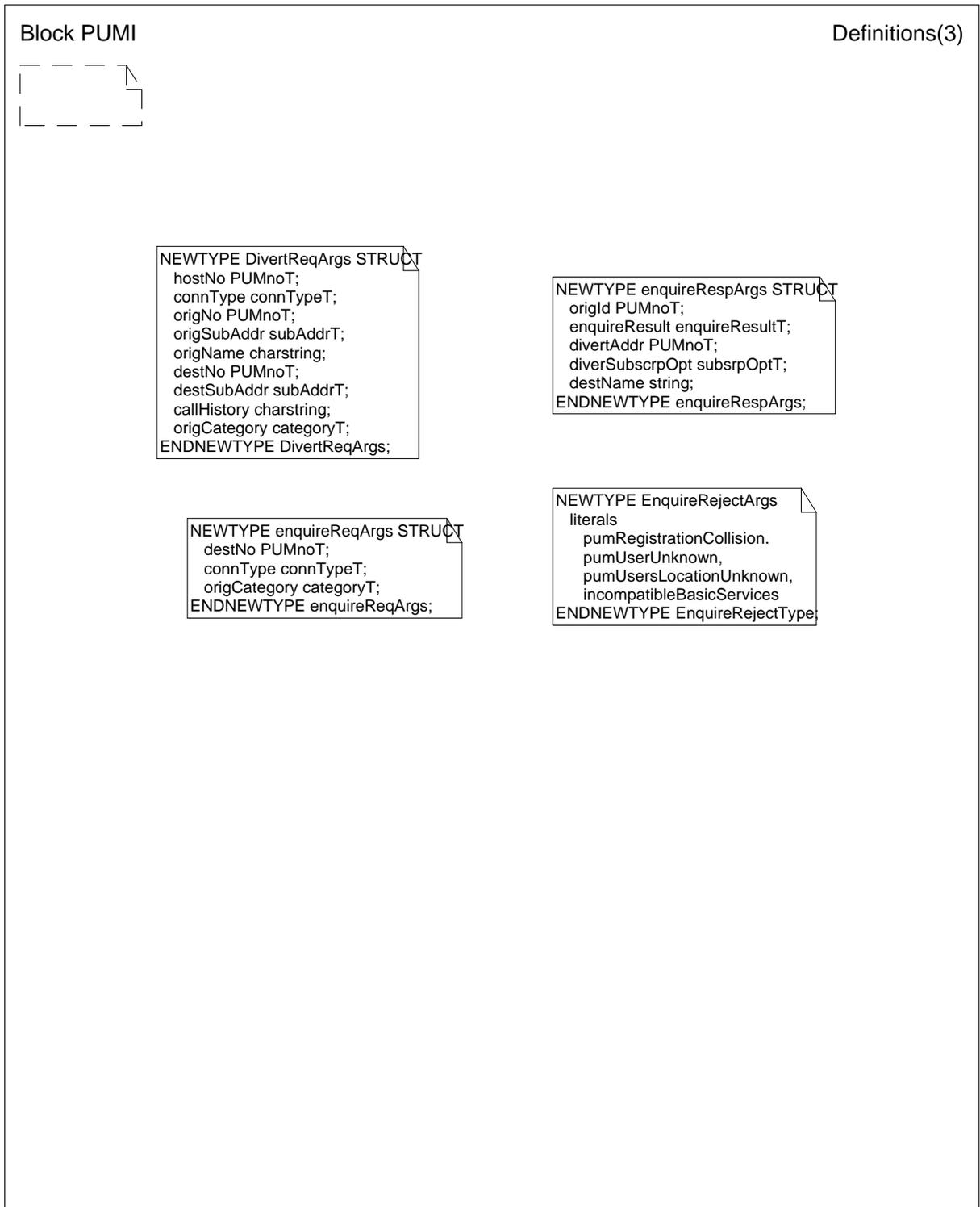


Figure 21: Step D:1 Block definitions

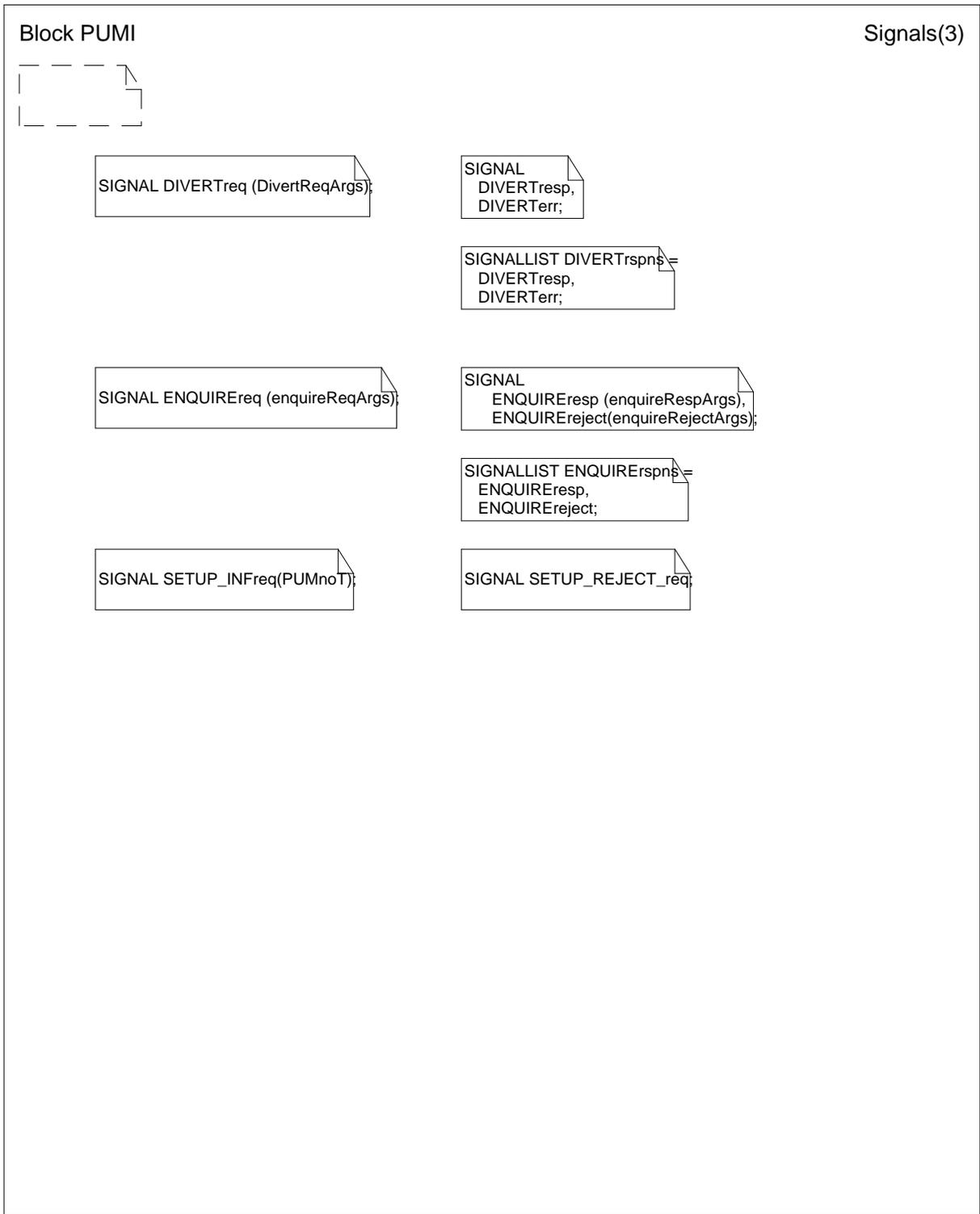


Figure 22: Step D:1 Block signals

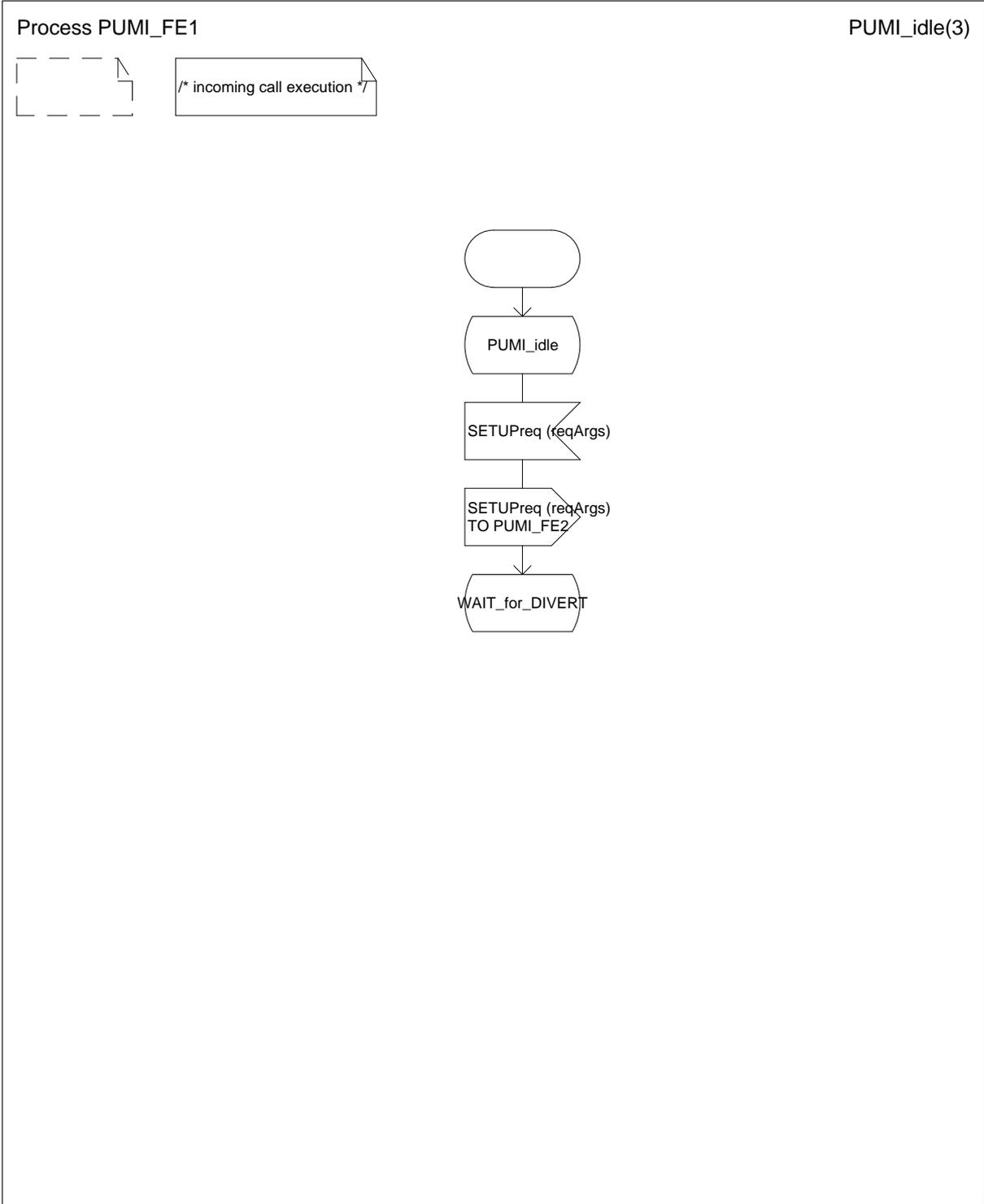


Figure 23: Step D:1 Process PUMI_FE1: PUMI_idle

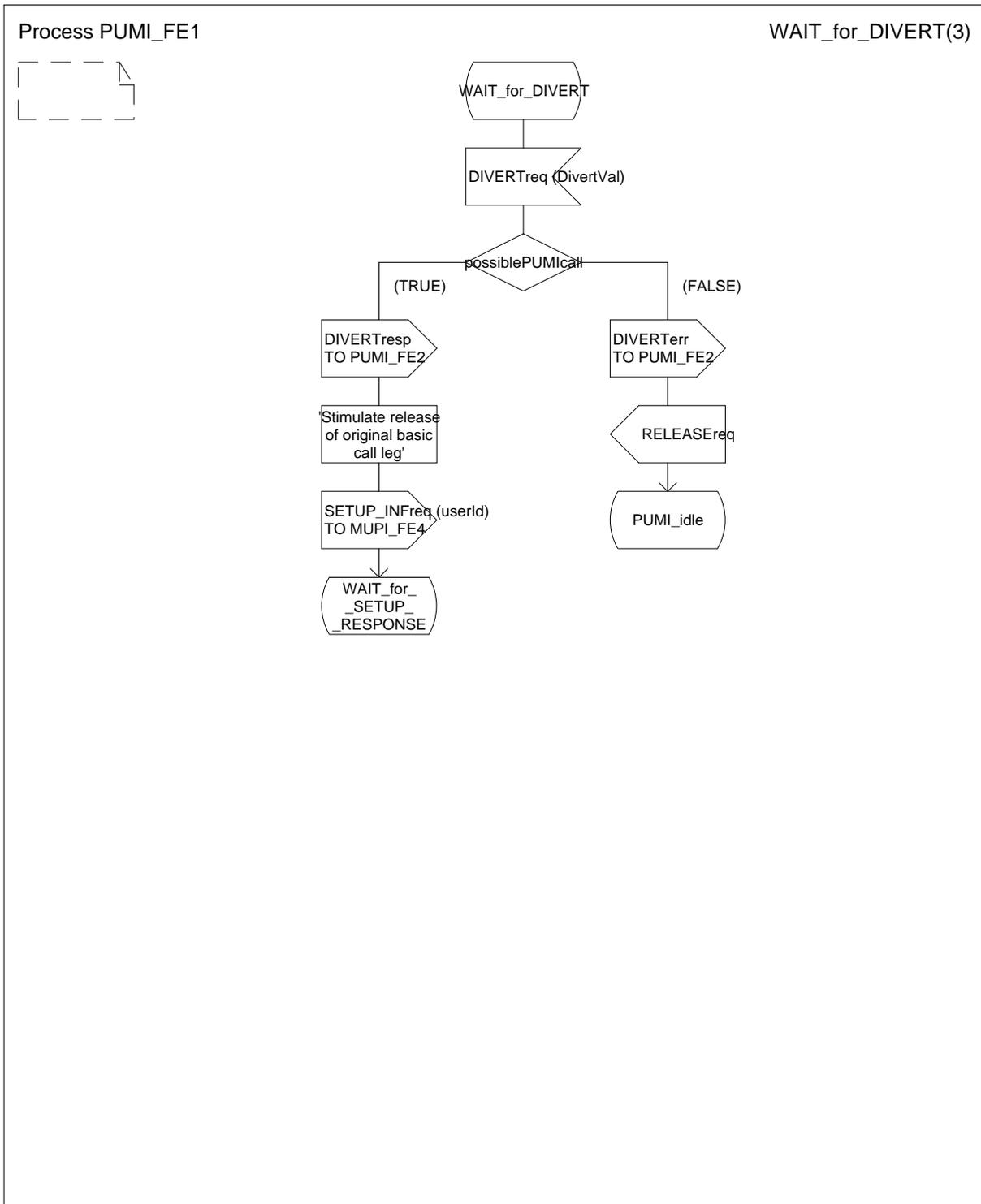


Figure 24: Step D:1 Process PUMI_FE1: WAIT_for_DIVERT

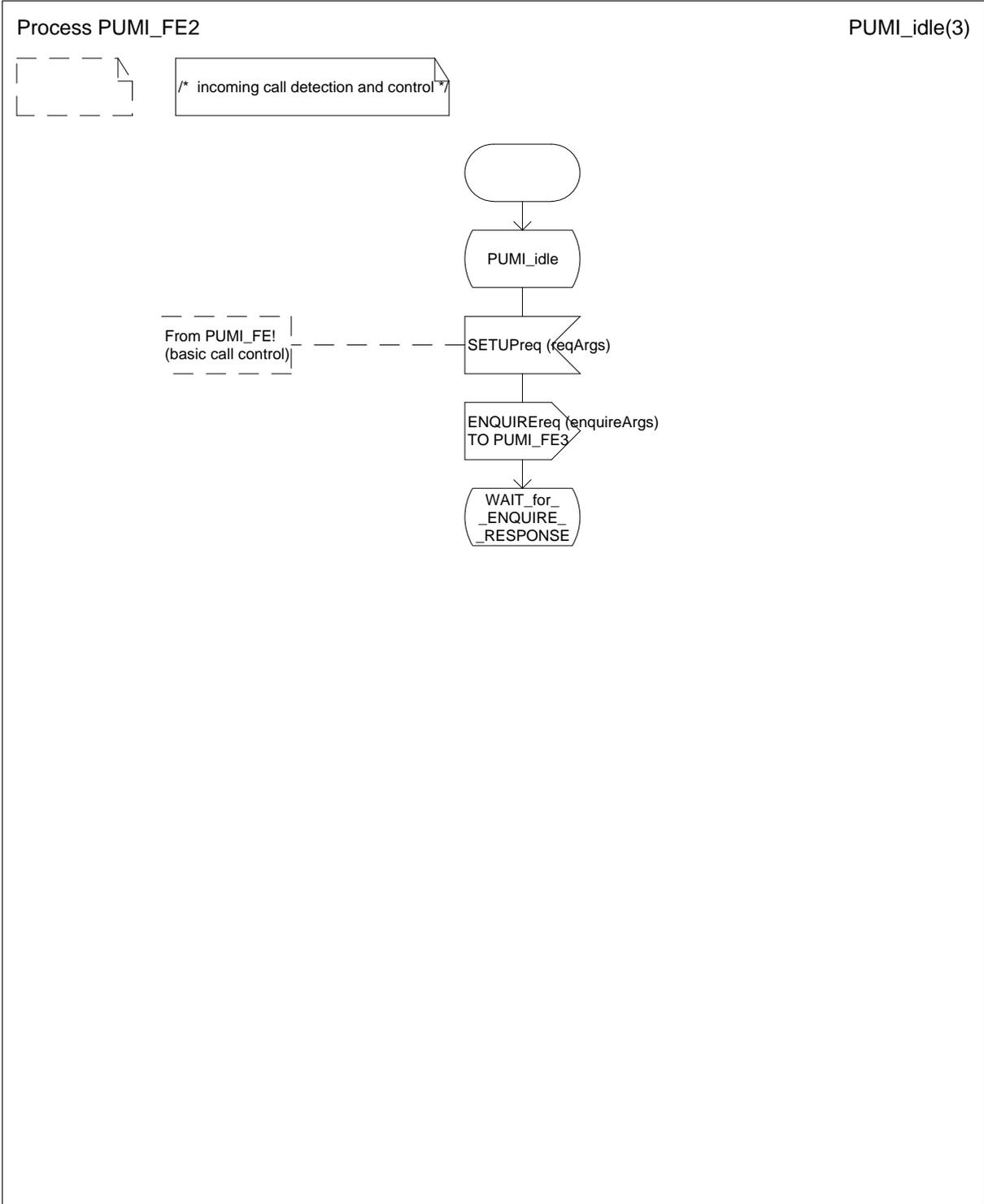


Figure 25: Step D:1 Process PUMI_FE2: PUMI_idle

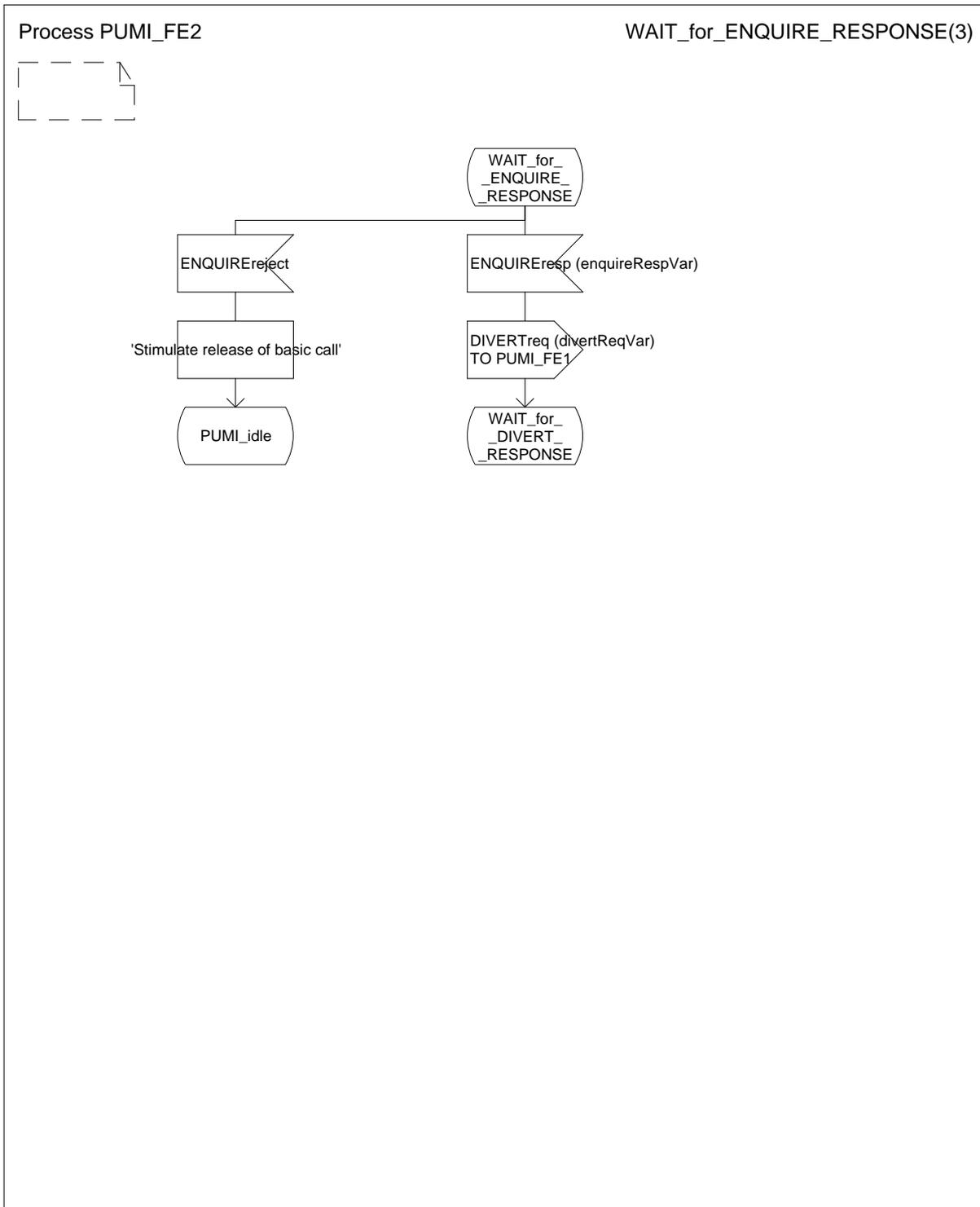


Figure 26: Step D:1 Process PUMI_FE2: WAIT_for_ENQUIRE_RESPONSE

6.2.3.2 Step D:2 Process and procedure parameters

Step D:2 is illustrated in subclause 6.2.3.1.

6.2.3.3 Step D:3 Signal variables

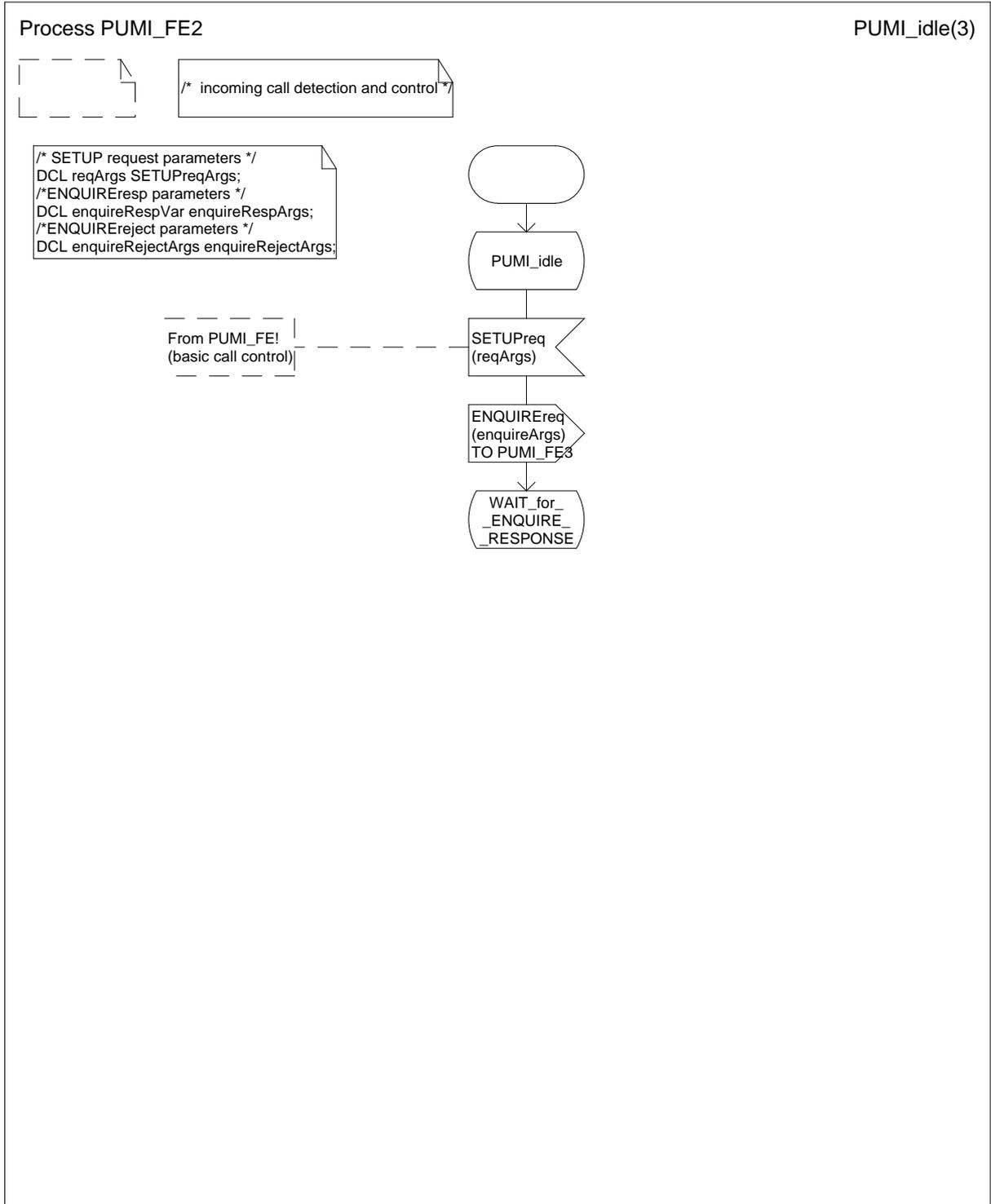


Figure 27: Step D3 Process PUMI_FE2: PUMI_idle

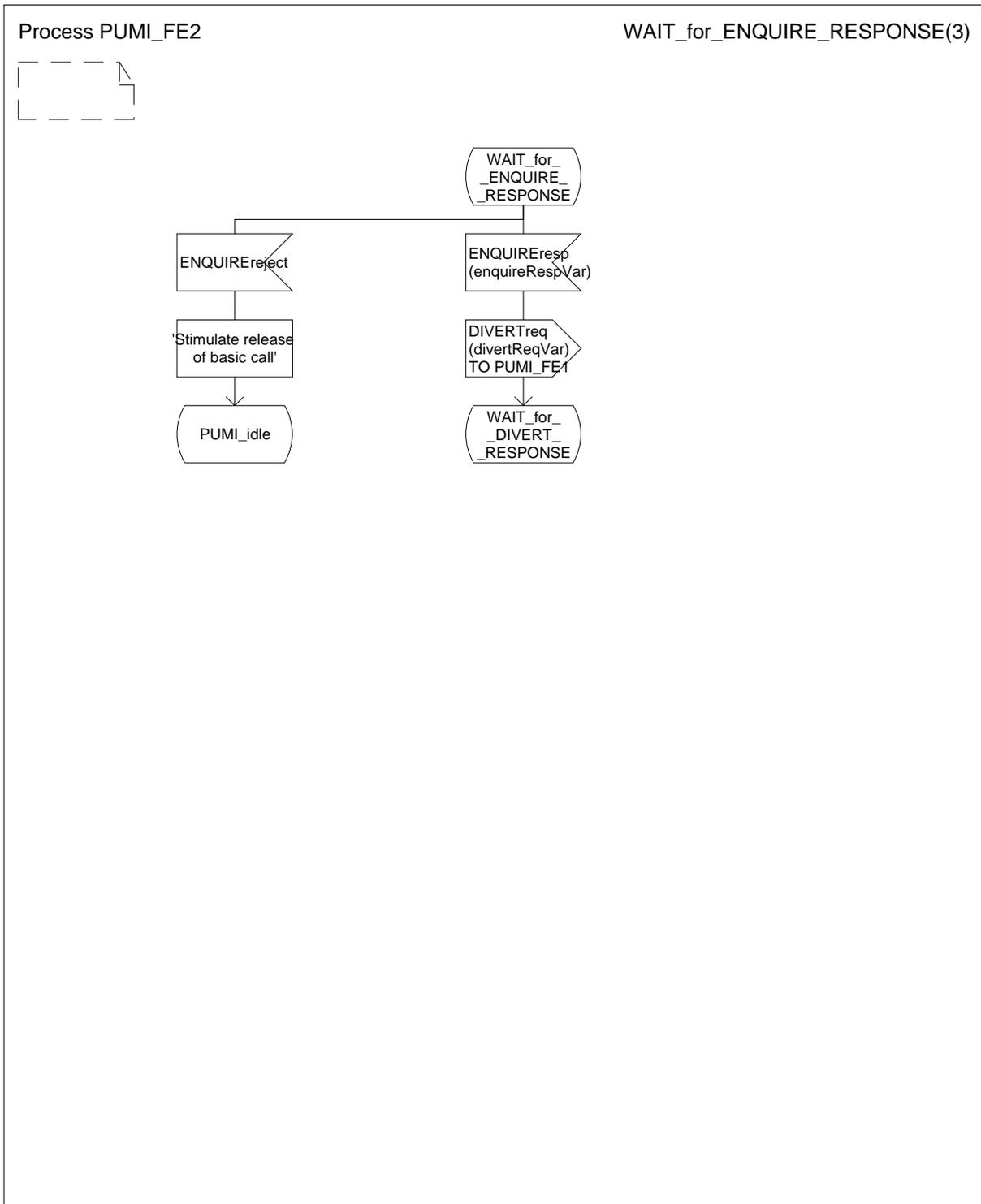


Figure 28: Step D3 Process PUMI_FE2: WAIT_for_ENQUIRE_RESPONSE

6.2.3.4 Step D:4 Formal transitions

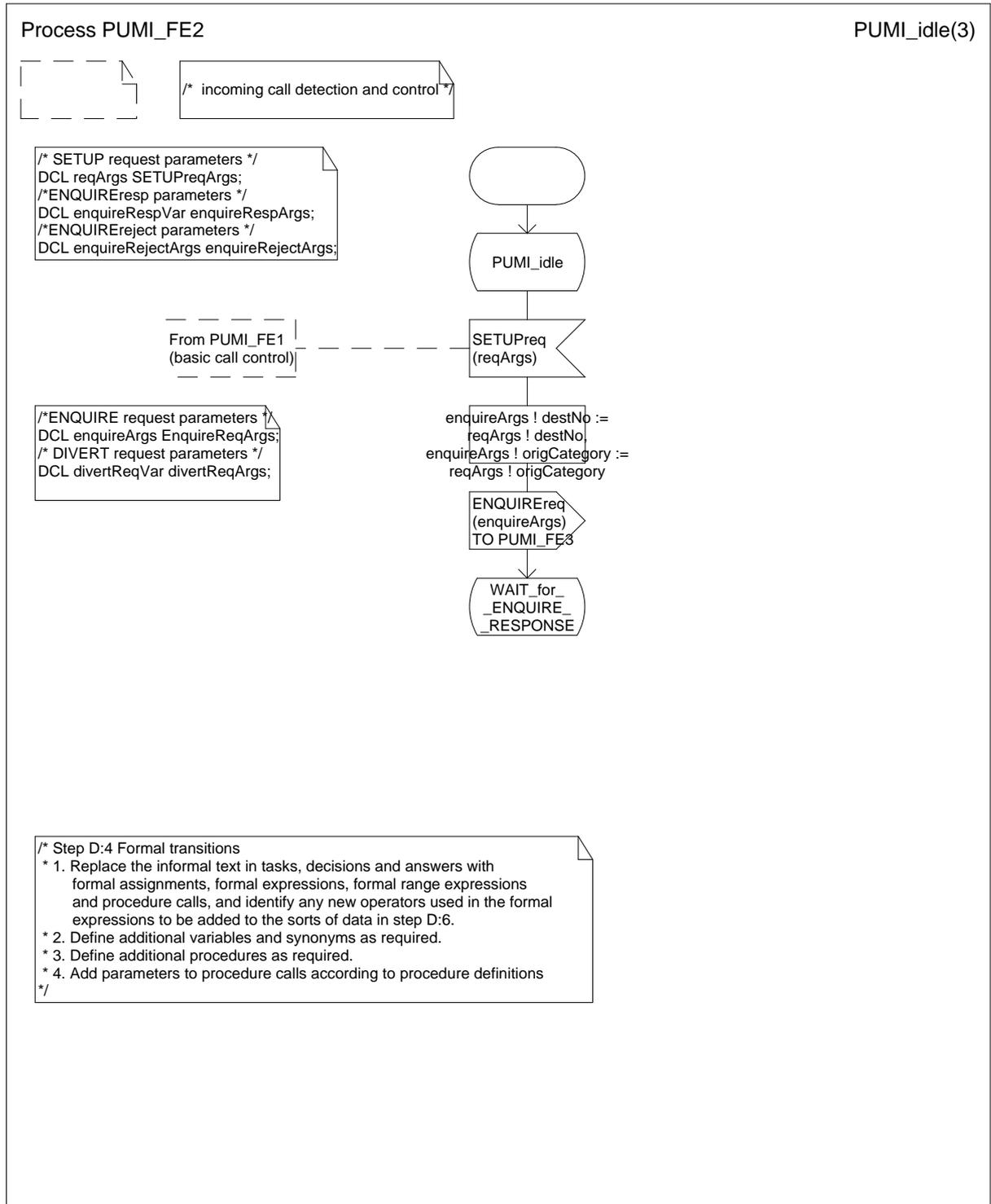


Figure 29: Step D4 Process PUMI_FE2: PUMI_idle

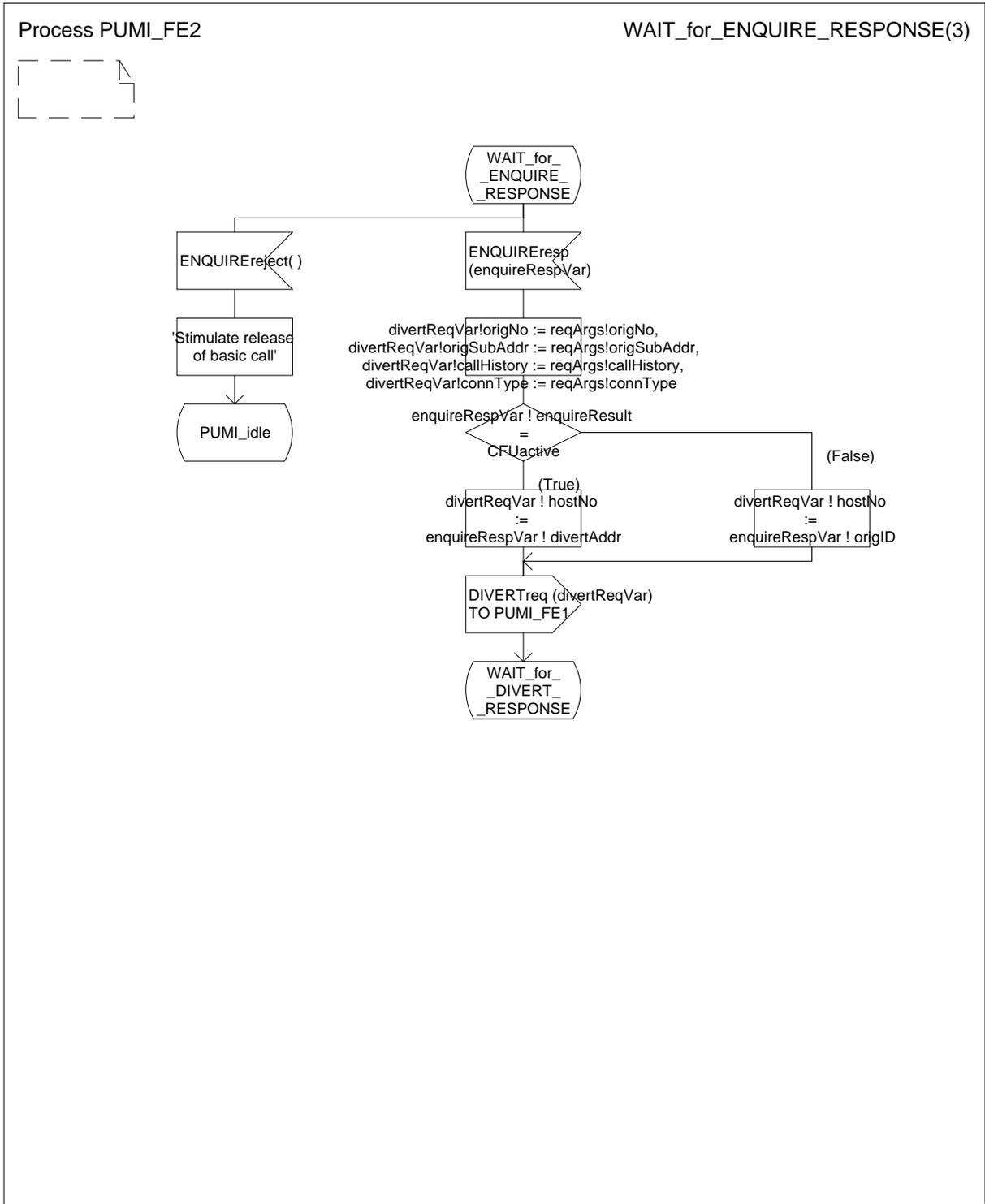


Figure 30: Step D:4 Process PUMI_FE2: WAIT_for_ENQUIRE_RESPONSE

6.2.3.5 Step D:5 Output and create parameters

Step D:5 is illustrated in subclause 6.2.3.4.

6.2.3.6 Step D:6 Data signatures

Step D:6 is illustrated in subclause 6.2.3.4.

6.2.3.7 Step D:7 Informal data description

Step D:7 is illustrated in subclause 6.2.3.4.

6.2.3.8 Step D:8 Formal data description

Step D:8 is illustrated in subclause 6.2.3.4.

6.2.3.9 Step D:9 Complete data formalization

Step D:9 is illustrated in subclause 6.2.3.4.

Annex A: Overview of ETS 300 414

A.1 Introduction

ETS 300 414 specifies rules for the use of the Specification and Description Language (SDL) in ITU-T Recommendation Z.100 [3] and Message Sequence Charts (MSC) in ITU-T Recommendation Z.120 [6] in ETSs. It is intended that SDL and MSC diagrams will be used in ETSs, in combination with text, informal figures and tables. SDL diagrams are to be used to formalize those parts of an ETS that need to be defined precisely and unambiguously.

Furthermore, it is recommended to use the Abstract Syntax Notation One (ASN.1) in ITU-T Recommendation X.208 [9] together with SDL. Therefore, this ETS also specifies rules for the use of ASN.1, when used in combination with SDL. The combined use of ASN.1 and SDL is the subject of ITU-T Recommendation Z.105 [4].

All rules are aimed at improving the possibilities to validate a standard in an early phase of development, and to improve the possibilities to test products that claim to implement a standard.

The document describes specification principles that enable validation and testing and applies these principles for defining rules. The complete list of rules can be found in annex C and some important elements and briefly described in the continuation of this clause.

consistency	An ETS should be internally consistent, i.e. there should be no contradictions between text paragraphs or between a diagram and the text or between different diagrams.
clarity	An ETS should clearly and unambiguously define requirements on the telecommunication product it specifies. The specification should be well-structured, in order to ease reviews by human experts.
correct use of formalisms	The diagrams presented in an ETS should be syntactically and semantically correct with respect to the formal language definitions. The use of concepts that are not supported by tools should be avoided. This principle is essential to enable the use of computer-based tools for consistency checking, validation and test derivation.
avoid state space explosion	An important class of formal validation methods is based on state space exploration. By specification of infinite numbers of process instances, or infinite data types, the state space of the system explodes. This prohibits the use of validation tools that are currently available.
avoid implicit non-determinism	If non-determinism is not explicitly specified, it is not easy to discover and may easily be overlooked by the designer of tests. Furthermore, it is difficult to determine whether the non-determinism is intentional or is a mistake in the standard.
indicate implementation options	A standard should clearly describe the implementation options. These options will be reflected in the ICS, that a manufacturer provides to indicate which options are supported in a product. This is important for selection of relevant tests for a product.
indicate normative parts of standard	A standard should clearly indicate which parts are normative. This determines which conformance requirements are implied by the standard.
one level of abstraction	A standard should not specify a system on multiple levels of abstraction. This is essential in order avoid misinterpretation about which conformance requirements are implied by the standard.

A.2 Normative interfaces

A normative interface in a telecommunication product is defined as the physical or software interface of a product on which requirements are imposed by a telecommunication standard. A telecommunication standard, or set of related standards that together define a product type, should clearly indicate which interfaces in the product are considered to be normative.

Normative interfaces should be marked in an SDL diagram with a comment "normative" attached to the channels that model the interfaces. Figure 31 shows an example.

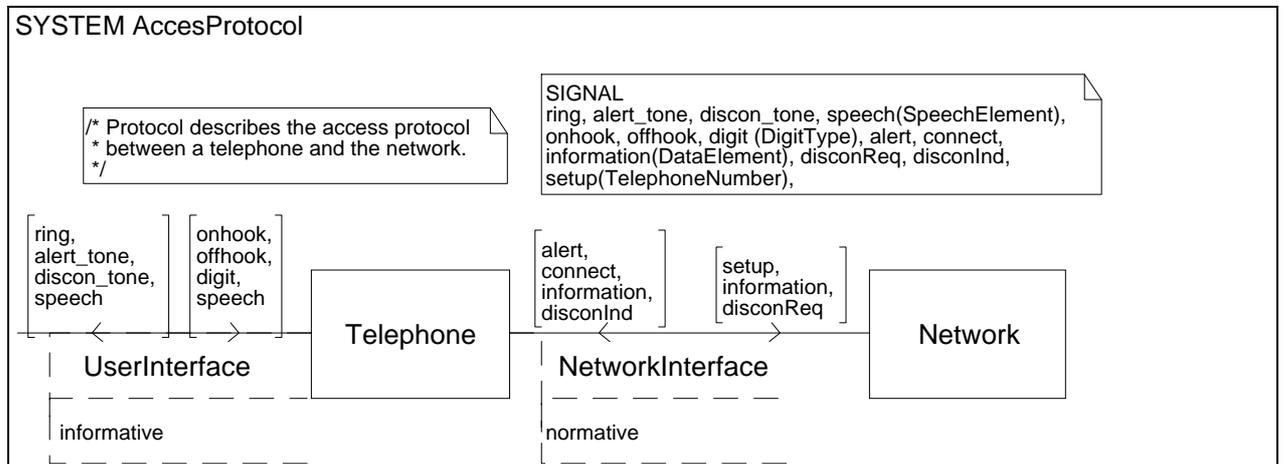


Figure 31: A system diagram indicating the normative interfaces of a system

A.3 Selection of SDL concepts

On the basis of principles listed above, the set of SDL concepts as defined in ITU-T Recommendation Z.100 [3] has been partitioned as shown in table 1. The motivations for such decision can be found in ETS 300 414 [1]. It should be noted that object-oriented features of SDL are not dealt with in ETS 300 414 [1].

Table 1: Selection of SDL concepts

	Unrestricted use allowed	Restricted use allowed	Not allowed
System or block diagrams	block, channel, comment, package, process, process create line, select, signal list, signal route, text, text extension	block substructure, data type definition, macro call, signal declaration	channel partitioning, signal refinement
Process or Procedure diagram	comment, input, join, label, optional transition, priority input, process creation, process start, process stop, procedure call, procedure return, procedure reference, procedure start, save, state, synonym , text, text extension, variable,	continuous signal , decision, macro call, output, task, timer	enabling condition, import and export, internal input and output, service, view and reveal
Data type diagram	predefined data	abstract data, ASN.1 type definition	name class literal

Table 1 gives an overview of the selection of SDL concepts. It should be noted that the object oriented extensions in SDL 1992 are not considered. This is because very little experience in using them exists:

- the concepts in the column "Unrestricted use allowed" are the concepts that can be used without any restrictions, i.e. they are not considered harmful for testability or validation, or they were considered indispensable by a panel of experienced SDL users;
- the concepts in the column "Restricted use allowed" are concepts that, when used in a specific way, influences testability or validation. For each such concept a rule is given restricting the allowed use;
- the concepts in the column "Not allowed" are concepts that are harmful for testability or validation, or were considered superfluous.

A.4 Selection of Message Sequence Chart concepts

ITU-T Recommendation Z.120 [6] is used to describe sequences of events that can be performed by the standardized system. MSCs are useful to give an overview or used to guide the selection of test purposes.

MSCs are closely related to ITU-T Recommendation Z.100 [3]: SDL diagrams give the complete behaviour of the standardized system, while MSCs give typical cases. The sequence of events in an MSC should be a part of the behaviour that is defined in the SDL diagrams. Consistency between SDL and MSCs can be checked automatically with tools.

Table 2 gives an overview of the MSC concepts that are allowed in combination with SDL.

Table 2: Overview of selection of MSC constructs for use in combination with SDL

unrestricted use allowed	restricted use allowed	not allowed
action	process creation	co-region
comment	instance	sub MSC
condition	message	
process stop	timer	

Figure 32 gives an example of an MSC of the multi-party supplementary service in GSM. It is the system functions. MSCs contribute to the testability of a standard, because it may be shown that the visitor's location register is consulted before a service is provided, and that a service request is rejected if the subscriber is not authorized for use of the service.

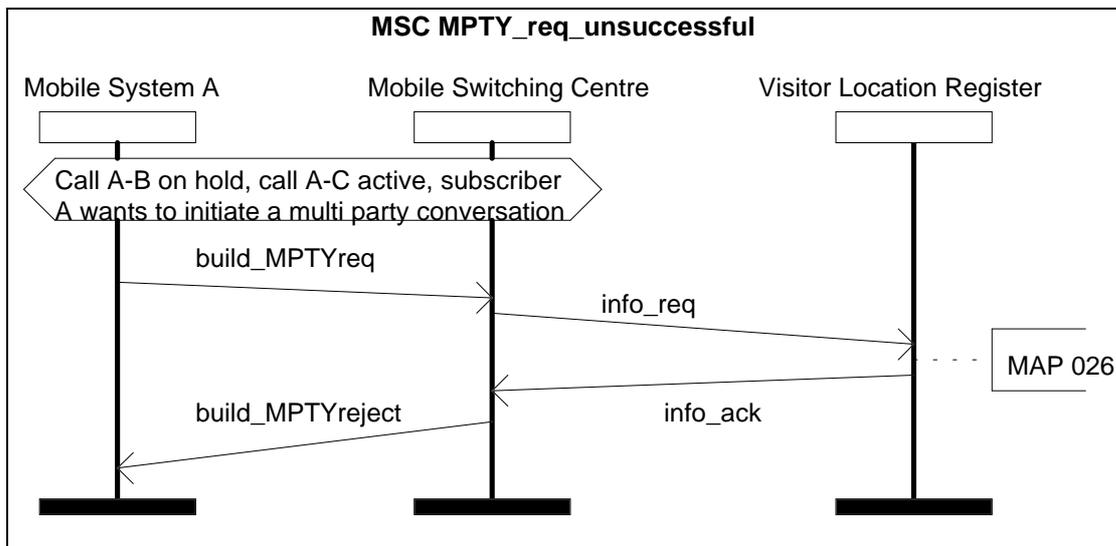


Figure 32: Example of MSC showing an unsuccessful request for a GSM multi party call

A.5 Selection of ASN.1 concepts

ITU-T Recommendation X.208 [9] is widely used to define structure messages that are exchanged by telecommunication systems. While it is also possible to use SDL data types for this purpose, practice shows that ASN.1 is preferred. In fact, many ETSs do already use ASN.1 in combination with SDL. Instead of forbidding what is done in practice, this ETS regulates such combined use of SDL and ASN.1. The approach taken in ETS 300 414 [1] largely contributed to acceptance of the ITU-T Recommendation Z.105 [4].

Table 3 gives an overview of ASN.1 concepts that can be used without restrictions in combination with SDL, concepts that can be used if the rules that restrict their use are met, and concepts that should not be used at all.

Table 3: Overview of selection of ASN.1 concepts to be used in combination with SDL

unrestricted use allowed	restricted use allowed	not allowed
ASN.1 module definition	ASN.1 identifier names	SET
IMPORTS, EXPORTS	SEQUENCE	ASN.1 macro (for example
NULL	SEQUENCE OF	the operation macro)
BOOLEAN	SET OF	ASN.1 comment
INTEGER	ASN.1 tags	value notation of
REAL	MIN, MAX	ASN.1 ANY type
ENUMERATED	PLUS-INFINITY	
OBJECT IDENTIFIER	MINUS-INFINITY	
BIT STRING, OCTET STRING	ANY	
The different character strings		
CHOICE		
default and optional component		
subtyping		

Annex B: Overview of methodology

B.1 The methodology activities

Figure 33 shows the methodology¹⁾ activities framework.

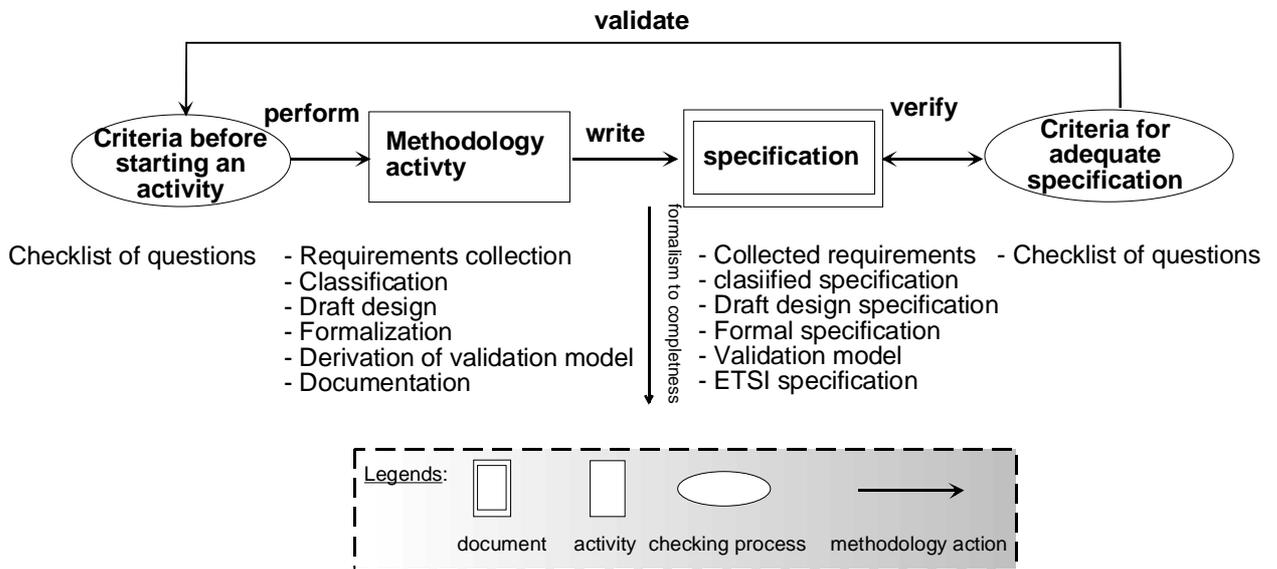


Figure 33: The methodology activities

As shown in figure 33, the result of performing an activity (requirements collection, classification, draft design, formalization, derivation of a validation model, or documentation) is a specification (collected requirements, classified specification, draft design specification, formal specification, validation model, ETS), consisting of a set of descriptions containing the knowledge acquired during the specification production.

Before and after each activity the methodology focuses on the definition of general criteria enabling the standard developer to follow closely the good performance of all the methodology processes. The criteria are present assisting the standard developer to determine (make a judgement) whether it is more appropriate to perform a specific activity or to skip it, helping to decide which activity is the appropriate entry point in the methodology. There are two types of criteria:

- criteria that define what information is needed in the input specification before an activity can be started;
- criteria that allow the detection and the handling of possible inconsistency and completeness in the analysis of the output specification after the performing of an activity.

If the judgement is in favour, the standard developer can proceed directly to the next activity. If the judgement is against, he has to come back to the activity which provided the output specification.

For example, at a certain stage of his project the standard developer may take the advantage of a specification already described in ITU-T Recommendation Z.100 [3]. In this situation, he has to determine if this specification:

- contains as much formalized information as possible to be unambiguous, complete, understandable, verifiable, well-organized, executable, etc.,
- still fulfils the project requirements,
- etc.

¹⁾ This annex is an overview of work published in ETSI/MTS(94) 010.

The six activities of the methodology are:

- **requirements collection:**
 to verify that all the requirements collected respect quality attributes. This is to check that these requirements are reasonably described;
- **classification:**
 to get a first understanding, structuring the informal specification, and to re-express it in terms of concepts of the application domain;
- **draft design:**
 to increase such understanding; the standard developer can analyse different perspectives of the system, using one or more models that describe the system partially;
- **formalization:**
 once he has obtained a thorough understanding of the problem, the standard developer actually writes down the formal specification using the formal technique of ITU-T Recommendation Z.100 [3];
- **derivation of the validation model:**
 once the specification has been formalized, the standard developer provides a detailed and an executable (by the support of tools) version of the specification;
- **documentation:**
 from a formalized specification, the standard developer formats this according to standard rules used at ETSI (in CEN/CENELEC Internal Regulations [2]).

B.2 The requirement collection activity

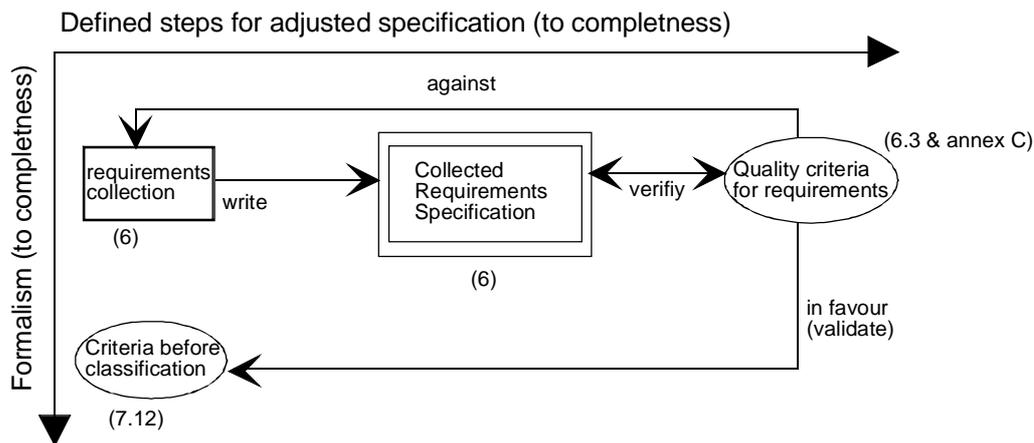


Figure 34: The requirements collection activity

The requirements collection activity determines whether relevant collected requirements (all the externally observable behaviours and characteristics expected of a protocol or a service) satisfy quality criteria. If not, the methodology considers that all the available and the relevant requirements have not been gathered. On the contrary, the standard developer may start the next activity (classification).

The requirements collection activity, as shown in figure 34, explores thoroughly the quality attributes in a collected requirements specification.

Any quality attribute can be achieved, but this may be at the expense of other attributes. On any one given requirement the standard developer needs to agree as to which quality attributes are most important and strive for those.

For example, if the standard developer removes all ambiguity, so much formality would be added to the collected requirements specification that they would no longer be understandable by a non expert. If the standard developer removes all redundancy, the collected requirements specification becomes difficult to read. If completeness is taken to extremes, conciseness will suffer.

B.3 The classification activity

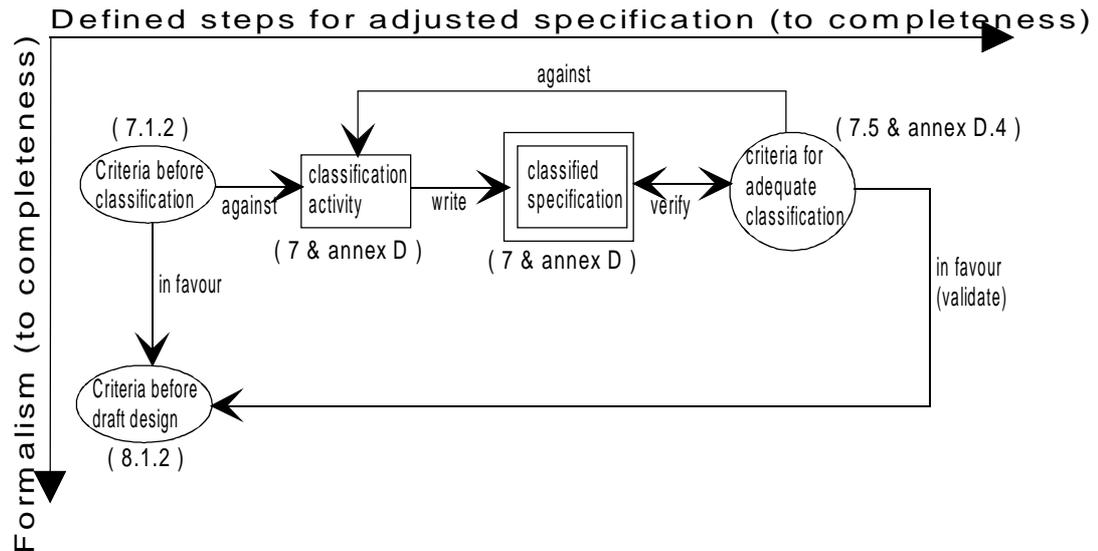


Figure 35: The classification activity

The classification activity, as shown in figure 35, is the stepwise process of inspecting, identifying, defining, analysing and organising a set of application domain information elements, from an input specification (collected requirements, and/or other input documents), that fulfil the previously defined collected requirements, and then re-expressing them into a well-structured (classified) specification.

For example, in the data communication application domain, there may be defined application domain elements like "transport protocol data unit", "transport service primitive", "transport protocol entity", etc. These information elements are related to the collected requirements.

Any method may be used by the standard developer to classify the input specification. The methodology offered by PT60 suggests that the classification process may be based on an object-oriented approach.

A classified specification is considered as adequate (complete, consistent, sufficiently structured, with a well-defined terminology) for the other next activities to be completed, when it complies with a checklist of criteria. In this case the classified specification can be regarded as a structured and referenced common information base for all standard developers.

During the classification process questions are raised and decisions are made. These are related to the contents of and the interpretation of the input specification.

B.4 The draft design activity

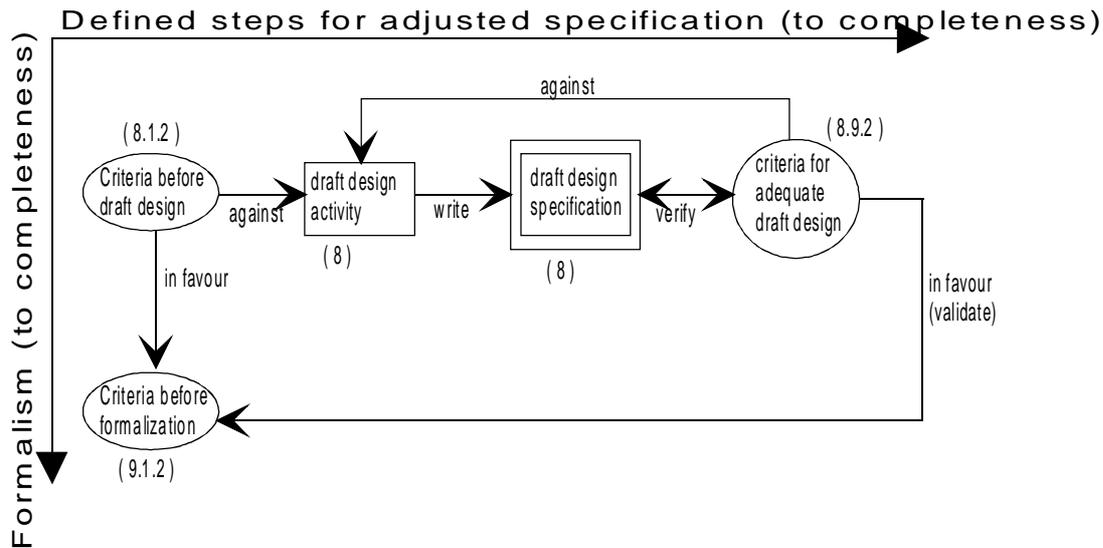


Figure 36: The draft design activity

The draft design process, as shown in figure 36, analyses aspects of the specification under development using various methods, called draft design methods. This analysis consists of building different views of (parts of) the classified specification in order to get a better understanding, to derive properties and/or to detect any ambiguities, incompleteness or contradictions. A produced specification expressed and structured according to the notation of a draft design method is called a draft design specification. The draft design specification describe the system partially using models.

To build each view, the standard developer uses a draft design method which is suitable to explore that particular perspective. For example, Message Sequence Charts are a quite adequate method to describe the dynamic aspects of an interface between some parts of a system under development. It should be noted that the set of views may overlap.

The set of views are recorded and linked with the corresponding parts of the classified specification.

During the draft design process questions and decisions are also made. These are recorded and linked with the relevant parts of the specification.

The methodology includes guidelines to choose the best set of draft design methods to support various analysis and to prepare for building the formal specification.

Facilities to define the several views, to link them with the corresponding parts of the classified specification are used throughout the draft design process.

The methodology includes the following draft design methods:

- entity Relationship Diagram (ERD): modelling types of entities and their relevant attributes, as well as types of relationships among instances of given entity types;
- abstract Syntax Notation One (ASN.1): standard notation formalising the structure and format of data in terms of elementary or predefined ones (ITU-T Recommendation X.208 [9]);
- message Sequence Chart (MSC): specific cases (usually typical, with many exceptions uncovered), of temporal and dynamic message interaction through interfaces between entities of the system under specification ITU-T Recommendation Z.120 [6];
- specification and Description Language (SDL): partial SDL specification of the behaviour of a system ITU-T Recommendation Z.100 [3].

B.5 The formalization activity

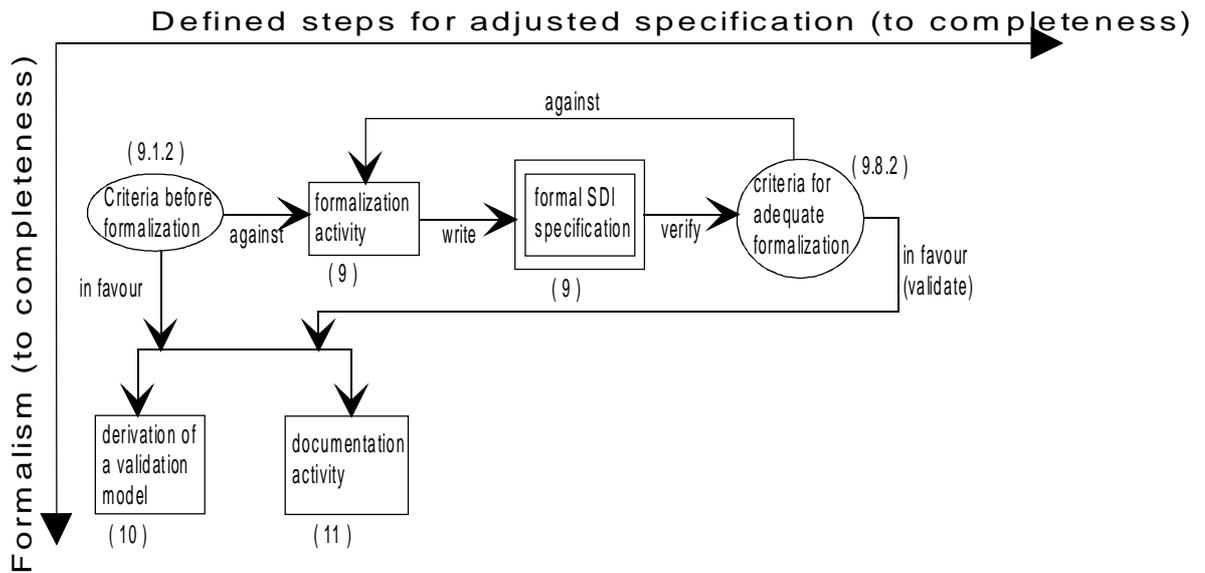


Figure 37: The formalization activity

The formalization process, as shown in figure 37, includes:

- the formal SDL description;
- the analysis of the system guided by the use of SDL: understanding increases and analysis takes place as the SDL is produced. If classification or draft design is omitted then analysis during formalization is more important. The guidelines associated with the formalization steps cover the analysis that is required;
- the checking of consistency: the creating of the SDL description can find inconsistencies and ambiguities in the draft designs, classified information and collected requirements. For each of these inputs, questions, answers, and items to be considered are recorded during formalization. This can lead to changes in any of the inputs and (except where an engineering error has been made in draft design or classification) the collected requirements should be changed.

For example, the formalization activity helps the standard developer on how to avoid pitfalls in ITU-T Recommendation Z.100 [3], how to structure system descriptions in SDL, what is the right level of abstraction using SDL, in which order to produce the different kinds of SDL diagrams, etc.

B.6 The derivation of a validation model activity

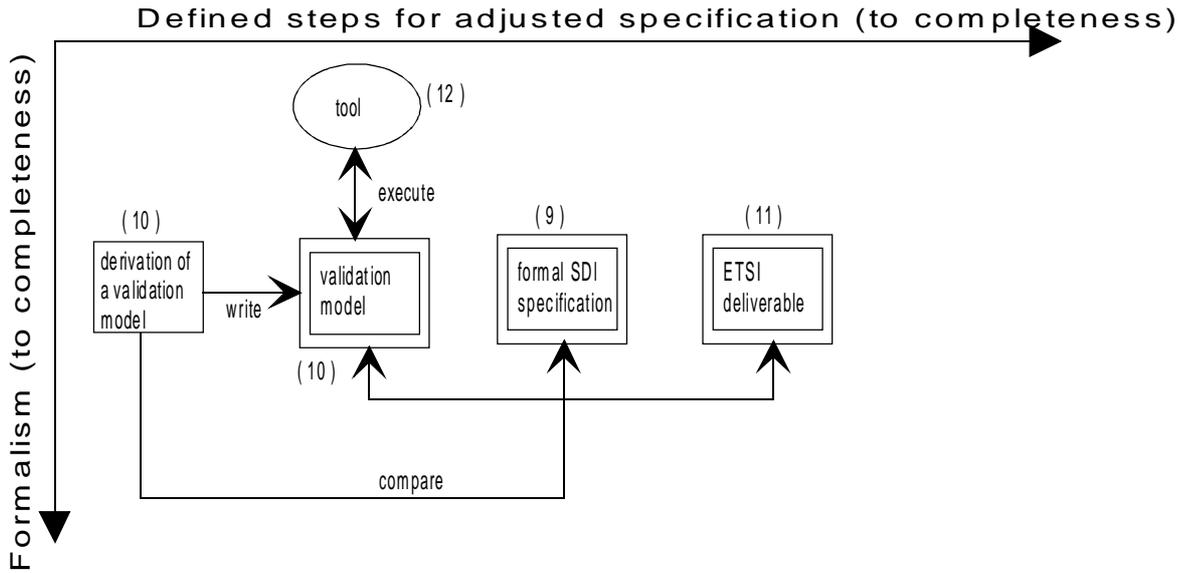


Figure 38: The derivation of a validation model activity

As shown in figure 38, a validation model consists of the SDL description in the ETS (documentation activity) and a minimum of modifications necessary to make the SDL model executable by a tool.

These modifications can be drawn from the formalized SDL model (extracted from the formalization activity), assuming that the formalized model is executable, and in some cases the validation model can be identical to the formalized model.

If the formalized model is not executable, or if it is executable but not practical for validation, a separate model is derived.

B.7 The documentation activity

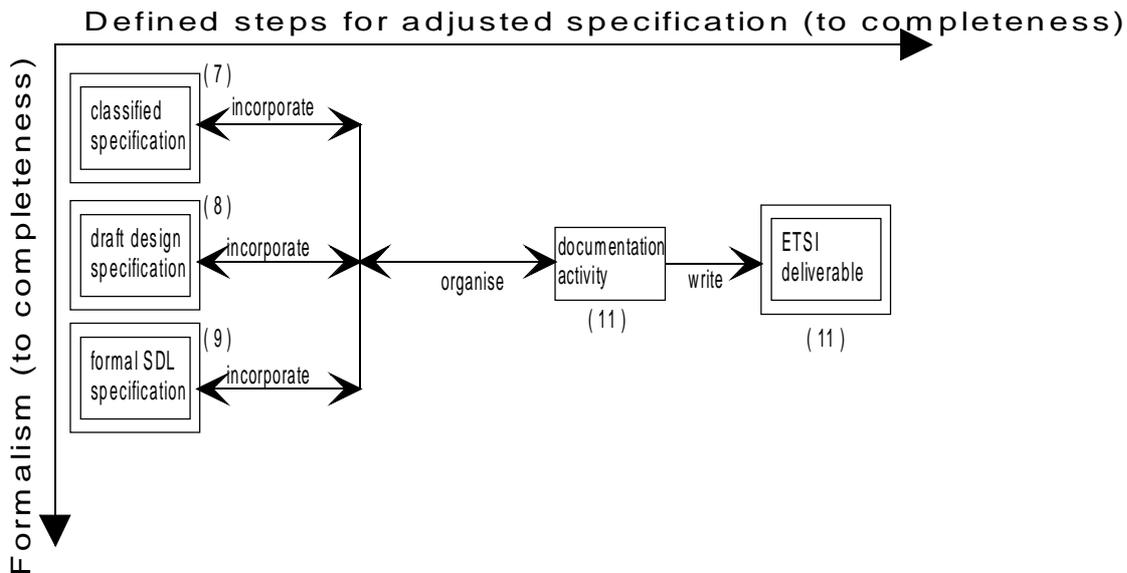


Figure 39: The documentation activity

The documentation activity, as shown in figure 39, organizes and formats all contents (text, figures, tables, etc.) according to ETSI standards pre-defined rules (CEN/CENELEC Internal Regulations [2]) from all specifications provided by the other activities.

The documentation activity assure that it:

- outlines the ETS organising the contents of recommended parts into required clauses and subclauses;
- extracts material from existing text and diagrams (SDL);
- provides needed text (introduction, text produced during other activity processes);
- includes a statement of conformance to the methodology recommendations.

Annex C: List of rules of ETS 300 414 [1]

- Rule 1** Only the graphical representation of SDL should be used.
- Rule 2** The following SDL symbols should not be used in ETSs:
- channel partitioning;
 - **signal** refinement;
 - enabling condition;
 - internal input and output;
 - view and reveal;
 - import and export;
 - service;
 - name class literals.
- Rule 3** A system diagram should be used to describe how the system is composed of functional units (modelled with blocks).
- Rule 4** In a system diagram, the blocks, channels and **signal** definitions should precede the data definitions.
- Rule 5** Block diagrams should be used to describe how the functional units are composed of processes or blocks.
- Rule 6** A block diagram in the normative part of a specification should not have alternative sub-block definitions, i.e. the feature of SDL (in ITU-T Recommendation Z.100 [3] subclause 3.2.2) to describe the decomposition of a block in blocks as well as in processes is not allowed.
- Rule 7** Implicit non-determinism arising as an effect of using channels with delay should be avoided.
- Rule 8** To every normative channel of the standardized system the comment, "normative", should be attached.
- Rule 9** The data types of all parameters of all **signals** relevant to the system and conveyed over normative channels should be specified.
- Rule 10** The data types of parameters of **signals** that are conveyed over normative channels should have a finite size.
- Rule 11** The selection expression in a select symbol should depend on implementation options.
- Rule 12** If the process diagram contains a **create** symbol, this shall be shown in the block diagram by using the **create line** symbol.
- Rule 13** In the normative part of a specification, the maximum number of instances of a process should be limited.
- Rule 14** A comment should be attached to a spontaneous transition that explains the condition for the transition to fire.
- Rule 15** The Boolean expression in a continuous **signal** should not contain **NOW**, **ANY**, imported variables, or remote procedure calls.
- Rule 16** The real time unit of a timer should be supplied using a comment associated with the timer definition.
- Rule 17** A timer should not be started as "set(<Time>, <TimerIdentifier>)" where <Time> is an absolute value.

- Rule 18** The condition of the option symbol will depend on implementation options.
- Rule 19** Task symbols with informal text should not occur in the normative part of a standard.
- Rule 20** Decisions with informal text should not occur in the normative part of the final version of a standard.
- Rule 21** If a decision is non-deterministic (using **any**), a comment should be given explaining the condition.
- Rule 22** The receiving process instance should always be uniquely identified in order to avoid non-deterministic behaviour.
- Rule 23** Message sequence charts should be used to give at least one example of message exchange for each required system function. The message sequence charts should also give examples of message exchange in exceptional situations.
- Rule 24** Message sequences shown in MSCs should be in accordance with the allowed behaviour of instances specified in the related SDL diagrams.
- Rule 25** The following MSC symbols should not be used in ETSs:
- co-region;
 - sub-MSC.
- Rule 26** The instances shown in MSCs should be related to system, blocks, processes or parts of the environment (related to channels connected with the environment) in the related SDL specification.
- Rule 27** Every message in an MSC should be defined in a **signal** definition in the related SDL system or block diagram and as an input and/or output in the related SDL process diagram.
- Rule 28** A timer in an MSC must be defined as a timer in a process that occurs in the SDL diagram that is related to the MSC instance.
- Rule 29** The created MSC instance should be of type "process".
- Rule 30** The following ASN.1 concepts should not be used in combination with SDL: ASN.1 SET, ASN.1 macro, value notation of ASN.1 ANY type, ASN.1 comment.
- Rule 31** ASN.1 types should not be defined with names that differ from the names of other defined types only in the case of the letters that compose the name.
- Rule 32** Names of ASN.1 identifiers that are used in combination with SDL should not contain dashes ("-").
- Rule 33** Names of ASN.1 identifiers that are used in combination with SDL may contain underscores ("_").
- Rule 34** An ASN.1 module that is directly or indirectly imported in an SDL diagram should meet the rules stated in this document, with the exceptions that ASN.1 comments are allowed and names of identifiers are allowed to contain dashes.
- Rule 35** Recursive ASN.1 data structures should not be used in combination with SDL diagrams.
- Rule 36** In a type definition, an identifier should be provided for every component type of an ASN.1 SEQUENCE type.

Rule 37 An ASN.1 type should not be composed from two subtypes using the or operator ("|").

Annex D: List of rules in formalization

- Rule 1** The system boundary defines what is going to be specified (described). Communicating entities within a system have to be specified (described) irrespective of whether they are normative or informative. Entities outside the boundaries are assumed to exist, but cannot be described in SDL. The communication is possible only by means of exchanged discrete messages. The message in SDL is called signal;
- Rule 2** The unit for time data should be recorded in a comment in the system diagram.
- Rule 3** The textual definitions of a diagram should be placed in text symbols inside diagrams.
- Rule 4** Each type of definition (for example: **signal** definitions, **signallist** definitions, data definitions, etc.) should be placed in a different text symbol. If textual definitions occupy more than 50 % of a diagram, it should have separate pages for each type of definition.
- Rule 5** There should be no more than five blocks at the system level (or directly enclosed within a block).
- Rule 6** A definition should have the smallest scope that includes all uses of the defined item.
- Rule 7** If a block (process or procedure) is informative and is not part of an enclosed informative block (process or procedure), it should have the annotation "informative" in the diagram referencing it or in its referenced diagram or in both places.
- Rule 8** There should be only one channel between two blocks.
- Rule 9** Every channel that is normative should have the comment "normative" attached.
- Rule 10** No more than three **signals** (or **signallists**) should be listed in a **signallist** symbol, instead use a **signallist** attached to the channel. In principle it is better to use **signallists** consistently.
- Rule 11** The **signallists**, **signals** and data used in all communication of a system should be defined in one (or more) text symbol(s) separate from other definitions.
- Rule 12** The diagrams should be nested by reference rather than direct enclosure.
- Rule 13** The number of channels from each block should be no more than five.
- Rule 14** For each block, at least one process should have its initial number of instances greater than zero, so that it can create other instances in the block.
- Rule 15** The number of process definitions in each block should be no more than five.
- Rule 16** Spontaneous transition should not be used in normative parts of the standard.
- Rule 17** The MSC should either be correct traces of the handling of messages by the SDL system or be clearly annotated to indicate how and why they differ from the SDL behaviour.
- Rule 18** All states in a process should be reachable from the start of the process.
- Rule 19** A procedure that is exported by a process (to be used as a remote procedure) should not be saved in every state of that process.

- Rule 20** Each **signal** received by the process should have at least one input leading to a non empty transition.
- Rule 21** if bit tables have been used to define data then these should be converted to ITU-T Recommendation X.208 [9] or ITU-T Recommendation Z.100 [3].
- Rule 22** the names of existing sorts of data should not be used for new sorts of data even if they have different scopes.
- Rule 23** a process parameter should not contain the PId of the process that has created it because this value is returned by **parent** operator.
- Rule 24** A **signal** parameter should not contain the PId of the process that has sent the **signal** because this is value is returned by **sender** operator.
- Rule 25** a value or decision that is non-deterministic (using **any**) should always have a comment attached explaining how the choice is made.
- Rule 26** If a parameter is omitted in an output, there should not be a corresponding input that expects a value for this parameter.
- Rule 27** Addressing information should always be defined in an output.
- Rule 28** SDL axioms should not be used to define operator properties.

Annex E : Allowed symbols

Table 4: Symbols allowed in a system diagram

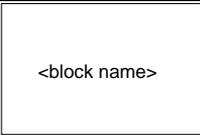
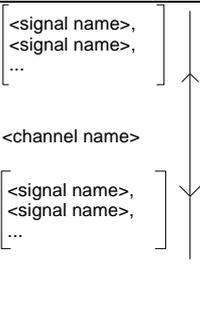
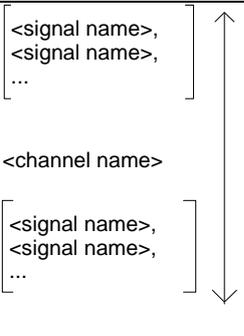
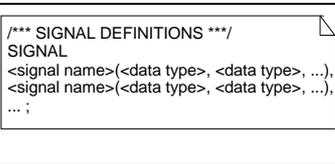
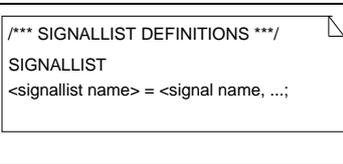
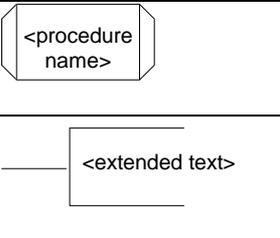
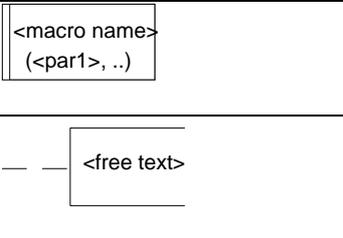
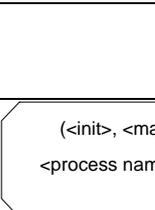
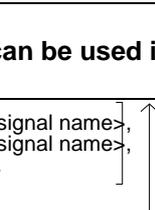
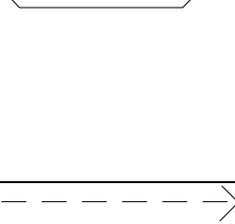
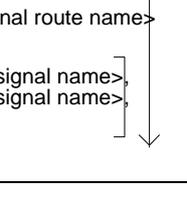
	<p>Block symbol</p>		<p>Text symbol</p>
	<p>Channel with delay symbol</p>		<p>Channel without delay symbol</p>
	<p>Signal definitions in text symbol</p>		<p>Signallist definitions in text symbol</p>
	<p>Select symbol</p>		<p>Data type definitions in text symbol</p>
	<p>Procedure reference symbol</p>		<p>Macro call symbol</p>
	<p>Text extension symbol</p>		<p>Comment symbol</p>

Table 5: Additional symbols that can be used in a block diagram

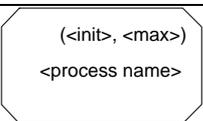
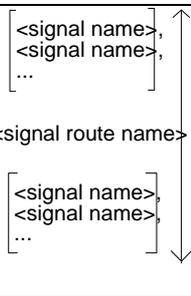
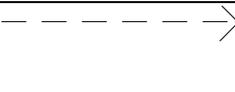
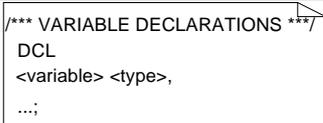
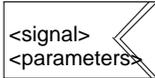
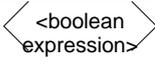
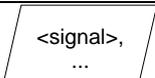
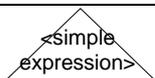
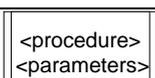
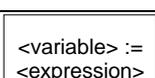
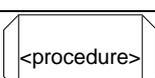
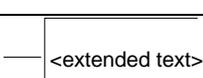
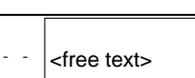
	<p>Process symbol</p>		<p>Signal route symbol</p>
	<p>Process create line symbol</p>		

Table 6: Allowed symbols in process diagrams

	Process start symbol		Process stop symbol
	State symbol		Variable declarations in text symbol
	Input symbol		Output symbol
	Priority input symbol		Continuous signal symbol
	Spontaneous transition symbol		Save symbol
	Decision symbol		Option symbol
	Procedure call symbol		Process creation symbol
	Task symbol		Procedure reference symbol
	Text extension symbol		Comment symbol
	Join symbol		

The symbols that can be used in MSCs are shown in table 7.

Table 7: Allowed symbols in MSCs

	<p>Instance symbol</p>		<p>Signal exchange</p>
	<p>Text extension symbol</p>		<p>Comment symbol</p>
	<p>Timer symbol</p>		<p>Condition symbol</p>
	<p>Action symbol</p>		
	<p>Process creation line symbol</p>		<p>Process end symbol</p>

Annex F: Bibliography

- Belina, F., Hogrefe, D., Sarma, A. (1991): "SDL with Applications from Protocol Specification".
- Bræk, R., & Haugen, Ø.(1993): "Engineering real time systems: An object-oriented methodology using SDL".
- Davis A. et al.: Proc. Software Metrics Symp., IEEE CS Press (1993): "Identifying and Measuring Quality in a Software Requirements Specification".
- Færgemand, O. (1991): "Stepwise production of an SDL description" in "Formal Description Techniques III".
- Rumbaugh, J.(1991): "Object-Oriented Modeling and Design".
- Sarracco, R. et al. (1989): "Telecommunication system engineering using SDL".
- Steedman, D. (1990): "Abstract Syntax Notation One (ASN.1) - The tutorial and reference", Technology Appraisals Ltd.
- Lacoste, Gerard (1993): "The SCORE Service Creation Process Model" IBM.
- Reed, R. et al. (editors) (1993): "Specification and programming Environment for Communication Software".
- SPECS-Specification Generation External deliverable D.WP3.8 (1992): "Final Methods and Tools for the Generation of Specifications", GSI-Tecsi.
- SPECS-Specification Handling External deliverable D4.15 (1992): "Final Methods and Tools for the handling of SDL of Specifications", GSI-Tecsi.
- Færgemand, O. et al., (1994): "Systems Engineering Using SDL-92".
- Tilanus, P. (1994): "How to get Complete ADT Definitions? A Tutorial" in "SDL 91 Evolving Methods".
- Turner, K. (editor), (1992): "Using Formal Description Techniques", John Wiley & Sons.
- West, C. Computer Networks and ISDN Systems, 24, 219-242 (1992): "Protocol validation - principles and applications".

History

Document history	
September 1996	First Edition