Temporary document **003**

European Telecommunications Standards Institute

MTS#31

24 to 26 October 2000
Sophia-Antipolis

Source:	**STF141 Leader**
Title:	**DEG/MTS-00062: Methodological approach to the use of object-orientation in the standards making process**
Date:	**22 September 2000**
Document for:	**Approval**
Agenda item:	**6.4**

DES/MTS-00062 V2.0.9 (2000-09)

Methods for Testing and Specification (MTS);
Methodological approach to the use of object-orientation
in the standards making process;

European Telecommunications Standards Institute

Reference
DES/MTS-00062

Keywords
Unified Modeling Language, UML, protocol, testing, Object Orientation, specification, methodology

ETSI Secretariat

Postal address
F-06921 Sophia Antipolis Cedex - FRANCE

Office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16
Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

X.400
c= fr; a=atlas; p=etsi; s=secretariat

Internet
secretariat@etsi.fr
http://www.etsi.fr

Contents

Intellectual Property Rights

Foreword

Introduction

1 Scope

The present document describes a methodological approach to the use of object-orientation, and, in particular the Unified Modeling Language (UML), in the ETSI standards-making process. The purpose of the document is to establish a set of guidelines that provide the user with a framework within which the concepts of the UML can be used effectively in the development of ETSI standards.

The guidelines presented in this document are intended primarily for use in the production of standards specifying communications protocols. However, they could be applied in part to the use of UML to other types of standard where this is deemed to be appropriate and beneficial.

This document presents a straightforward process for using the UML from the collection of the initial broad requirements through to the point where it is necessary to begin describing detailed behaviour.

The application of the UML to the development of a protocol standard does not preclude the use of the Specification and Description Language (SDL). The methodological approach described in the document can be used in conjunction with the guidelines specified for the use of SDL given in ETR 298 [2], EG 201 015 [3], EG 201 383 [4] and EG 202 106 [5].

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, subsequent revisions do apply.
- A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[1] ETR 266 (1996): "Methods for Testing and Specification (MTS); Test purpose style guide"

[2] ETR 298 (1996): "Methods for Testing and Specification (MTS); Specification of protocols and services; Handbook for SDL, ASN.1 and MSC development"

[3] EG 201 015 (1999): "Methods for Testing and Specification (MTS); Specification of protocols and services; Validation methodology for standards using Specification and Description Language (SDL); Handbook".

[4] EG 201 383 (1999): "Methods for Testing and Specification (MTS); Use of SDL in ETSI deliverables; Guidelines for facilitating validation and the development of conformance tests".

[5] EG 202 106 (1999): "Methods for Testing and Specification (MTS); Guidelines for the use of formal SDL as a descriptive tool".

[6] TCR-TR 011 (1993): "Business Telecommunications (BT); Private Telecommunication Network (PTN) internal mobility; Private user mobility and cordless terminal mobility; General principles and service aspects".

[7] ITU-T Recommendation Z.109 (2000): "SDL in combination with UML"

[8] ISO/IEC 17875 (2000): "Information technology -- Telecommunications and information exchange between systems -- Private Integrated Services Network -- Specification, functional model and information flows -- Private User Mobility (PUM) -- Registration supplementary service"

[9] ISO/IEC 9646: "Information technology – Open Systems Interconnection – Conformance testing methodology and framework"

[10] ISO/IEC 17876 (2000): "Information technology -- Telecommunications and information exchange between systems -- Private Integrated Services Network -- Inter-exchange signalling protocol -- Private User Mobility (PUM) -- Registration supplementary service"

[11] Jacobson, Booch & Rumbaugh: "The Unified Software Development Process", Addison-Wesley (1999), ISBN 0-201-57169-2

3 Definitions, symbols and abbreviations

3.1 Abbreviations

For the purposes of the present document, the following abbreviations apply:

HDB	Home Data Base
CP	Coordination Point
IUT	Implementation Under Test
MSC	Message Sequence Chart
MTC	Main Test Component
HDB	Home Data Base
PCO	Point of Control and Observation
PISN	Private Integrated Services Network
PINX	Private Integrated services Network eXchange
PTC	Parallel Test Component
PTN	Private Telecommunication Network
PTNX	Private Telecommunication Network eXchange

Note: Since the publication of TCR-TR 011 [6] in 1993, the terms PTN and PTNX, which were used extensively in that document, have been replaced by PISN and PINX in the context of Corporate Network standardization. Throughout this document, PISN and PINX have been used as the current terms.

PUM	Private User Mobility
PUMR	PUM dynamic Registration
SDL	Specification and Description Language
TTCN	Tree and Tabular Combined Notation
UML	Unified Modeling Language
VDB	Visitor Data Base

3.2 Definitions

For the purposes of the present document, the following definitions apply:

actor: an abstraction for entities outside a system or subsystem that interact directly with that system or subsystem

artefact: a piece of information that is used or produced during the development of a standard

Note: This definition of the term "artefact" is commonly used in the context of the UML. Examples of artefacts are models, textual descriptions, standards and external documents.

domain model: a related set of UML diagrams and text which together identify at a high level of abstraction, the logical and physical entities of a system and the relationships between them

feature: a candidate requirement

postcondition: a constraint that must be true at the completion of an operation

precondition: a constraint that must be true when an operation is invoked

QSig: a corporate network signalling system defining basic and supplementary service protocols at the Q-reference

requirements model: a set of UML diagrams and text which together elaborate the requirements to be met by a standardized system

use case: the specification of sequences of actions that a system or subsystem can perform by interacting with actors

user: a human being or an item of equipment to which a service is provided

4 A methodology for the use of the UML in telecommunication standards development

4.1 Introduction

The UML is a powerful, graphical language that can be used effectively and beneficially within the ETSI standards making process, particularly in the specification of communication protocols. This document presents a general framework within which the UML can be applied to this process but the three following points should be considered before making a commitment to its use:

1. the UML is an ideal language for the collection, analysis and processing of requirements. Consequently, the process described here introduces formality to the early stages of the standards development where such formalism has not generally existed in the past;

2. particular UML diagram types are recommended at each stage of the process but this should not be regarded as "set in concrete". If different UML diagram types appear to be more appropriate or meaningful in particular situations then these should be used;

3. The use of the UML in the standards making process should not imply that the UML diagrams produced must appear in the standard, although that, too, is not precluded. The language should be regarded as a valuable tool for producing standards of a high quality and not just another means of drawing diagrams to describe protocols.

4.2 A process based upon the UML

The UML is a modelling language and is not, itself, a development process. It is possible to think of it as a set of individual diagram types and symbols which together make up the language and which can be used in an ad hoc manner wherever and whenever an opportunity arises. However, greater benefits can be gained if it is considered as the basis for a straightforward process for the overall development of telecommunication standards. It is just such a process which is described here. The process has been derived from the Unified Development Process [11] with some modifications to reflect the specific requirements of standards development. There are many different types of UML diagram which can be produced within the process but these have been segregated into three overall modelling stages as follows:

- Context Modelling

  the collection, refinement and expression of ideas and existing knowledge to establish the objectives of standardization project.

- Requirements Modelling

  the further processing of the Context Model to establish and express a set of achievable technical requirements to be met by the protocol standard(s).

- Specification Modelling

  the extension and refinement of the Context Model and the Requirements Model to provide sufficient detail for the development of a behaviour model.

Throughout this document, UML activity and package diagrams are used to illustrate the use of the UML in the standards development process. Figure 1 shows the three discrete models as simple packages.

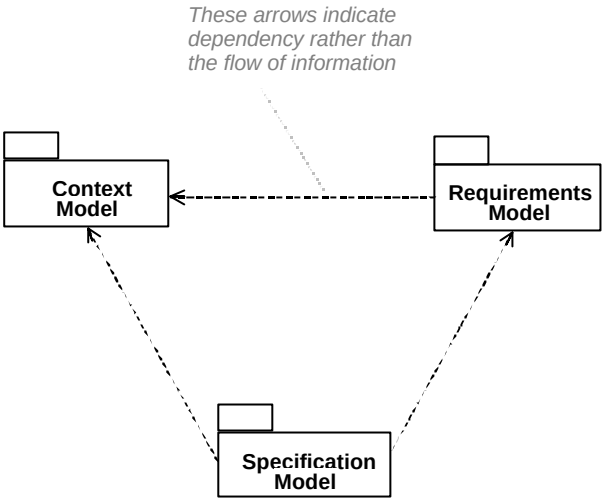

Figure 1: UML models required in the standardization process

Figure 2 presents an overview in graphic form of a process for using the UML to produce these models.

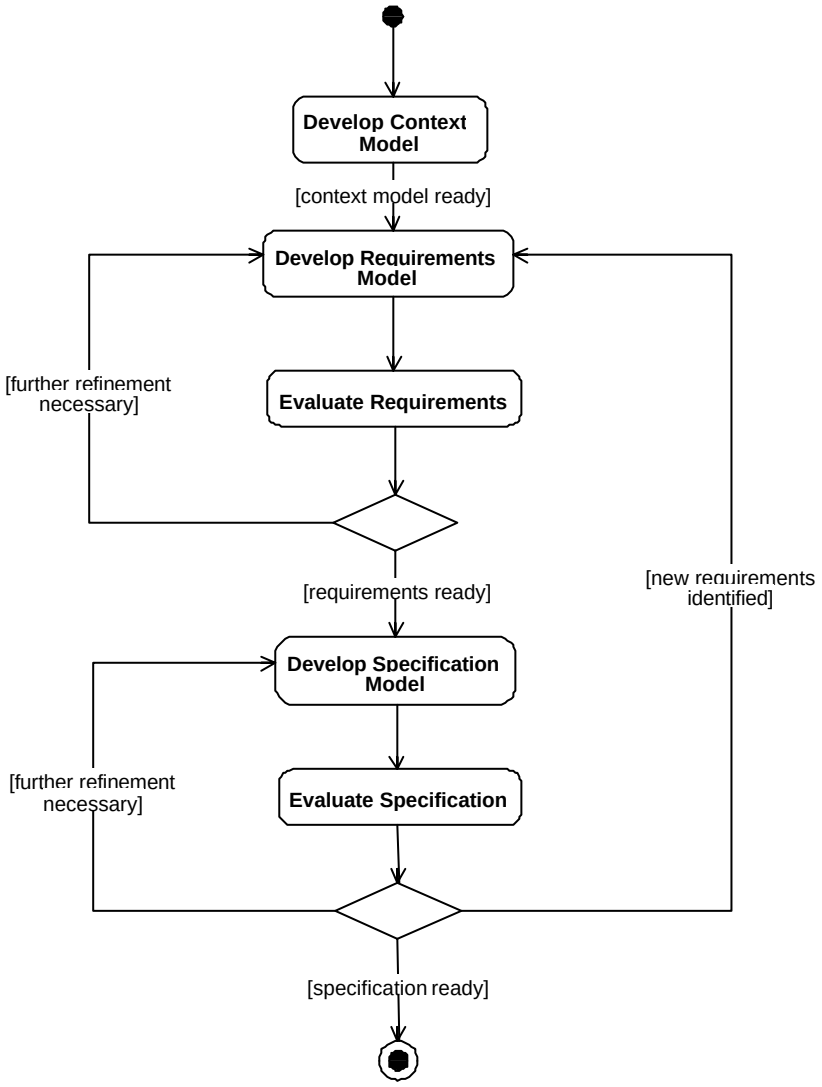

Figure 2: A process using the UML in writing standards

The process involves the following steps:

1. carry out an initial study to produce a Context Model comprising a list of desired features and a domain model based on existing knowledge and experience;

2. model the requirements for the standard so that the requirements can be evaluated and refined;

3. use these requirements in the production of a specification model of the system on which the standard is to be based;

4. continuously evaluate and refine the specification and, consequently, the requirements.

In each of these activities there will be a number of different types of UML diagram and textual descriptions produced. The following list indicates which are the most likely to be useful at each stage but this does not preclude the inclusion of any UML diagram type at any stage if its use is likely to improve the understandability of the overall specification:

- Context Model:
  - class diagrams;
  - object diagrams;
  - text;
- Requirements Model:
  - use case diagrams;
  - activity diagrams;
  - text;
- Specification Model:
  - class diagrams;
  - sequence diagrams;
  - collaboration diagrams;
  - statechart diagrams;
  - text.

Both the Requirements Model and the Specification Model should form the main input to the development of detailed behaviour specifications of the standardized system and of a corresponding conformance test suite.

It is unlikely that a single pass through this process will result in a fully specified protocol so it should be used iteratively to refine the requirements and the definition.

4.3 Examples based on the Private User Mobility (PUMR) supplementary service

In order to illustrate each of the stages involved in developing UML Context Models, Requirements Models and Specification Models, the QSIG Private User Mobility (PUM) supplementary service, PUM Dynamic Registration (PUMR) has been used as an example throughout this document. A pre-normative study of mobility issues within private networks, TCR-TR 011 [6], was produced by ETSI Technical Committee, Business Telecommunications (TC-BTC). The PUMR inter-PINX protocol standards, ISO/IEC 17875 [8] and ISO/IEC 17876 [10] were produced by TC-BTC and ECMA technical committee TC32. The UML examples in this document and the complete example shown in Annex A are based on the relevant parts of TCR-TR 011 and the protocol standards.

4.4 Develop a Context Model

4.4.1 Activity overview

The development of a Context Model includes the collection of a list of features as well as the development of a Domain Model. As is shown in Figure 3, both activities take place in parallel. In most cases, it is possible to begin collecting and evaluating desired features while developing a Domain Model based on information already known about the system.

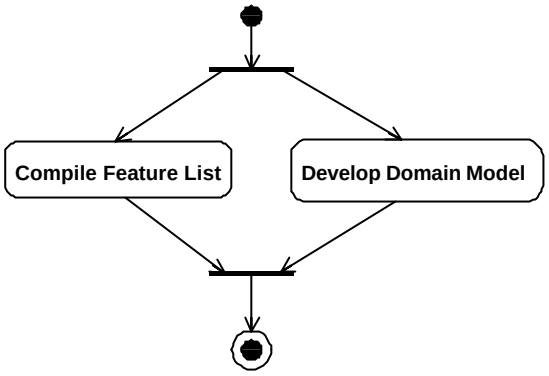

Figure 3: The Context Modelling process

4.4.2 Artefacts

The following artefacts are produced as part of the Context Model (Figure 4):

- a Domain Model consisting of class and object diagrams;
- a feature list

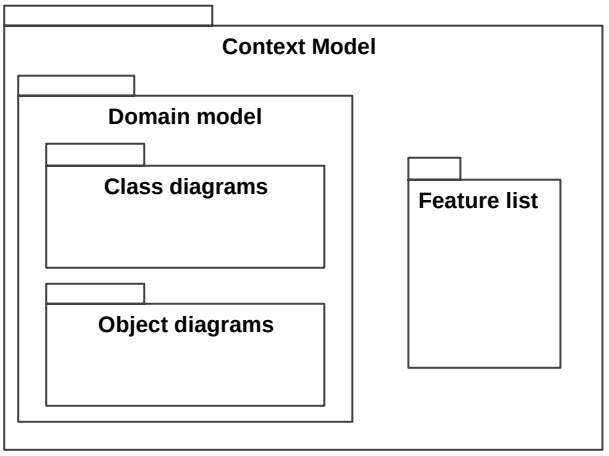

Figure 4: Artefacts produced as part of the Context Model

4.4.3 Compile feature list

During the standardization process, different parties come up with many ideas for features to be standardized. These features are used as the basis of discussions from which the requirements for a standard emerge. The UML does not provide a suitable graphical model for the collection and management of features. However, they are an important input to the overall process described here for the use of the UML. Therefore, it is useful to collect all desired features into a feature list. From that collection, each feature is evaluated and its status recorded to indicate whether it has been selected for inclusion in the next release of the standard, deferred to a subsequent release or rejected altogether. This process is shown graphically in Figure 5.

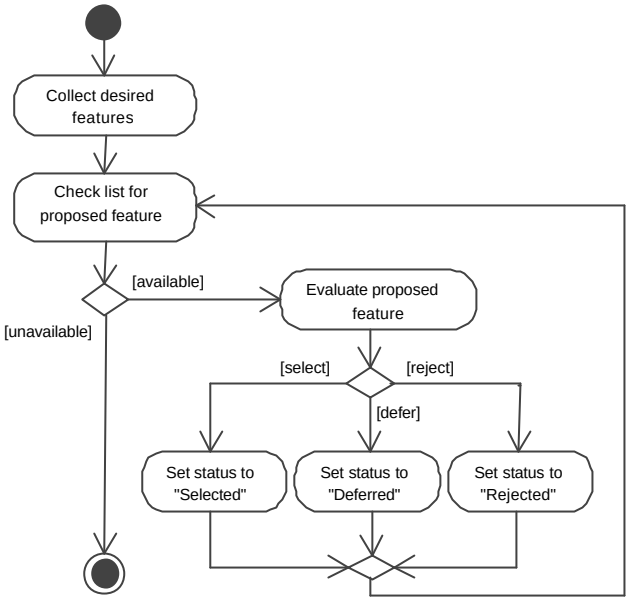

Figure 5: The feature evaluation process

Additional status indications should be given to those features which have not yet been evaluated and to those which have been implemented in a specific release of the standard.

Each item in a feature list should be assigned a number and a brief definition along with information regarding its origin and priority.

4.4.3.1 PUMR example

An example of how a feature list could be structured is shown in Table 1.

Note: The *Priority* and *Status* columns contain values which have been inserted for illustrative purposes only and do not reflect the real-world situation.

Table 1: PUMR feature list

No	Feature	Priority	Status
1	A PUM user should be able to register at any capable (wired or wireless) terminal within the PISN	1	Selected
2	A PUM user registered at another terminal should have a service profile which is as close as possible to the service profile offered at the user's normal point of connection to the network (i.e., at the user's home location)	2	Deferred
3	Registration should be for incoming calls, outgoing calls or both incoming and outgoing calls	1	Selected
4	Registration for outgoing calls may be limited by the PUM user to a preset period of time (duration) or a specific number of calls	3	Rejected
5	Giving an alternative identifier for the PUM registration requires the Visitor PINX to enquiry to the Directory PINX to obtain the PUM user's PISN number	2	Selected
6	Registration for incoming calls should always be mandatory	1	Selected
7	The security mechanisms provided by PUM to support mobility services should at least be as good as for existing services	2	Selected
8	For the purposes of security at registration, a PUM user should be able to register using the user's assigned PISN number or an alternative identifier	1	Deferred
9	A PUM user with a high security level should not be precluded from using an ordinary terminal	1	Deferred
10	Before registering to another Visitor PINX, a PUM user should be de-registered from a Previous Visitor PINX	3	Selected
11	A PUM user may register directly from a Visitor PINX or indirectly from a Remote PINX via a Visitor PINX, to the Home PINX	3	Deferred
12	Security mechanisms should not appear as complicated procedures to the PUM users but they should be a part of the general PUM procedures	2	Rejected
13	For the purposes of security, it should be possible to request the provision of a Personal Identification Number (PIN) in addition to the PUM user's identity (PISN number or alternative identifier)	3	Deferred
14	A PUM user may be offered a set of possible optional security mechanisms to decide upon, for authentication and access control	3	Rejected
15	The PUM user should be able to move between terminals during an active call (change of access point)	2	Deferred
16	Several PUM users may register for incoming calls at the same terminal access simultaneously	3	Deferred
17	The PUM user should be able to specify different terminal accesses according to the feature (service type) requested	2	Selected
18	Bearer services offered to a PUM user should include at least a 64 kbit/s circuit mode, a 3,1 KHz audio, and speech telephony service	3	Selected
19	To register, a PUM user should send a message to the PISN containing e.g. its PUM number, the identifier of the terminal, the indication of the PUM feature (e.g. registration for incoming and/or for outgoing calls)	2	Selected
20	The PUM user's own number is used as the basis for accounting, independent of any terminal or PINX used by the PUM user	2	Deferred

As can be seen from this list, there is no limit placed on the level of detail which can be included as a desired feature. Low-level descriptions such as the requirement for a PIN are equally as valid at this stage as high-level ones such as the ability to register at any terminal.

4.4.4 Develop Domain Model

As shown in Figure 6, the development of a domain model is done in three steps:

1. identification of communication entities and communication paths;
2. identification of possible system architectures;
3. identification of interfaces.

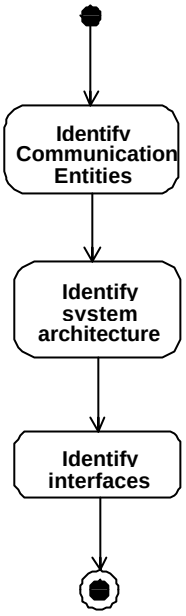

Figure 6: Domain Modelling process

Domain models can be developed for different levels of abstraction but the final domain model should consist of a reasonable selection of class and object diagrams. However, the purpose of a domain model is the development of an overview of a communication system and should not include detailed specification. As a guide, a domain model should contain no more detail than would normally be presented in a pre-normative study report.

Within ETSI, there is a large and valuable base of knowledge and experience which is the result of producing numerous standards for a wide range of communication technologies. This knowledge and experience should be used to simplify the development of the domain model by providing "short-cuts" to possible solutions. As an example, when specifying a new protocol for an emerging technology, it is not necessary to redesign the ISO Layered Model as its application in this area is already well understood.

4.4.4.1 Identify communication entities

The high-level structure of a specification's context can be described with a domain model which is represented using UML class diagrams. Associations are used to express the relationships between entities in the domain diagram. A generic domain model class diagram for communication systems is shown in Figure 7.

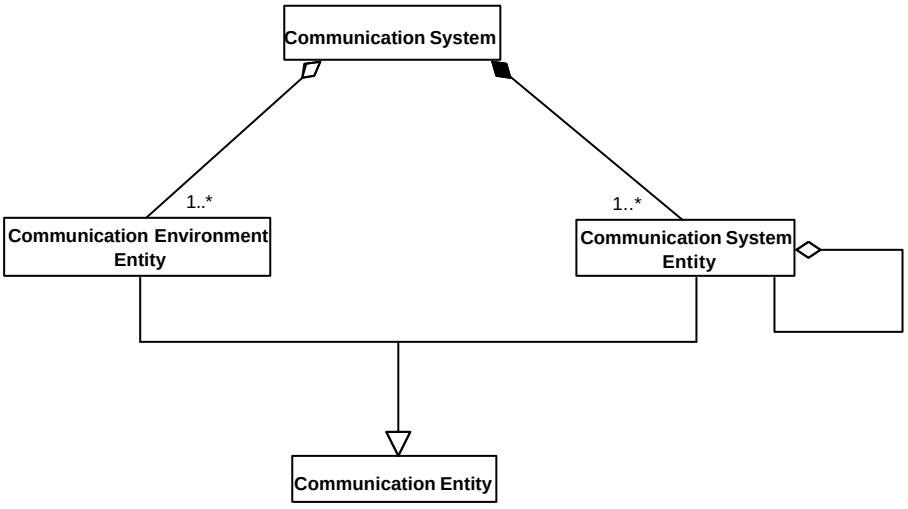

Figure 7: Generic class diagram for communication systems

The Communication System encapsulates the whole system for which a standard is to be specified. It contains Communication System Entities which can, themselves, be composed of other (sub-)entities. Communication Environment Entities lie outside of the system.

Both Environment and System Entities are generalisations of the same abstract base class, Communication Entity. Communication entities are associated with other such entities. This means that communication paths exist between instances of communication entities as shown in Figure 9. Communication paths use interfaces to exchange signals, however, the interfaces are realised by the communication entities. This is illustrated in Figure 8.

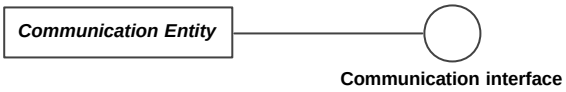

Figure 8: Communication entities realise interfaces

4.4.4.2 Identify system architecture

Step two in the development of a domain model is the collection of possible system architectures. This information is then used in the third step to identify interfaces. Figure 9 shows an architecture where two terminals are connected to a pair of interconnected exchanges.

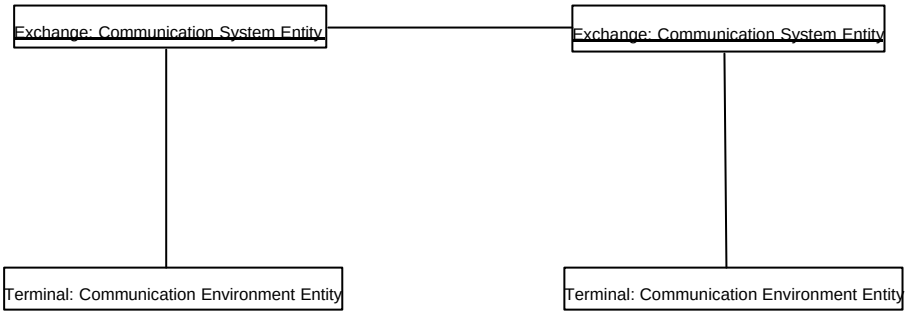

Figure 9: Object diagram showing system architecture

4.4.4.3 Identify interfaces

From the object diagrams developed in the previous step (subclause 4.4.4.2), interfaces can be identified. There are two kinds of interfaces: Normative and non-normative ones. Normative interfaces are the subjects of standardization; non-normative interfaces are either proprietary or standardized in a different document.

As a general rule, system entities exchange information with one another through normative interfaces and they communicate with environment entities through non-normative interfaces. Nevertheless, there may also be non-normative interfaces between system entities.

Note: Due to the interpretation of signals being modelled as operation calls on objects (see subclause 4.6.3.1.1), each communication path has to be seen as the combination of two interfaces, one on each end of the path.

4.4.4.4 PUMR example

Figure 10 shows the top-level class diagram of the context in which PUMR will be found. It contains the following information: On the system level, there is the Private Integrated Service Network (PISN). A PISN system has associations with terminals and Private Integrated services Network eXchanges (PINX). Terminals are stereotyped as environment entities, meaning that their behaviour will not be specified within the model. Nevertheless, there would be no use of the PISN without terminals, that is why they are associated with the system through an aggregation. PINXs are communication system entities and their association with the PISN through composition shows that they are the building blocks of the system; without the exchanges there would be no network.

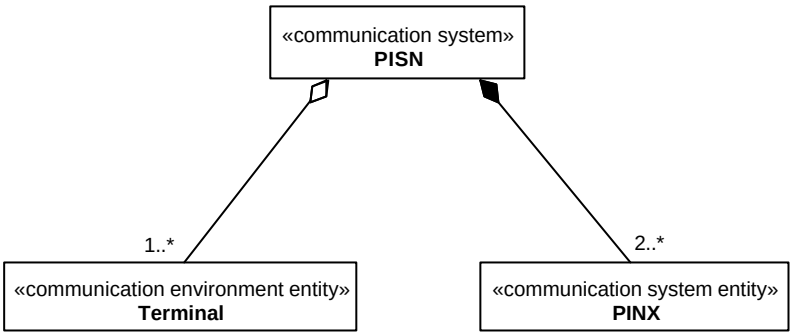

Figure 10: System-level class diagram for the PUM Registration supplementary service

The feature list in Table 1, the class diagram in Figure 10 and existing knowledge of similar systems and technologies together provide the basis for the object diagram in Figure 11 which represents the basic architecture of PUMR. The class diagram shows that there are two basic classes, PINXs and terminals. Each of the objects shown in Figure 11 are instances of one or other of these classes.

The main feature of PUMR is the ability of a PUM user to register at any terminal connected to the PISN. Existing knowledge of the GSM network architecture and protocol has been used in the development of the Home PINX/Visitor PINX/Previous Visitor PINX architecture represented in Figure 11. Feature 5 from the PUMR feature list in Table 1 has led to the addition of the Directory PINX.

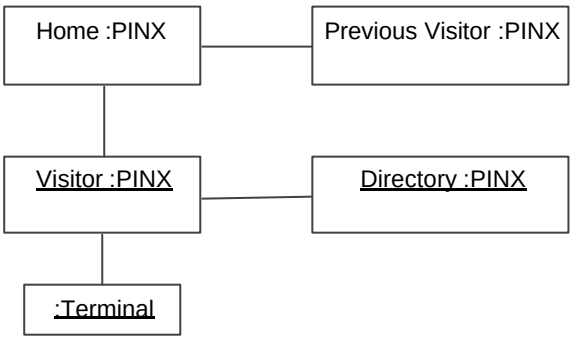

Figure 11: Basic PUMR object diagram

From Figure 11, three functional entities which are necessary for the specification of the PUMR supplementary service can be identified. These are the Home location, the Visitor location and the Directory function (assuming that a Previous Visitor is also a Visitor). These entities communicate with one another which means that each of them is required to handle a specific set of signals. Using the UML, these sets of signals can be represented by interfaces. Figure 12 shows that a PINX entity realises three PUMR interfaces, one for each of the Home, the Visitor and the Directory function. These interfaces are collected together in a package to build the PUM Registration supplementary service. Since this service is the subject of the standardization effort, the PUMR interfaces are normative. In addition, there is a non-normative interface between a PINX and a terminal.

Note: At the domain level, classes do not represent physical objects. While Figure 12 suggests that every PINX has to be able to act as a Home, Visitor and Directory entity, this does not have to be the case during the deployment of physical exchanges.

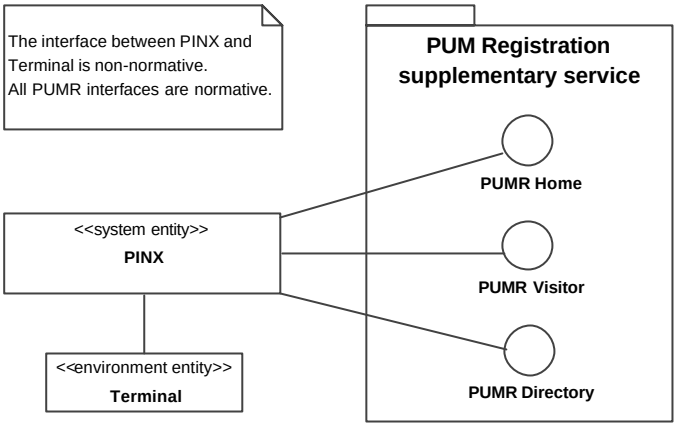

Figure 12: Identification of interfaces for PUMR

4.5 Develop a Requirements Model

4.5.1 Activity overview

The purpose of the Requirements Model is to evaluate the list of features developed as part of the Domain Model and to elaborate them as formal requirements. A process for developing a Requirements Model is shown in Figure 13.

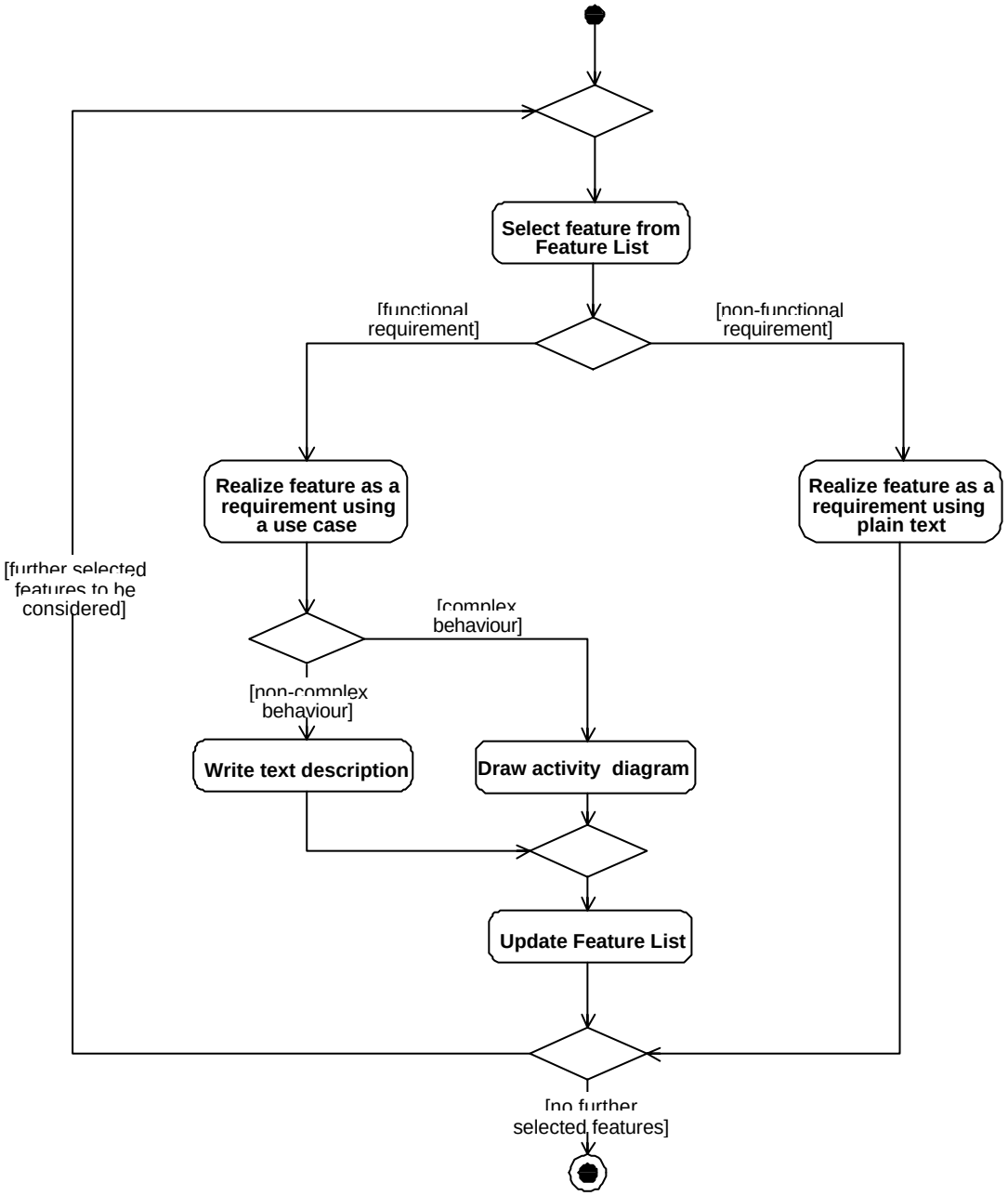

Figure 13: The Requirements Modelling process

4.5.2 Artefacts

By analysing and developing the selected features from the Feature List, it is possible to specify a set of requirements for the protocol to be standardized. As can be seen in Figure 14, requirements can be broadly classified as either functional, which can be described with UML use cases, or non-functional which can only be described in plain text.

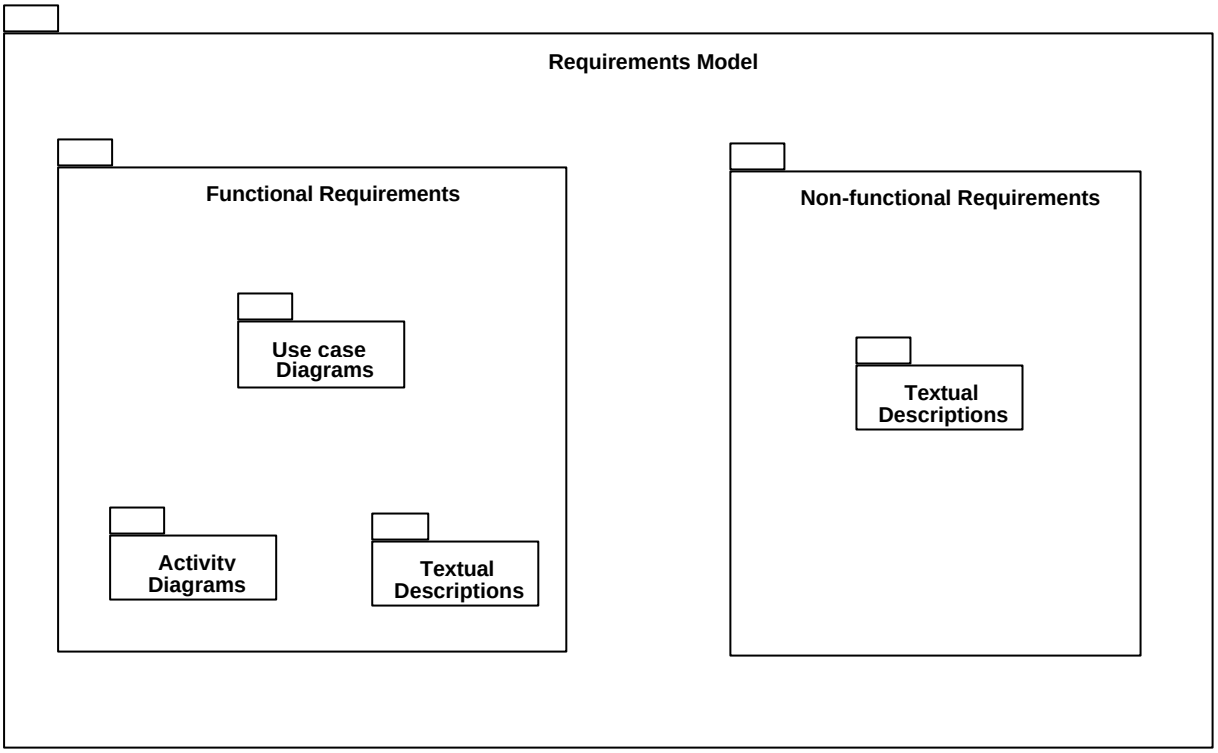

Figure 14: Artefacts produced as part of the Requirements Model

When specifying a protocol system, it is often the case that both functional and non-functional requirements are identified. As an example, it is clear that the feature "For the purposes of security at registration, a PUM user should be able to register using the assigned PISN number or an alternative identifier" shown in Table 1 will result in a number of functional requirements whereas, "Bearer services offered to a PUM user should include at least a 64 kbit/s circuit mode, a 3.1 kHz audio and speech telephony service" probably will not.

4.5.3 Collect functional requirements

4.5.3.1 Develop use cases

Use cases make it possible for requirements to be captured in a structured way. They usually consist of a textual description, but activity diagrams can be used to represent activities inside the system and the interaction of the system with actors.

The Feature List and the Domain Model should be used as the starting point for the development of use cases in the Requirements Model. The Domain Model provides the context in which selected features are developed into requirements for the standard.

The development of use cases is an iterative process which involves the following activities:

- identification of actors;
- identification of use cases;
- description of each use case.

Generally, only a few use cases will be found in the first iteration; new ones will be added during subsequent passes and the existing ones will probably need to be refined.

4.5.3.2 Identifying actors

Actors are used to represent external systems or some internal parts of a system which use a particular subsystem. There can also be actors which are related to system initialization and maintenance.

Two criteria should be used in finding actors:

1. for every actor there should be at least one user which will enact the role. A user in this context can be at any level of abstraction, for example, a mobile terminal or protocol layer;

2. there should be minimal overlap of roles between actors. This prevents having two actors that have essentially the same role.

All actors should be given relevant names and short textual descriptions of the role they play and how they use the system.

In the PUM Dynamic Registration example, the actor is the PUM User.

4.5.3.3 Identifying use cases

The identification of use cases is not a simple task. The following guidelines may be helpful:

1. Review the Feature List which has been compiled during domain modelling as a source for system requirements;

2. Consider each service provided by the future system as a good starting point for use case identification;

3. Consider the actor's point of view. What do actors want to do with the system?

The identification of possible use cases in a system can be simplified by addressing only one system service in each use case and by considering only the primary actors as they will initiate most of the use cases.

Every actor needs one or more use cases to fulfil its needs. Each candidate use cases identified in this way will not necessarily become a unique use case as it may be possible for some to be incorporated into other use cases. A potential use case that appears complete in itself should be identified separately, whereas one that always follows as a continuation of another should probably be combined with it.

The choice of a name for a use case can help considerably in making the model easy to understand. Use case names should clearly identify the function represented by it and, in most instances, should start with a verb.

4.5.3.4 PUMR Example

In order to illustrate the process of developing use cases, an example has been taken from TCR-TR 011 [8]. This example deals with the requirements for PUM Dynamic registration for incoming calls identified on page 24 of the document. These requirements are identified as follows:

- the PUM user can specify a terminal access to which some or all incoming calls to the PUM user will be presented;

- a different terminal access may be specified for each service type (e.g. voice, telefax);

- the PUM user will be able to determine the desired "service profile" attached to this new registration, i.e., depending on the calling party's identity, call importance indication, for "no answer" and "busy" conditions, and other possible criteria;

- several PUM users may register for incoming calls at the same terminal access simultaneously;

- in addition to new facilities brought by the PUM service, the supplementary services usually offered to any PISN user should be made available to PUM users.

There are three use cases that can be defined for PUM Dynamic Registration, as follows:

Use case 1: Specify Access Point for Incoming Calls

The PUM user specifies a terminal access to which some or all incoming calls to the PUM user will be presented. Several PUM users may register for incoming calls at the same terminal access simultaneously.

Use case 2: Specify Service Type

A different terminal access may be specified for each service type (e.g. voice, telefax).

Use case 3: Specify Profile

The PUM user will be able to determine the desired "service profile" attached to this new registration depending on the calling party's identity, call importance indication, "no answer" and "busy" conditions and other possible criteria [6].

Figure 15 shows how the three use cases for PUM incoming call registration can be presented graphically in a Use Case Diagram.

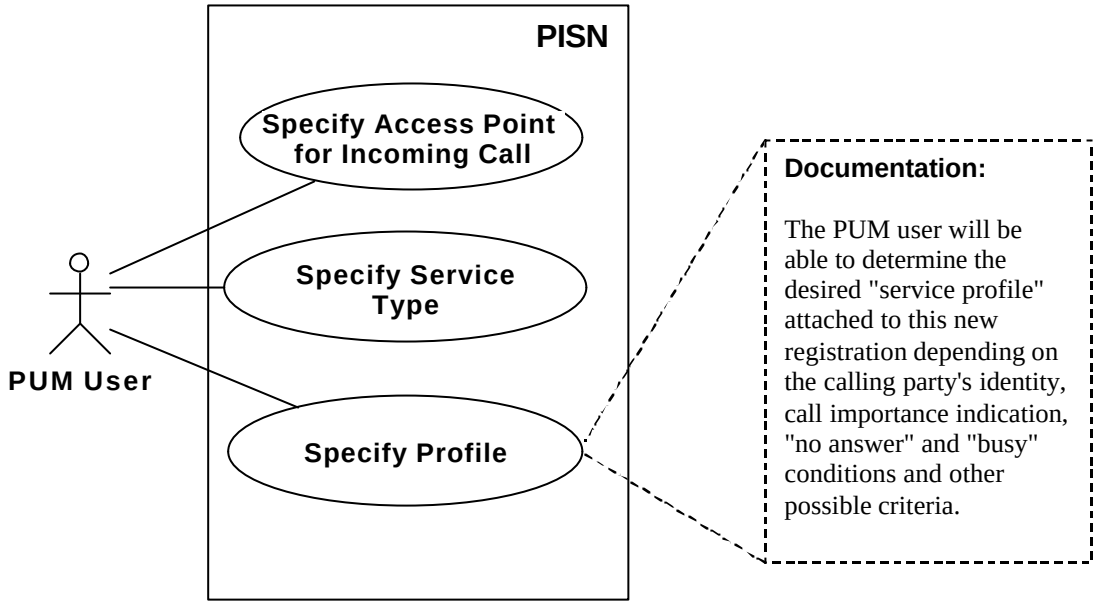

Figure 15: Example Use Case diagram for PUM Registration for Incoming Calls

NOTE: The box marked "Documentation" in Figure 15 is intended to show the text that might be included as part of the specification of the "Specify Profile" use case. It is included here for clarity and would not normally appear in a use case diagram.

4.5.3.5 Describing each use case

In most instances, it is difficult to construct a use case name which is both easy to read and comprehensive in its description of the function of the use case. It is, therefore, useful to produce a short description of each use case included in the Requirements Model. The following information should be used to describe each use case:

- name of the use case;
- brief description:
  - short overview of the purpose of the use case;
  - identities of the actors involved in the use case;
  - any preconditions;
  - step-by-step specification of what the use case needs to do when interacting with its actors

    this can be plain text but, if it extends beyond 2 or 3 steps, an activity diagram could be used to provide additional clarification;
  - any postconditions.

4.5.3.5.1 Activity Diagrams

When describing communication protocols it is often not possible to describe the functions of a use case in very simple terms. When the description of a use case cannot be expressed simply in a few lines of text, a UML activity diagram can be used as well. In particular, an activity diagram should be used if:

- the use case represents functions which are complex;

- there are conditional branches implied in the function of the use case.

Care should be taken to avoid the inclusion of too much detail in an activity diagram. The purpose of the Requirements Model is to define the requirements that are to be met by the standardized protocol, not to describe the detailed behaviour of the constituent entities.

4.5.3.5.2 PUMR Example

A simple, tabular description of the "Specify Profile" use case is shown in Table 2.

Table 2: "Specify Profile" use case description

Name	Specify Profile
Description	The PUM user specifies the desired service profile to be associated with the new registration
Preconditions	The PUM user is registered at the Visitor PINX.
Processing	See activity diagram
Postconditions	Service profile established for the PUM user

Figure 16 shows how the use case for processing a Service Profile setup request from a PUM user (Figure 15) could be described in an activity diagram.

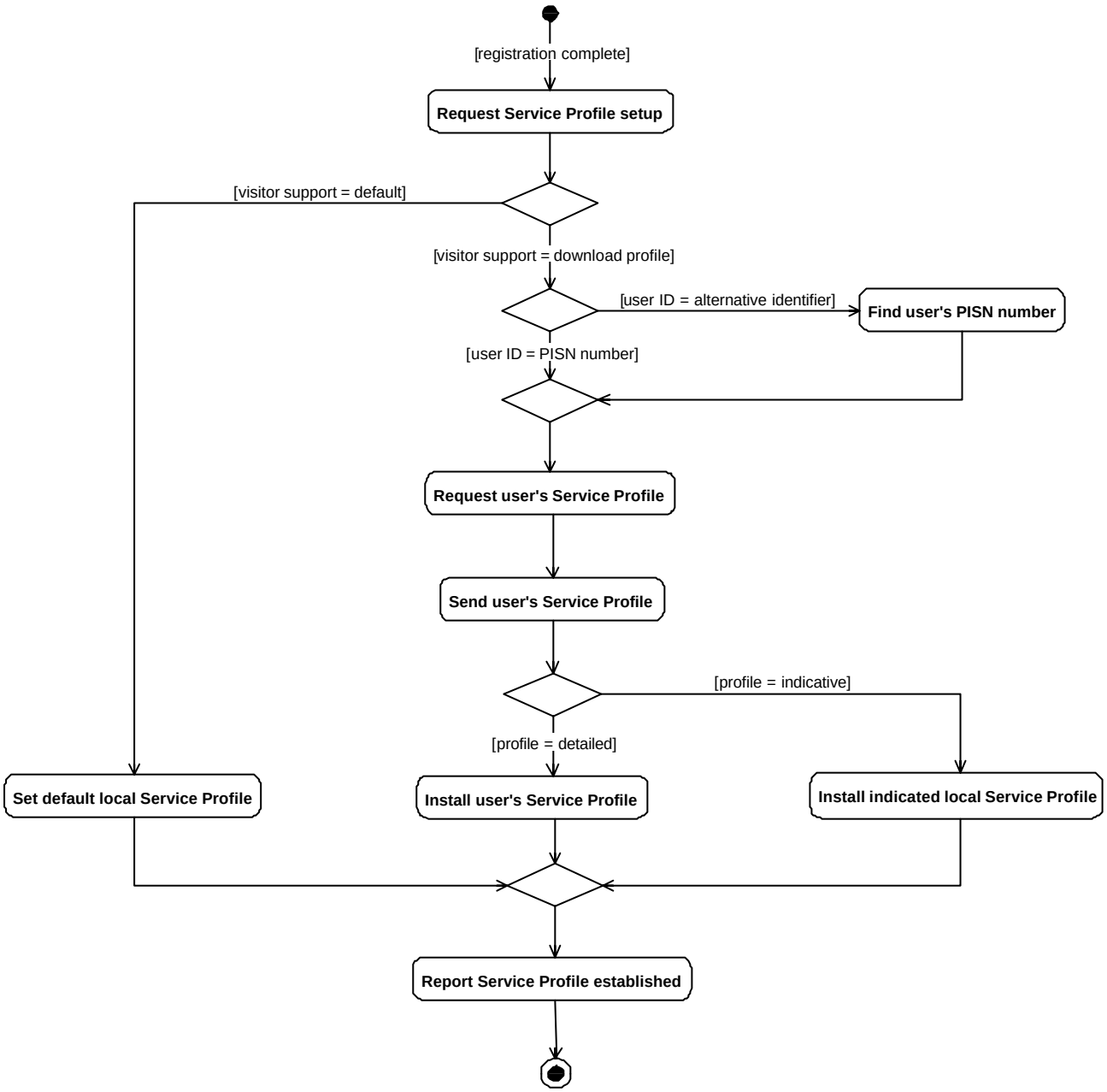

Figure 16: Activity diagram showing an overview of the Specify Profile use case

The overview of the Specify Profile use case in Figure 16 shows that a request from a PUM user to set up a Service Profile will cause one of the following:

- a default Service Profile provided by the Visitor Location will be established if the Visitor Location does not support the downloading of Service Profile information from the Home Location;
- a Service Profile provided by the Visitor Location will be established if the Home Location provides a simple indication of the PUM user's service classification (for example, "select Service Profile No. 5");
- a Service Profile provided by the Home Location will be transferred to the Visitor Location and established for the PUM user.

If the user provides an alternative identifier rather than a PISN (directory) number as identification, this will be resolved into a PISN number before any request is made to the Home Location.

Although Figure 16 identifies the activities that must occur as part of the Specify Profile use case, it does not indicate where in a network each activity should take place. Simple visual analysis is usually sufficient at this stage to determine where the responsibility for each action is likely to lie. For example, it is clear that the sending of the user's Service

Profile information will almost certainly take place at the Home location. UML swimlanes can be used effectively to highlight these divisions of responsibility as shown in Figure 17.

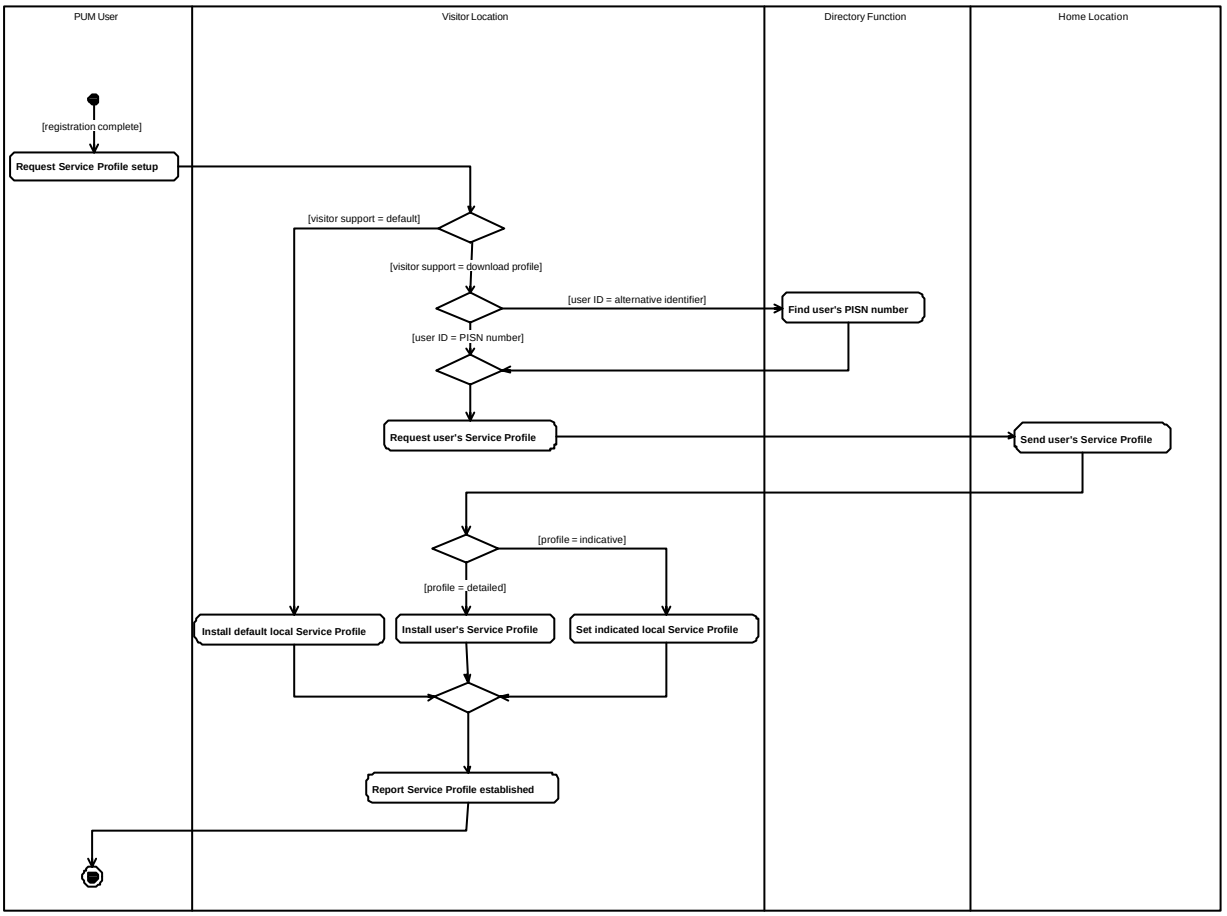

Figure 17: PUM Specify Profile activity diagram using swimlanes

When partitioning an activity diagram with swimlanes it can be tempting to start adding more detail to the activity itself. However, at the Requirements Modelling stage, activity diagrams should show only tasks and conditions. Even the addition of swimlanes should be limited to those instances where their placement is obvious with only the minimum of analysis.

4.5.4 Collect non-functional requirements

The UML is not an appropriate language for expressing requirements which are not action-based. Well structured text and tables should be used for this purpose.

4.6 Develop a Specification Model

4.6.1 Activity overview

A Specification Model is developed from the Domain Model by:

- refining the model of communication entities;
- adding communication interfaces;
- adding new entities if necessary;
- specifying the communication between entities;

The Domain Model and the Requirements Model should be used together as the base from which the Specification Model is developed. The Requirements Model provides guidance on how more detailed class diagrams can be developed from the Domain Model. It is at this point in the process that the flow of information across the interfaces which connect the communication entities should be considered. Initially, the functional messages (for example, SETUP and RELEASE) of the protocol should be identified and the temporal relationships between them specified using sequence and collaboration diagrams. In those cases where it is necessary to describe complex behaviour, it may also be useful to develop some statechart diagrams. A Specification Model should only express the relationships between classes and describe sequences of actions, but it should not specify how the communication mechanisms are to be realised. Developing a Specification Model may highlight inadequacies and inaccuracies in the Domain Model which should be reviewed and revised as necessary.

The activity diagram in Figure 18 shows, in simplified terms, the process involved in the development of a Specification Model.

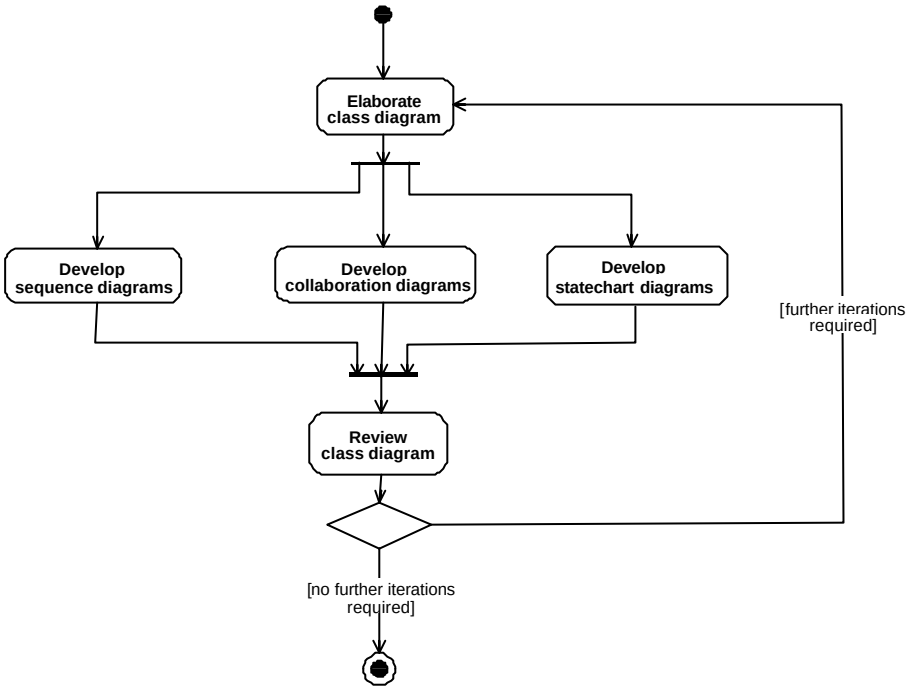

Figure 18: Specification Model development

4.6.2 Artefacts

The elaboration of a Specification Model is an iterative process involving a number of complementary diagrams as shown in Figure 19.

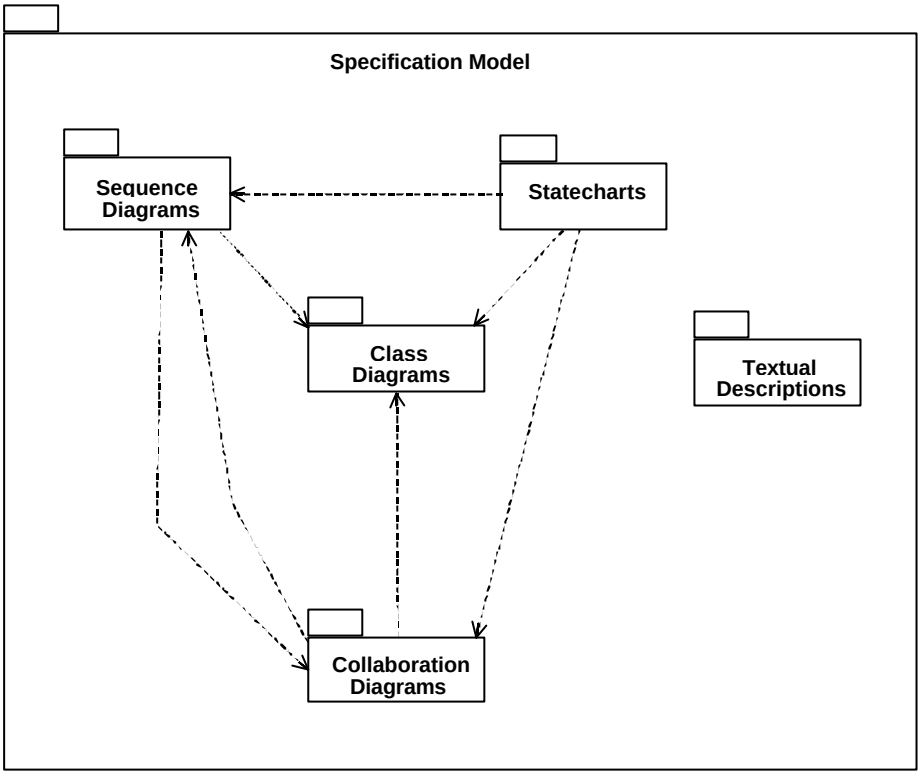

Figure 19: Artefacts produced as part of a Specification Model

4.6.3 Refining the model of communicating entities

A class diagram based on the generic class diagram for communication systems shown in Figure 7 is ideal for specifying a system in terms of its communication entities but it does not identify the interfaces necessary for these entities to communicate. The generic Specification Model shown in Figure 20 extends the Domain Model by adding communication interfaces which can be either normative or non-normative. It also introduces a "communication message" class which should be used for specifying the protocol signals which are to be passed across the interfaces.

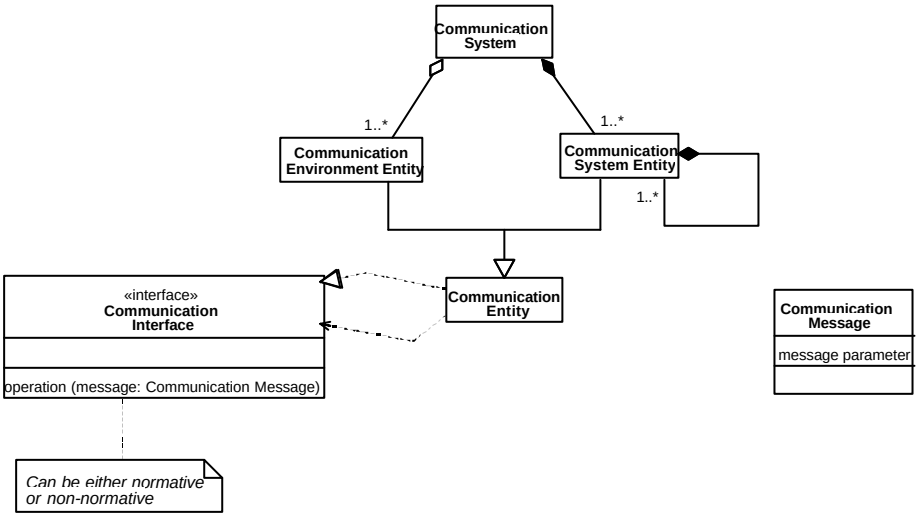

Figure 20: Generic Specification Model (class diagram) for communication systems

4.6.3.1 Class diagrams

4.6.3.1.1 Identifying candidate classes

Although a number of classes will have been identified during the Domain Modelling stage, it is certain that these classes will require the addition of further detail and that new classes will need to be specified.

Within a specification model, use cases are realized by classes and their derived objects which communicate and co-operate together to perform the necessary functions.

Three types of classes are identified within the generic Domain Model and should also be used when specifying classes for the Specification Model. These class types, denoted by stereotypes, are:

- communication entities:
  - communication system entities;
  - communication environment entities;
- communication interfaces;
- communication messages.

The following approach, shown graphically in Figure 21, should be used in refining these classes:

- review the use cases and the data flows to determine what new communication entities should be added to those already specified in the Domain Model;
- determine what, if any, new interfaces are required between the communication entities;
- add operations to each of the interfaces to handle the protocol messages that are necessary to support the use cases described in the Requirements Model;
- specify new message classes for each of the messages identified at each interface;
- add attributes to each of the message classes to indicate what information the message should carry.

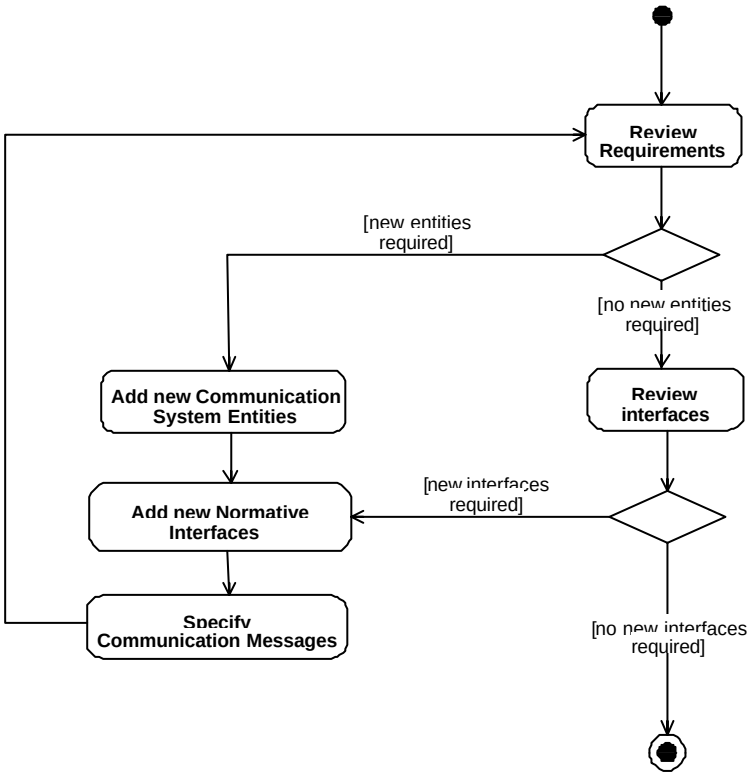

Figure 21: Class diagram elaboration

Those interfaces which are the subject of the standard should be clearly identified by attaching a text box to the interface class indicating that it is normative, as shown in Figure 22.

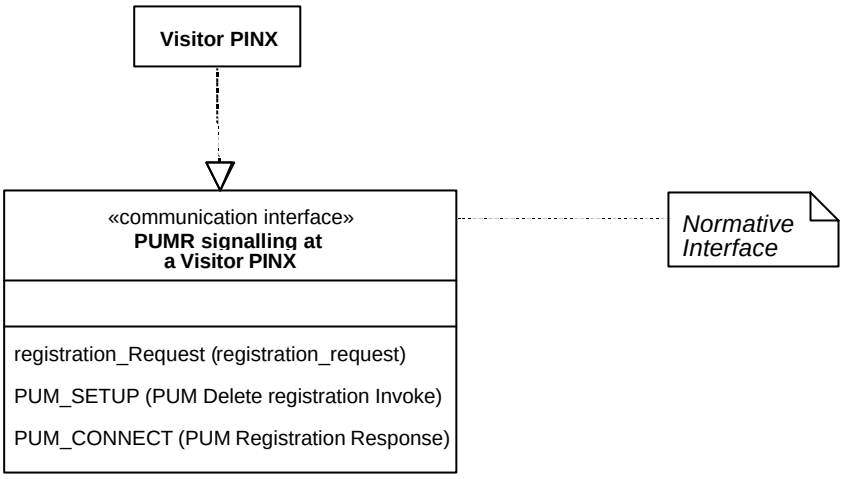

Figure 22: Example of an interface identified as "Normative"

4.6.3.1.1.1 Operations

Class operations should be used to identify which signals can be legitimately processed by a particular communications interface. It is conventional within the UML to indicate only those messages that can be received by a class. Those that may be sent are implied by the signals that can be received by adjacent classes.

The following example illustrates how this approach can be used to specify the signalling at a telecommunication interface. A normative (or non-normative) interface is usually considered to be a notional point in the communication path between two entities implementing the interface. Each of these entities will support the transmission and reception of a group of signals which together form the protocol at the interface. Figure 23 uses a traditional reference diagram to illustrate the implementation of an interface at an imaginary reference point "X". It also shows the protocol messages that can be exchanged between the Terminal and the Network Access.

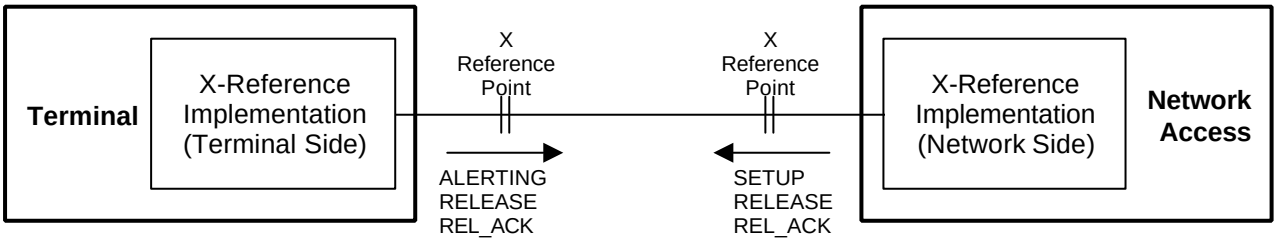

Figure 23: Representation of an imaginary communication system

Figure 24 shows how the UML can be used to represent this type of system as two communication interfaces each of which is realized by one of the communication entities. It uses a class diagram to show how the Terminal side interface at the X Reference Point can receive SETUP, RELEASE and REL_ACK messages which are all of the generic type, "PDU" while the Network Access side interface can only receive ALERTING, RELEASE and REL_ACK.

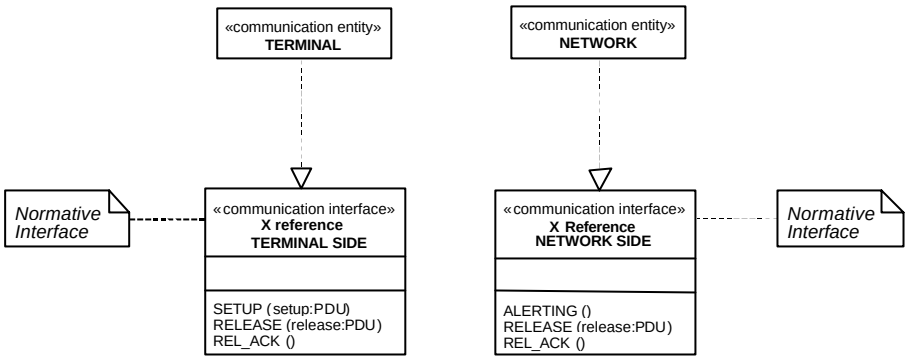

Figure 24: Class diagram showing signals as operations

4.6.3.1.1.2 PUMR Example

The very simple class diagram in Figure 10 shows the PUMR system comprising little more than a generic PINX which realizes three separate interfaces. At the specification modelling stage, this model, shown in Figure 25, has been developed further to give considerably more detail about the messages that can be sent across the interfaces. For each of the distinct functions (Home, Visitor and Directory) within the PUMR system, a class has been specialized from the general PINX class. In addition, the communication interface classes have been tagged as either "Normative" or "Non-normative".

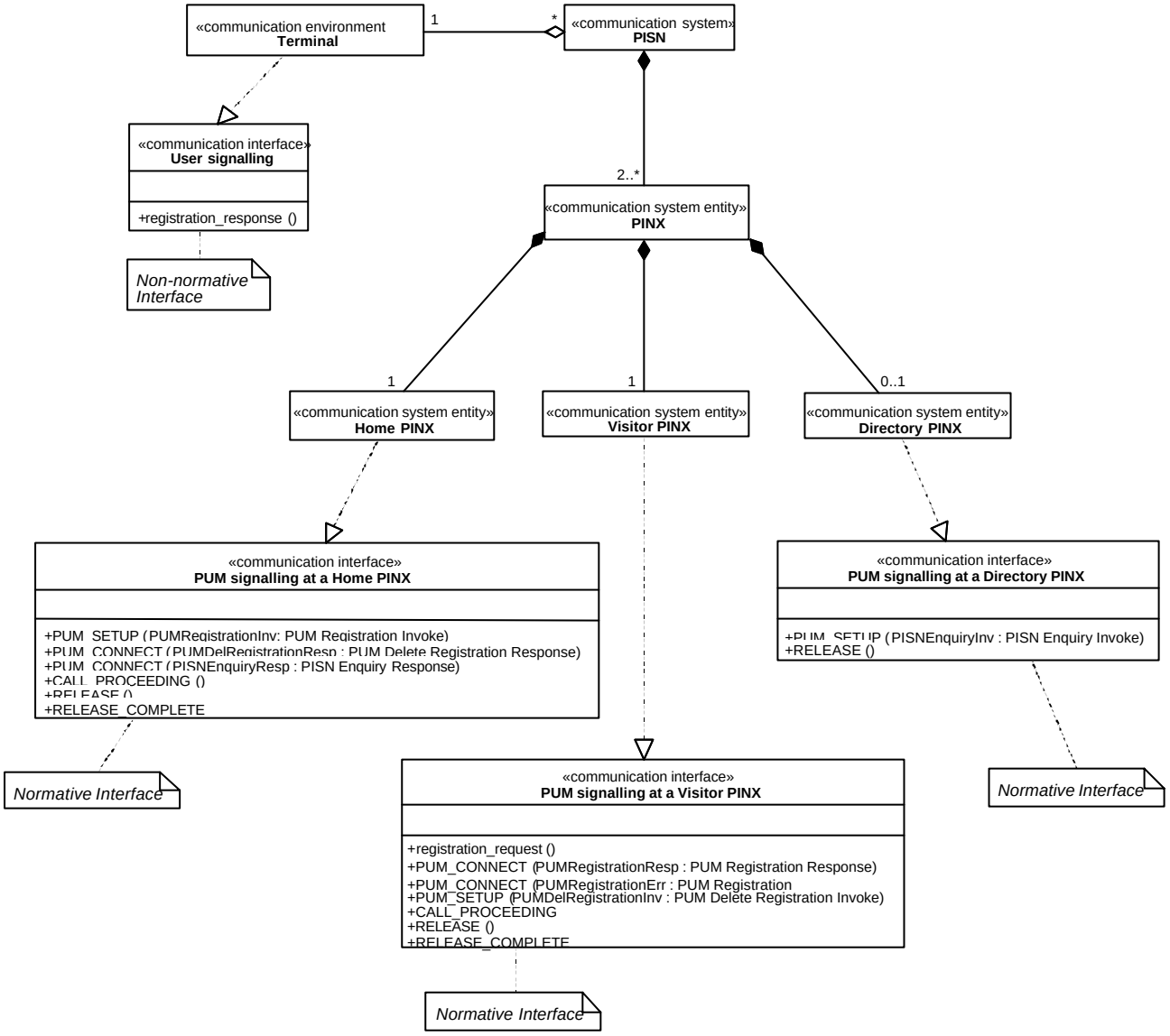

Figure 25: System-level specification model class diagram for PUM Registration

4.6.3.1.1.3 Attributes

Class attributes can be used to describe the contents of protocol messages where these messages are described as UML classes. The attributes should clearly identify which items of information are included in a particular message but should not attempt to describe the detailed format that they will take as this is better achieved using ASN.1. However, it is possible to describe basic data structures using UML.

Figure 26 shows how a SETUP message will contain an originating address and a destination address, which are both network addresses, and a service identifier which can take any of the values allowed for a basic service. It also shows how a Network Address has an address portion and a sub-address, each of which is a string of up to 26 dialled digits and that there are a range of enumerated values possible for the Network Basic Service argument.

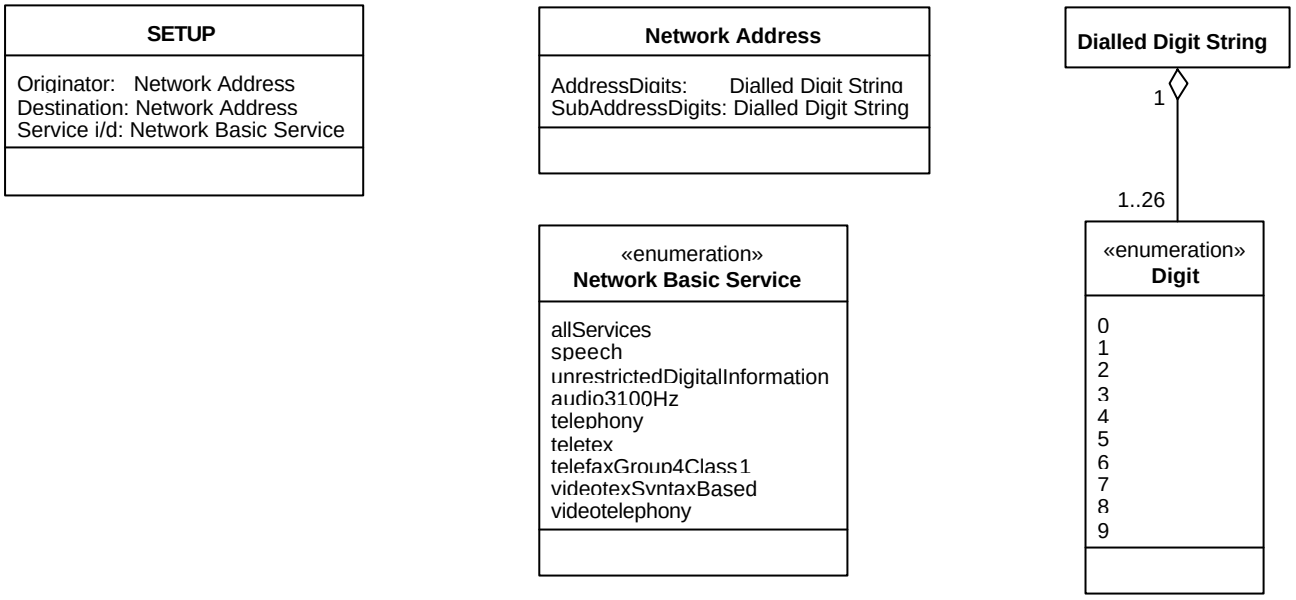

Figure 26: Example of message classes using attributes

4.6.3.1.2 Further iterations of the model

The classes specified in the earlier stages of modelling should be refined or amended through subsequent iterations so it is important to maintain a log of the changes made to each class and in which use case realisations it participates.

A class should depict only one major objective and should be given a name which clearly identifies that objective using the vocabulary of the domain. Additional documentation should be added to each class to ensure that its purpose is made clear and unambiguous.

In order to avoid unnecessary complexity in the Specification Model, it is useful to review each of the possible classes identified, considering the following points:

- if the specification is similar to another class then it may be possible to combine the classes;
- if the specification of the class cannot be expressed in a few lines then it is probably too complex and should be sub-divided;
- if neither a clear name nor a concise specification can be devised then it is probably that the class is not valid and further analysis is required;
- if it is difficult to decide how a use case can be realized then it is possible that there are further classes to be defined.

4.6.3.2 Sequence diagrams

Sequence diagrams are used for modelling the relationship in time of the messages which are exchanged within a system. They show how the “responsibilities” specified in use cases are assigned to the different objects and classes of the system. The operations of objects are used to identify the messages which can flow between objects.

Figure 27 shows a very high level sequence diagram for the PUM Registration service. It illustrates the simple requirement that a PUM user should be able to send a registration request to the PISN which will return a registration response after processing the request and registering the user at the new location.

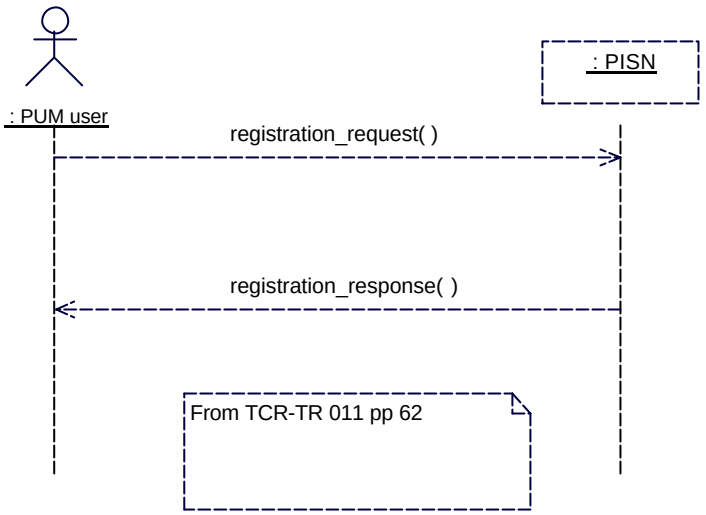

Figure 27: High level sequence diagram for PUM user registration

Each sequence diagram should include one or more of the participating actors and the system objects between which messages are exchanged. The normal message flow should be described first and, if there are complicated exceptions, these should be shown in separate diagrams. Constraints can be used to highlight the differences between normal and exceptional flows of messages. Figure 28 represents a refinement of the PUM registration scenario, considering the different objects in the system, i.e., Visitor, Home and Previous PINXs.

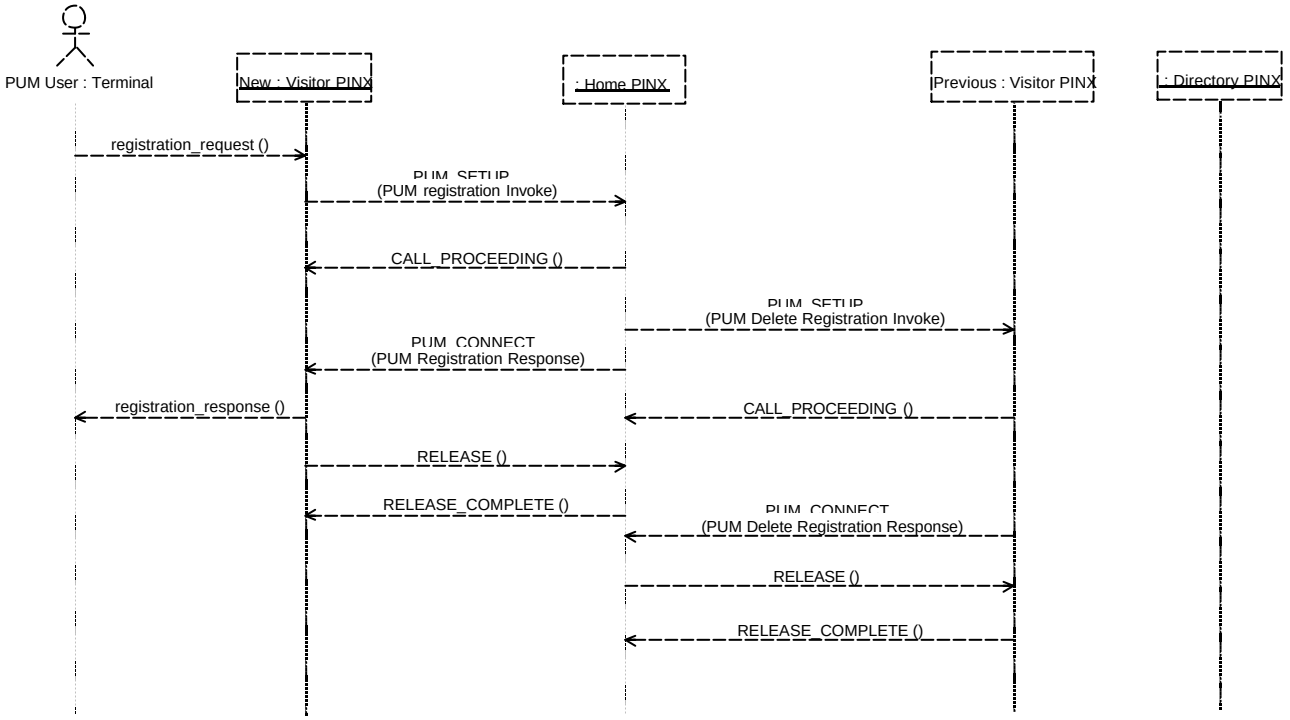

Figure 28: Complete sequence diagram for successful PUM user registration

Sequence diagrams are used in UML to describe the detailed interaction between objects and, as such, are similar to basic Message Sequence Charts (MSCs). However, it is not possible to include parts of another sequence diagram or to structure complex ones similar to HMSCs. Hence sequence diagrams should not include too much detail if they are to remain understandable.

4.6.3.3 Collaboration diagrams

Collaboration diagrams depict a set of objects in a given situation. Links between objects that can interact together show the messages that can be exchanged. These are numbered in sequence to specify the time order in which they occur and can have arguments in the form of a parameter list.

Figure 29 shows the collaboration diagram which represents the same message scenario depicted in the sequence diagram in Figure 28.

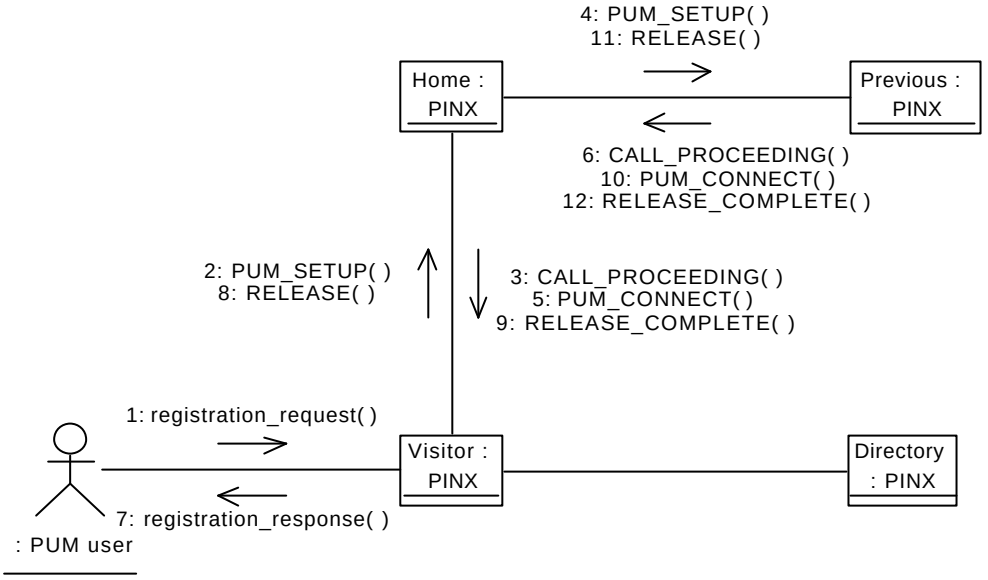

Figure 29: An example UML collaboration diagram for PUMR

Collaboration diagrams are especially useful at the specification modelling stage when determining what objects are required in the system and specifying the meaning of their interactions. These objects can be either named or unnamed instances of classes.

Collaboration diagrams and sequence diagrams are different views of the same information. The difference between them is that sequence diagrams focus on the relationship in time of the messages that flow between objects whereas collaboration diagrams focus on the relationships between the objects themselves.

4.6.3.4 Statechart diagrams

Although class diagrams are very useful for showing the structure of a protocol system, its dynamic behaviour can only be represented through communication interfaces (operations in classes), functional requirements (use cases), and object interaction examples (sequence diagrams). By using statechart diagrams, it is possible to describe the individual behaviour of a given object of a particular class in terms of state changes caused by events. This state behaviour should correspond to the interpretation of the messages received by the object. Once an object for which it would be helpful to have a more detailed description of behaviour has been selected from a sequence diagram, the use cases related to its class should be studied to determine what behaviour is to be modelled. Then, any relevant sequence diagrams and collaboration diagrams should be studied to ensure that all messages sent and received by the class are included in the statechart diagram as actions or events.

When receiving an event, the statechart initiates the sole transition that is enabled by it, causing an action and a state change. Actions are the operations specified in the class of the object that receives the event. For example, an action may be to send a signal to another object. It is acceptable to have events and actions associated only with state transitions and not with the states themselves. The main reason why this is acceptable is that in the standards making process, SDL is used for detailed behaviour in the next stages and so such basic statechart diagrams are sufficient at the specification stage.

In the Figure 30, the notation "*EVENT [GUARD] / ACTION*" is used to label the transitions. Control is transferred from state "Idle" to state "Processing" or state "Relocating" depending on the value of the "Location" argument.

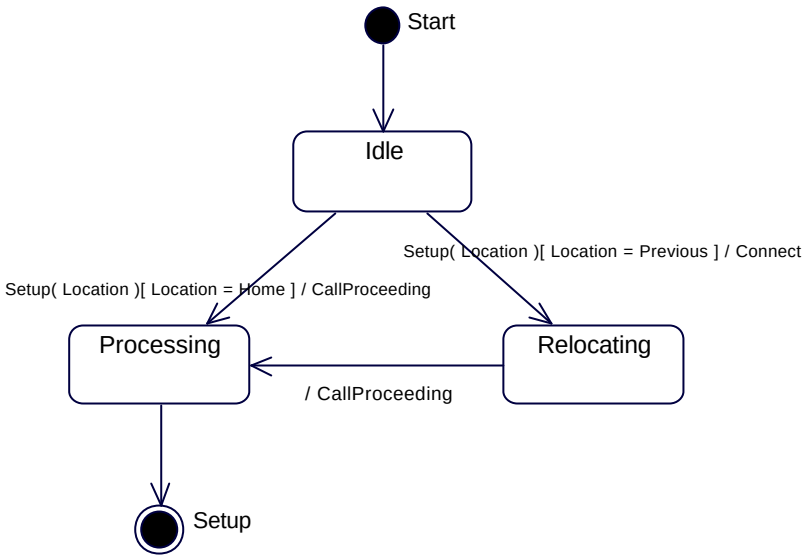

Figure 30: A simple statechart diagram derived from the PUM study

It must be noted that both guards and actions are only textual and are used for descriptive or referencing purpose only. Even though they are not executable, they should be expressed in a structured and meaningful form.

All interface objects must have a set of message-receiving operations showing all possible incoming events. Statechart diagrams should be used if:

- an interface object has a large set of operations;
- an interface object has operations representing behaviour which is complex;
- there are conditional branches in the internal behaviour of the object; and/or
- at a given time, only a subset of operations of an interface are feasible.

The statechart diagram in Figure 31 shows an overview of the operation of PUM user registration at the Home PINX. The caret (^) character preceding each action indicates the sending of a message (e.g., "^CALL_PROCEEDING") during a transition from one state to another. This example is consistent with the sequence diagram shown in Figure 28.

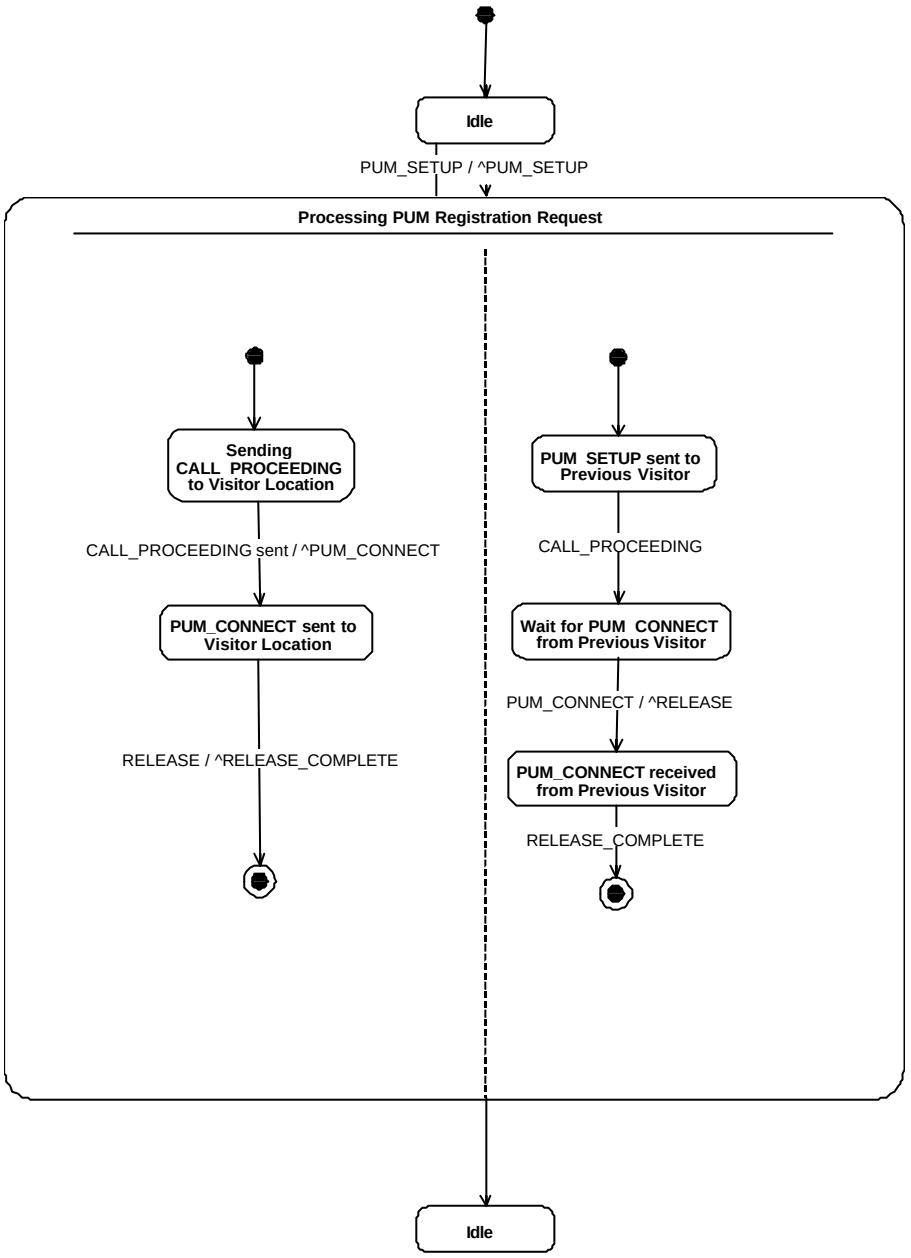

Figure 31: Statechart diagram showing PUM user registration at Home PINX

The statechart diagram shows how the Home PINX begins in the Idle state and always returns to it at the end of processing a registration request. The dual path through the chart indicates that CALL_PROCEEDING is be processed at the same time as the PUM_SETUP to the Previous Visitor PINX.

Generally, a statechart diagram is attached to a class in order to describe the behaviour of its instances, specifying the events to which they must react and how they should react. If the behaviour of a use case is already defined using activity diagrams then the corresponding statechart diagrams should refine this behaviour in order to be consistent. Statechart diagrams should be used in the latter stages of the process to specify details of the behaviour.

4.7 Use SDL and MSC to specify detailed behaviour

Although the UML can support the specification of detailed behaviour using existing industry-standard text-based languages such as C++ and Java, it currently has no graphical action semantics of its own. This means that in using the UML there is a reliance on an implementation language to get an executable specification. SDL, however, is able to produce executable models which are independent of any implementation. Therefore, protocol standards are generally described using SDL and MSC, and this should continue at least until a viable alternative is available within the UML.

The general approach presented in this EG is that UML is used to identify, analyse and specify the system entities, together with their relationships and then SDL and MSC are used for architectural and detailed behaviour design. The joint use of the three notations requires a smooth transition from analysis (Specification Model) to design. This is specified in ITU-T Recommendation Z.109 [7] that defines a set of mapping rules between UML and SDL constructs, known as the SDL UML profile. This mapping between the diagrams of the two notations can be realised by introducing a number of stereotypes in the UML classes.

The UML development process defined here adopts a similar approach. For the purpose of protocol standardisation, there are five new stereotypes introduced here and the mapping to SDL concepts shown in Table 3 is suggested.

Table 3: Mapping between UML stereotypes and SDL concepts

Stereotype in UML	SDL concept
<<communication system>>	System
<<communication system entity>>	Block
<<communication environment entity>>	SDL environment
<<communication interface>>	Signal list
<<communication message>>	Newtype or ASN.1 type

In addition, the following guidelines should also be considered

- links between the UML objects should be converted to channels;
- associations between <<communication interface>> and <<communication system entity>> or <<communication environment entity>> join signal lists to channels in the SDL model;
- every block converted from a <<communication system entity>> should contain a process whose behaviour is defined by the associated UML statechart diagram.

Use case diagrams are not directly mapped to SDL (they often are informal) but they are realised by classes that are converted to an SDL structure diagram. UML statechart diagrams can be directly converted to SDL state machine diagrams with a few adaptations for hierarchical UML states. Finally, sequence diagrams can also be directly mapped to MSC on a one-to-one basis, as they are a subset of the MSC notation.

There is no single point in the UML development process where the transition to an SDL specification can easily be made. The graphical similarity between UML sequence diagrams and MSC make them an obvious point at which to move from one language to the other, particularly in those cases where the target standard is to contain a complete SDL model. Certainly, the UML-based process described here should be used up to this point. However, there are benefits to be gained by continuing beyond sequence diagrams and into the specification of statecharts.

Well-defined statechart diagrams, combined with a collection of sequence diagrams form a solid base from which to develop an SDL model that conforms to the requirements specified in UML. The activity diagram shown in Figure 32 illustrates a general process that can be followed in making the transition from a UML specification of a system to an SDL specification of its behaviour. It identifies the actions to be taken and the UML artefacts which are the inputs to those actions.

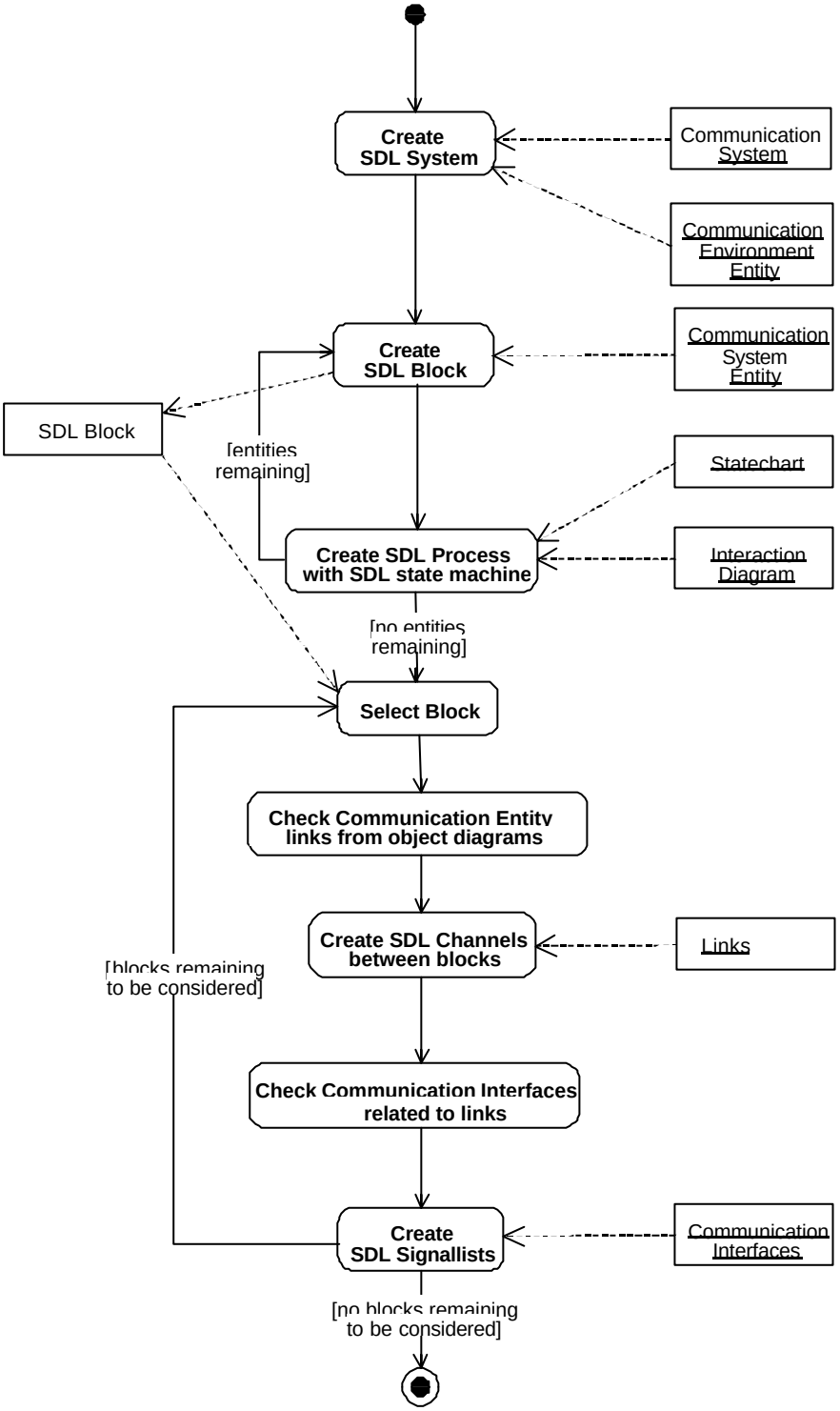

Figure 32: Transition from UML to SDL

4.8 Use the UML to support test development

The development of a conformance test suite is an activity which can take place in parallel with the development of the base standard. It should be a goal to reuse as much of the standardization specification as possible for test suite development. This clause provides guidelines on how the UML could be used to support the development of conformance test suites based on the ISO/IEC 9646 standard [9]. The Second Edition of the Tree and Tabular Combined Notation (TTCN) is considered to be the target test notation.

4.8.1 Activity overview

The development of a test model can be divided into several distinct activities as shown in Figure 33. In the first step, independent system components have to be identified. In the next step, test configurations are developed which describe the mapping of components on system and test nodes. In the third and fourth steps, test case structures and test purposes are defined for all test configurations.

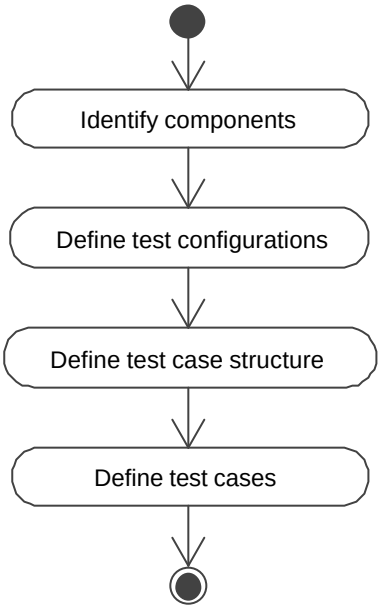

Figure 33: Activities during test model development

4.8.2 Artefacts

The following artefacts are produced as part of the Test Model (Figure 34):

- component and deployment diagrams for test configuration specification;
- class diagrams for test case structuring;
- sequence, collaboration and statechart diagrams for test purpose definitions.

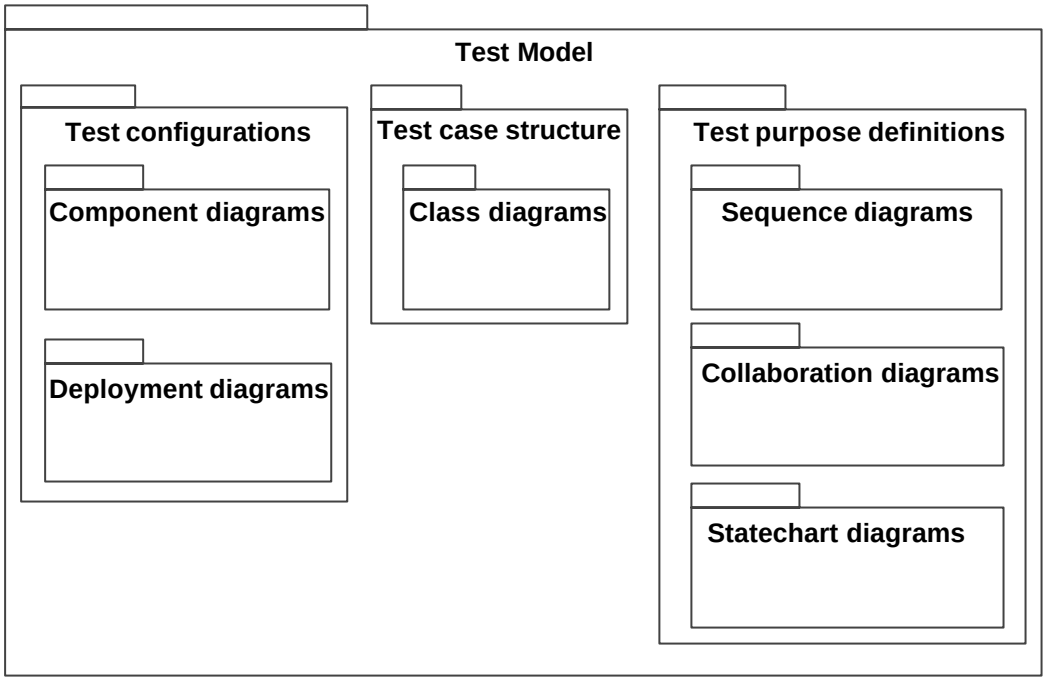

Figure 34: Artefacts produced as part of the Test Model

4.8.3 Identify components

The goal of the first activity during test modelling is to identify functional entities which can be tested independently. With the UML, these functional entities are depicted as components.

During conformance testing, only normative interfaces can be tested. Therefore, components must realise at least part of a normative interface. Normative interfaces have been identified during Context Modelling (see subclause 4.4.4.3), so the Context Model can be used as a reference point for component identification.

4.8.3.1 PUMR example

Using the information about normative interfaces in Figure 12, the following components have been identified for PUMR:

- PUMR Home;
- PUMR Visitor;
- PUMR Directory.

Figure 35 shows the PUMR components and the interfaces which they realise.

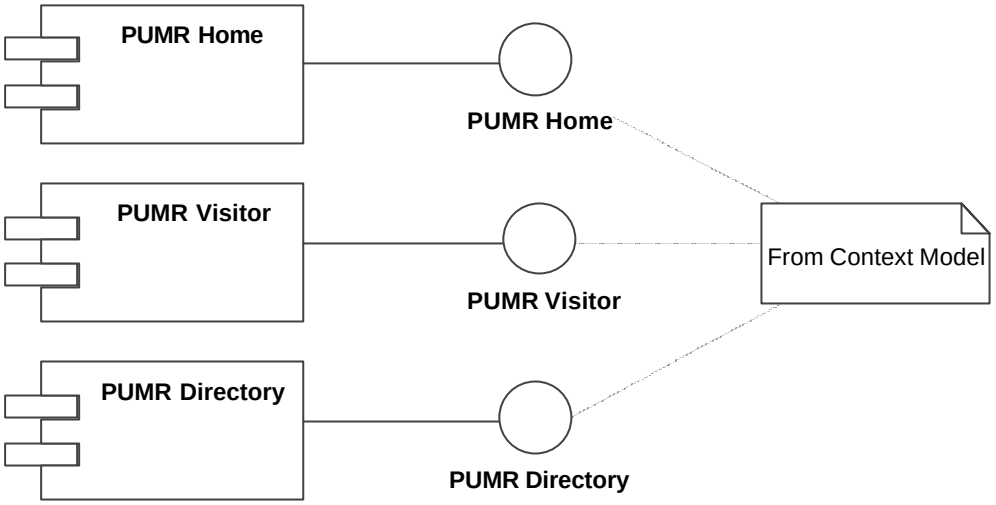

Figure 35: PUMR components

4.8.4 Define test configurations

After their implementation, the components (functional entities) defined in subclause 4.8.3 will be executed on some piece of hardware. There are connections between the components, either physical or logical.

During conformance testing, one or more implementation components are replaced with test components. Test components stimulate the Implementation Under Test (IUT) and then check the implementation's response for conformance with the standard.

In order to be able to specify a test suite, a test configuration has to be defined first. UML deployment diagrams can be used to identify the IUT, test components and their connection through Points of Control and Observation (PCO) and Coordination Points (CP). In Figure 36, the component FE1 resides in a node called "Implementation" which is stereotyped as IUT. Component FE2 has been moved from the IUT into the "Tester" node which is stereotyped as MTC (for Main Test Component) according to ISO/IEC 9646.

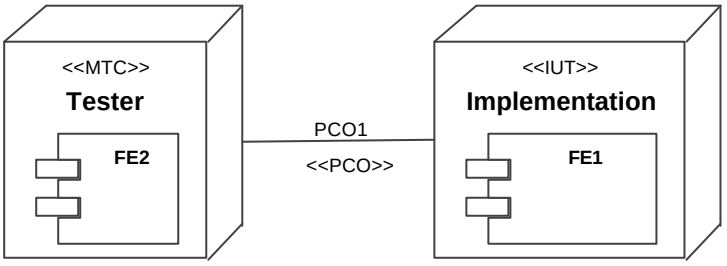

Figure 36: Generic test configuration

4.8.4.1 PUMR example

Figure 37 shows an example of a test configuration for a PUMR system. The IUT only contains the PUMR Home component, suggesting that this component is the target of the test suite. There are three Parallel Test Components (PTC) which are connected with the IUT through a PCO each. These test components act as Visitor PINX, Previous Visitor PINX and Directory PINX respectively. A main test component called "Test coordinator" is connected with the parallel test components through coordination points.

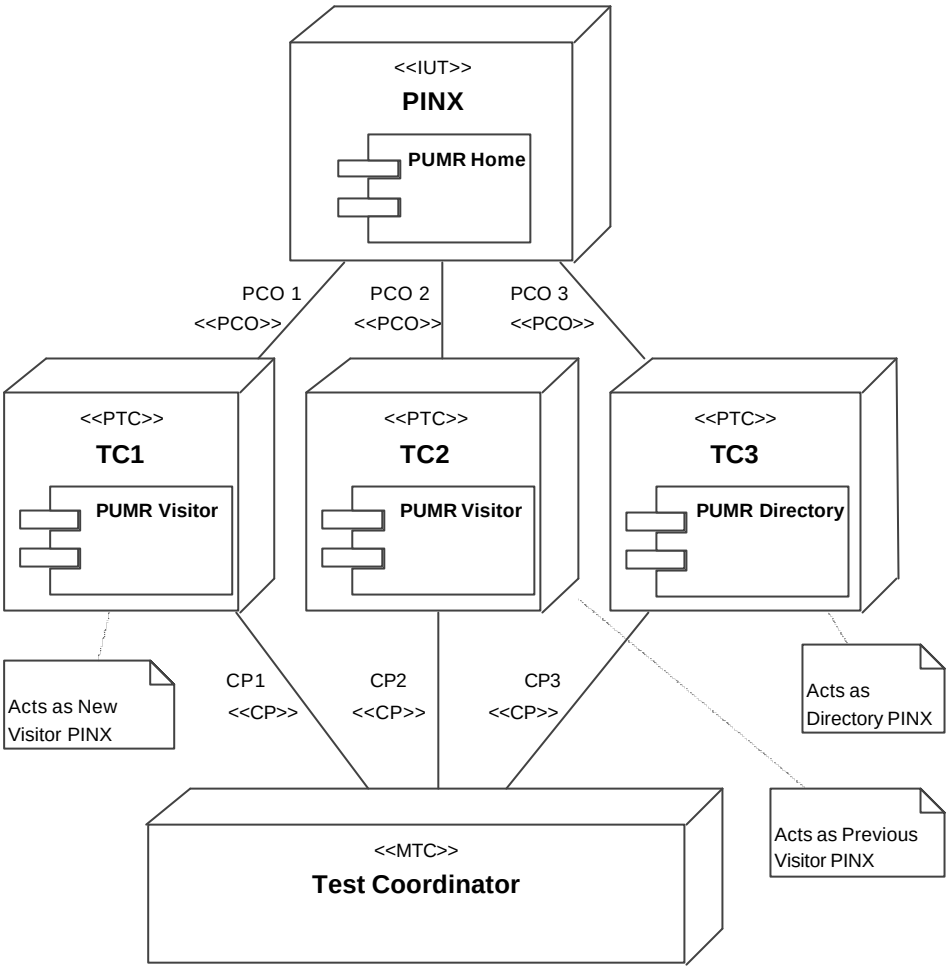

Figure 37: PUMR test configuration

4.8.5 Define test case structure

Test suites contain test cases which realise test purposes. It is common to put test cases with similar purposes into groups. Basic groups of test purposes have already been defined in ISO/IEC 9646 [9]. For example, there are Capability Tests, Valid Behaviour Tests and Timer Tests. Groups can be nested; the hierarchy of test groups is called Test Suite Structure.

Through the use of packages within class diagrams, a test suite structure can be defined graphically with the UML, as is shown in Figure 38. Test cases are also represented as packages. Of course, more than one diagram will be used to define the test case structure in real-world specifications.

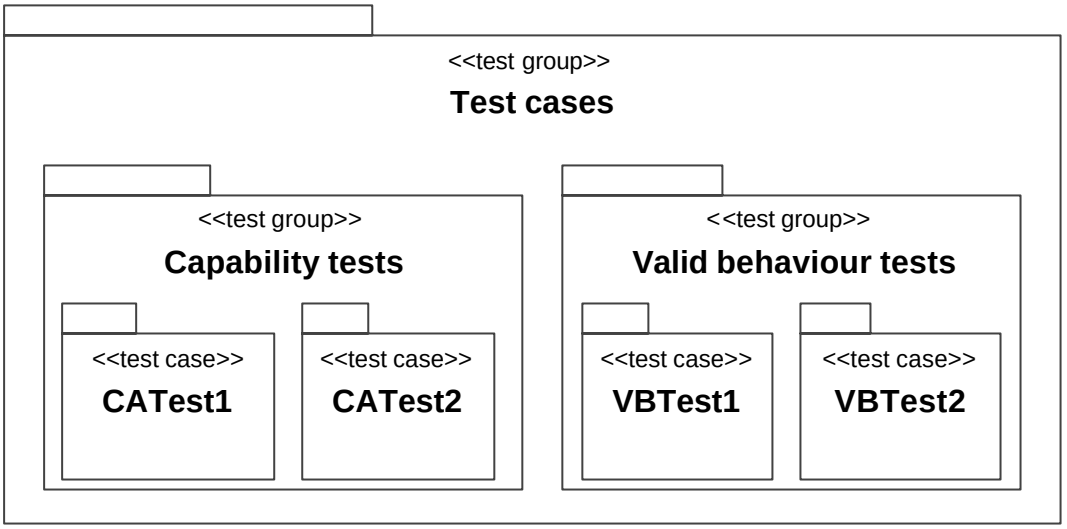

Figure 38: Test case structure

NOTE: The structure for test steps can be specified similarly to the test case structure.

4.8.6 Define test cases

Viewed at a conceptual level, a test case is the realisation of a test purpose. The "Test purpose style guide", ETR 266 [1], defines the information which has to be provided by the test designer in order to write TTCN test cases. This information is mostly textual, but Message Sequence Charts may also be included as a graphical representation of the test purpose.

NOTE: Message Sequence Charts can only express a subset of all possible TTCN behaviour descriptions.

The UML provides several diagram types which can be used to help the development of test cases. Sequence and collaboration diagrams can show the signal exchange between the tester and the IUT. These may be taken from the Specification Model and adapted for test specification purposes. As an alternative, statecharts may be used to model the functionality of individual test components; these can also be taken and adapted from the Specification Model.

4.8.6.1 PUMR example

In this example, a test case should be developed for use case 1 identified in subclause 4.5.3.4. The test purpose is to verify that a user can successfully register at a Visitor PINX and that he will be deregistered from his Previous Visitor PINX. The test configuration to be used is the one shown in Figure 37. In the Specification Model, a sequence diagram has been drawn which shows the message exchange necessary for the PUMR user registration (Figure 28). Figure 39 shows a version of this diagram which has been adapted to show the message exchange during test execution.

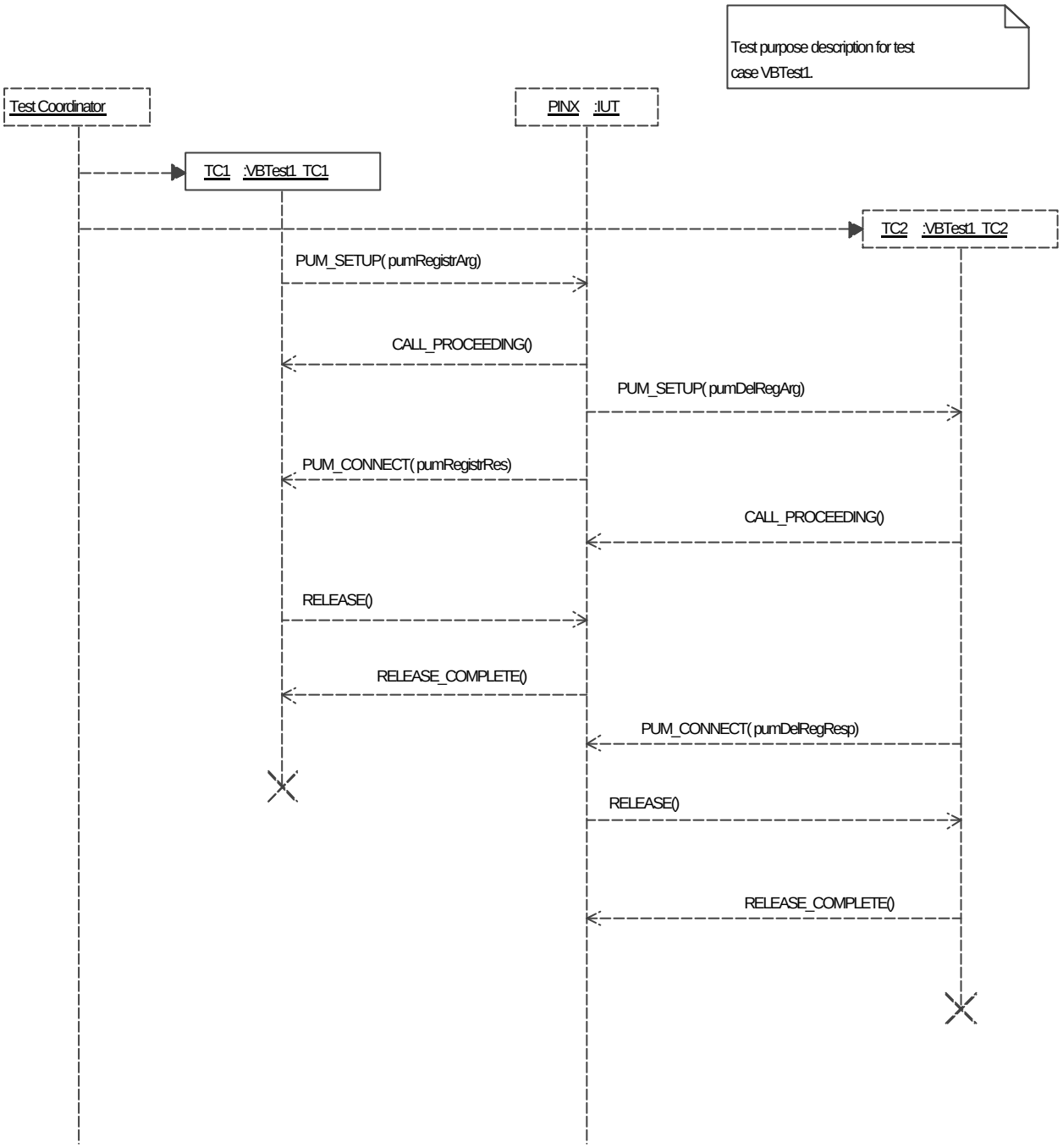

Figure 39: Sequence diagram for test purpose specification

Annex A (informative): Case Study

A.1 QSIG Private User Mobility Registration (PUMR) supplementary service

Private User Mobility Registration (PUMR) is a supplementary service that enables a Private User Mobility (PUM) user to register at, or de-register from, any wired or wireless terminal within the PISN. The ability to register enables the PUM user to maintain the provided services (including the ability to make and receive calls) at different access points. It was chosen to illustrate the UML guidelines for the following reasons:

- a pre-normative study highlighting the initial requirements for the service already existed in TCR-TR 011 [6];
- the stage 1 / stage 2 [8] and the stage 3 [10] standards are well expressed and include refined user requirements, ASN.1 specifications of operations, Message Sequence Charts and SDL process charts;
- the PUMR service is neither trivially simple nor prohibitively complex.

Contained in this annex are a Context Model, a Requirements Model, a Specification Model and a Test Model for PUMR. From these it would be possible to derive an SDL specification and a conformance test suite. The models may have elements missing but they are complete enough to show how the various UML concepts and diagrams can be used in the development of a protocol standard.

NOTE: The Context, Requirements and Specification Models are presented pictorially here but they are also available in electronic format as either HTML for browsing or as Rational Rose 2000 models for editing. The Testing Model is also available electronically as an XMI-compliant file.

A.2 PUMR UML models

A.2.1 Context Model

The PUMR Context Model is very simple and is just used to illustrate the basic concepts upon which the service is to be built.

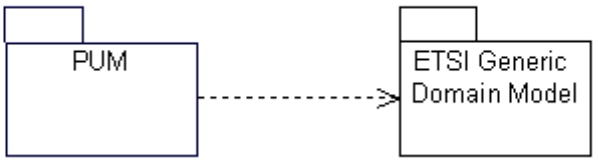

Figure 40: Context Model packages

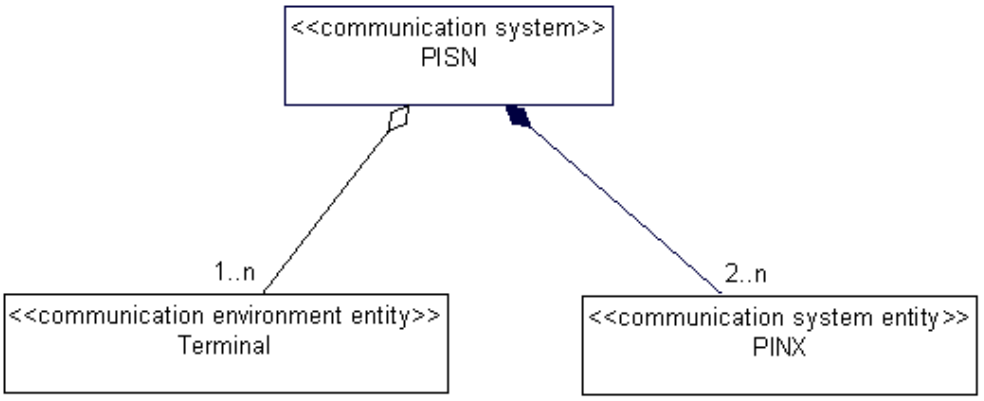

Figure 41: Simple PUMR Domain Model

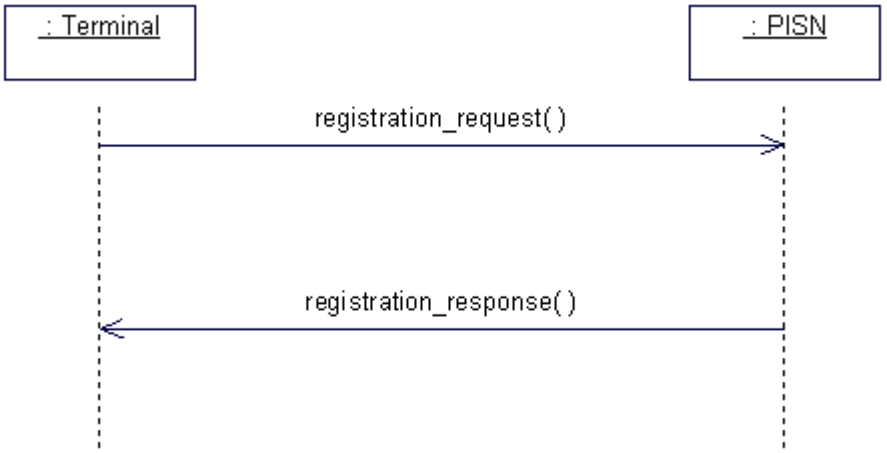

Figure 42: Sequence diagram indicating the flow of information between the user and the PISN

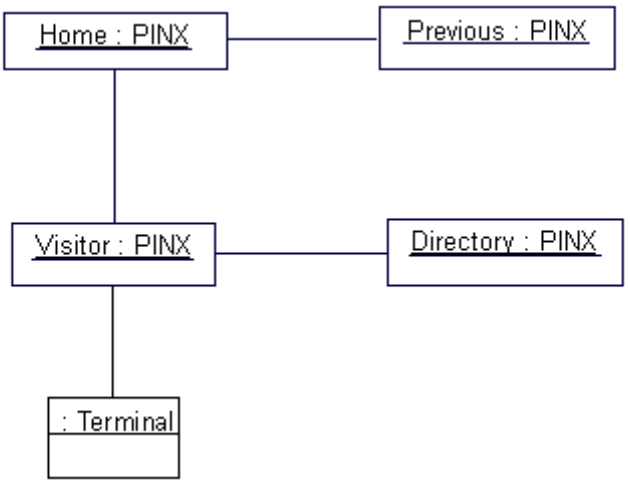

Figure 43: PUMR system architecture shown in an object diagram

A.2.2 Requirements Model

The use cases developed for the PUMR Requirements Model are based upon the requirements specified in TCR-TR 011 [6] and the stage1 / stage 2 standard, ISO/IEC 17875 [8] where the original requirements have been refined.

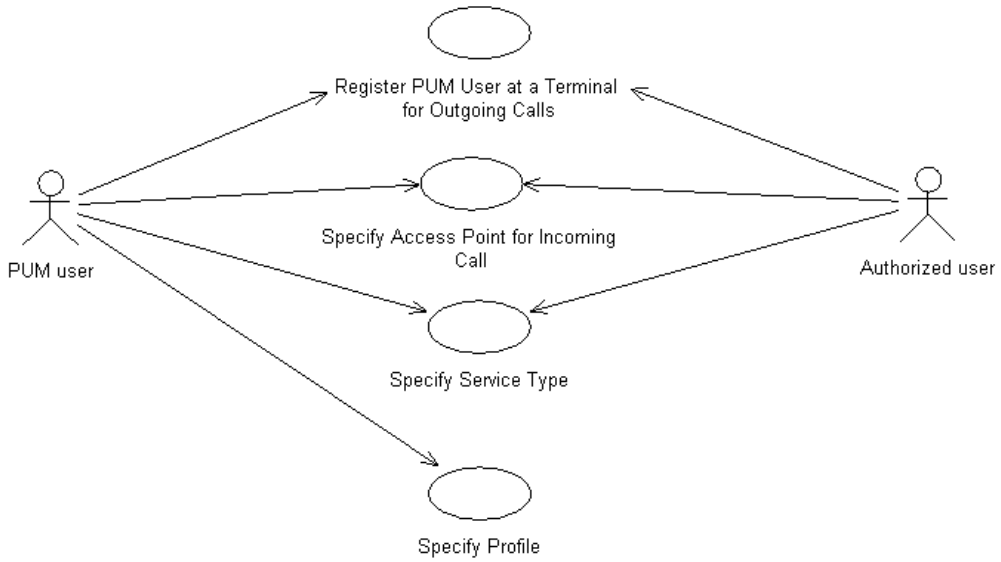

Figure 44: PUM Registration use case diagram

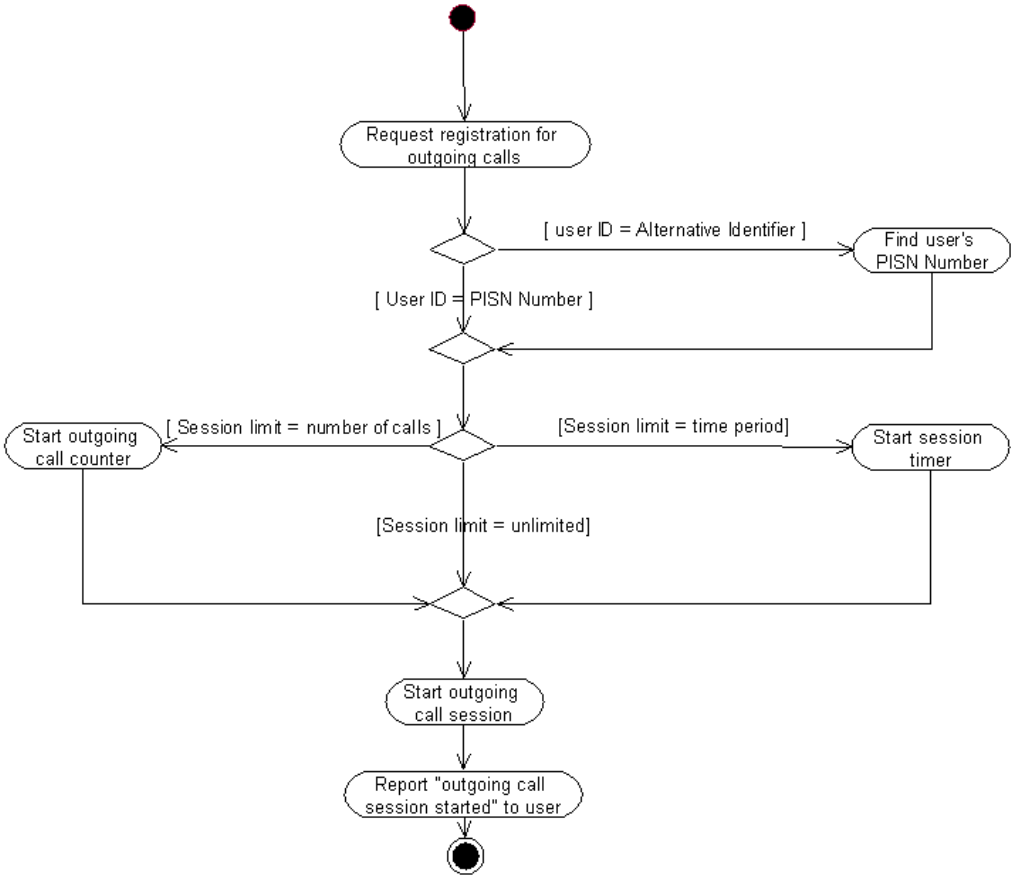

Figure 45: Activity diagram describing the "Register PUM User at Terminal for Outgoing Calls" use case

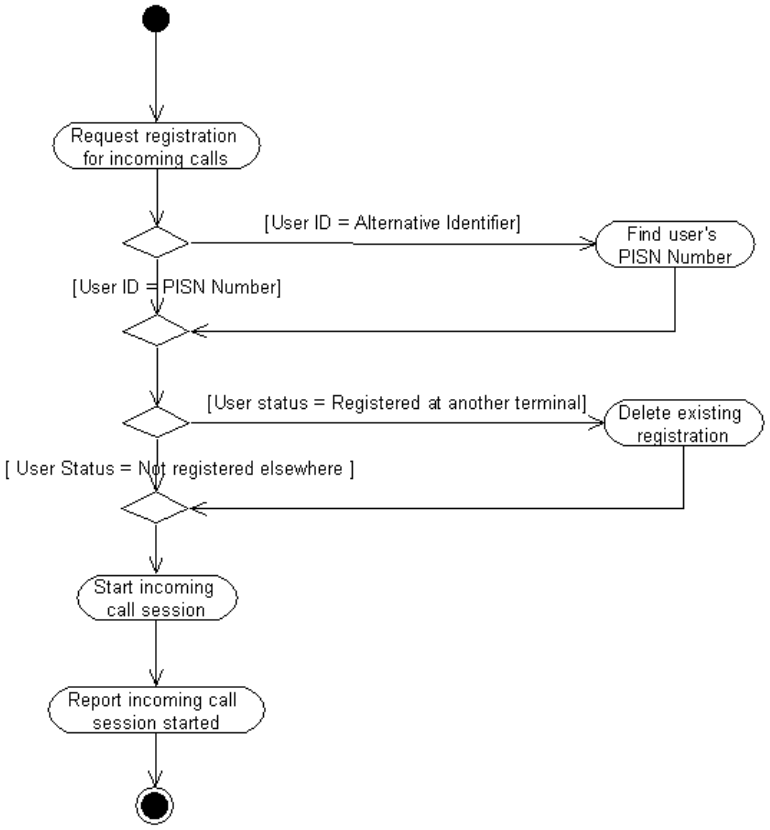

Figure 46: Activity diagram describing the "Specify Access for Incoming Calls" use case

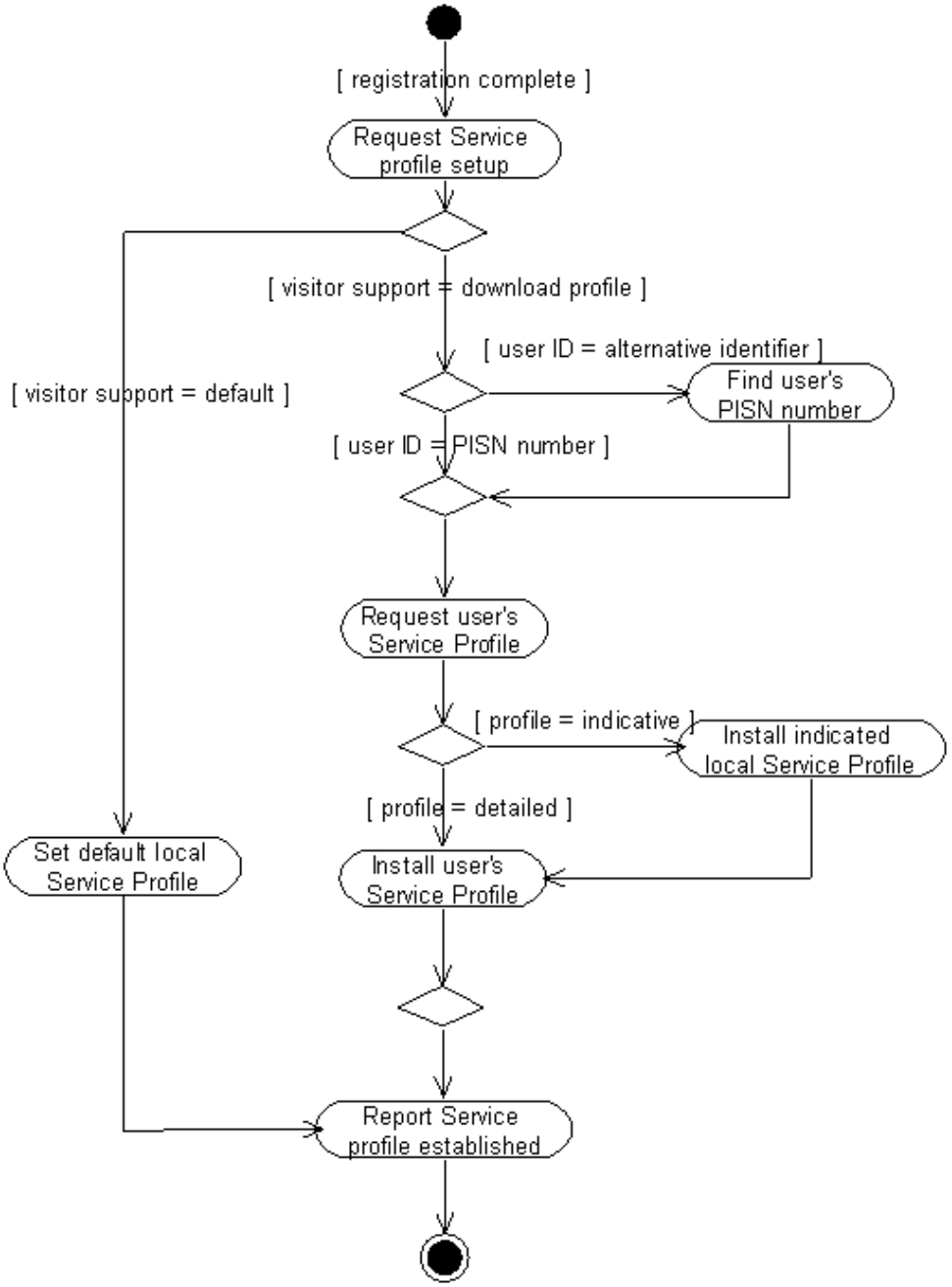

Figure 47: Activity diagram describing the "Specify Profile" use case

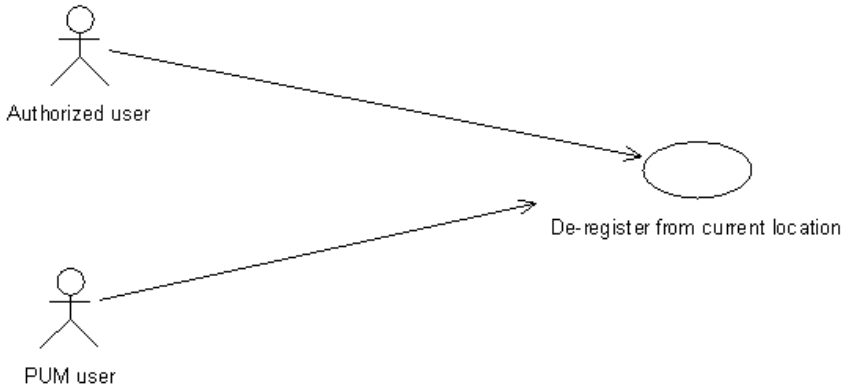

Figure 48: PUM De-registration use case diagram

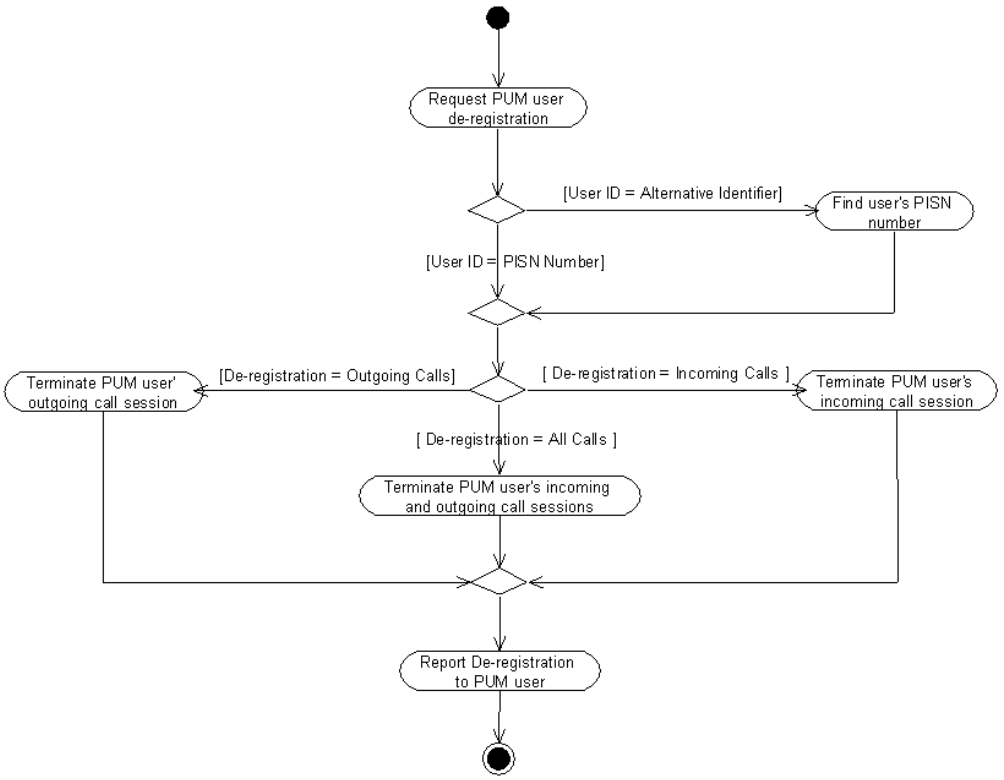

Figure 49: Activity diagram describing the "De-register from current location" use case

A.2.3 Specification Model

The Specification Model draws on the information presented in the Context Model and the Requirements Model as well as the existing PUMR Stage 3 standard, ISO/IEC 17576 [10] to offer a set of UML diagrams from which it would be possible to start developing a detailed behaviour specification in SDL. This "reverse engineering" approach would not normally be used as the purpose of using UML is to end up with a Stage 3 standard (or something similar). In this case, it was used to ensure that the UML specification is fully aligned with the "resultant" standard.

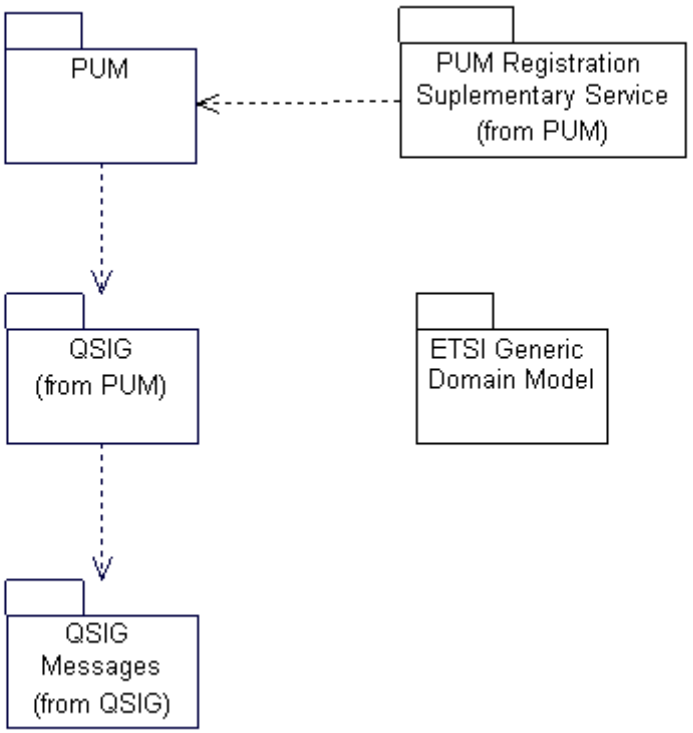

Figure 50: Specification Model packages

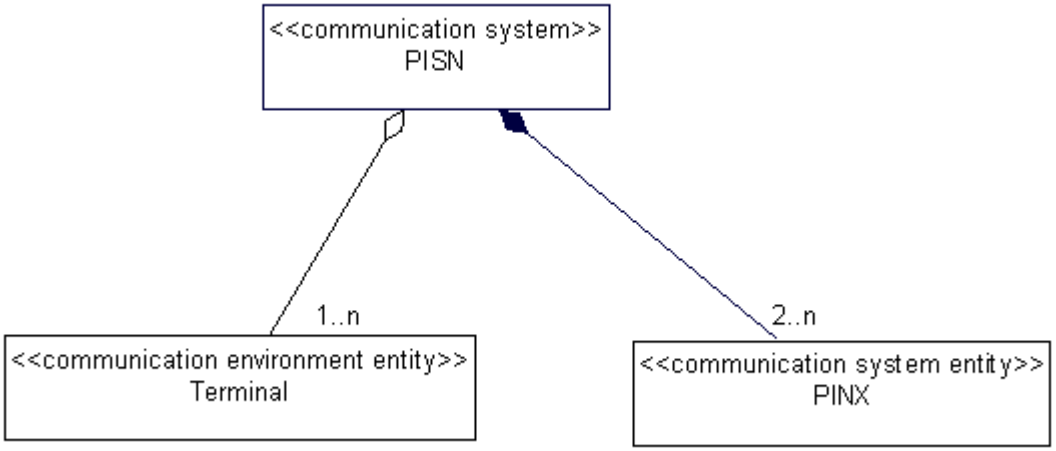

Figure 51: Basic Domain Model (from Context Model)

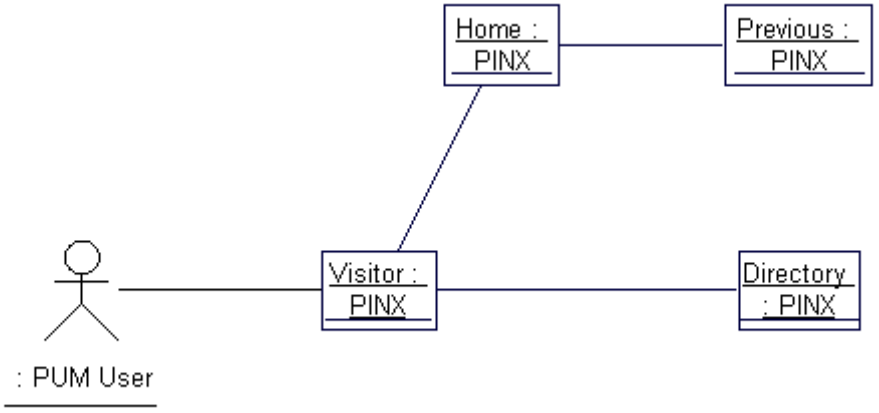

Figure 52: PUMR Object Model

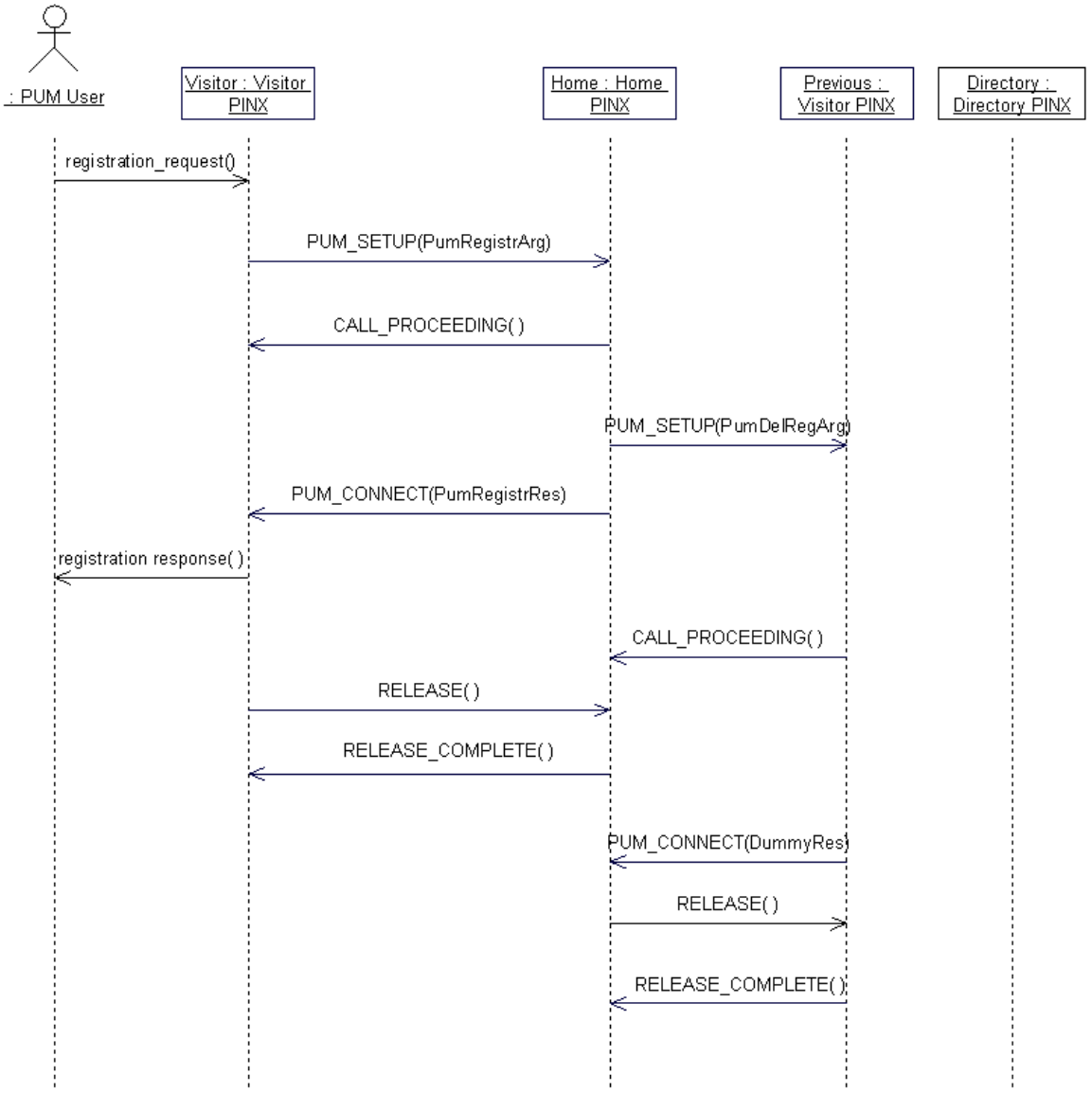

Figure 53: Example sequence diagram showing registration using the PUM Number

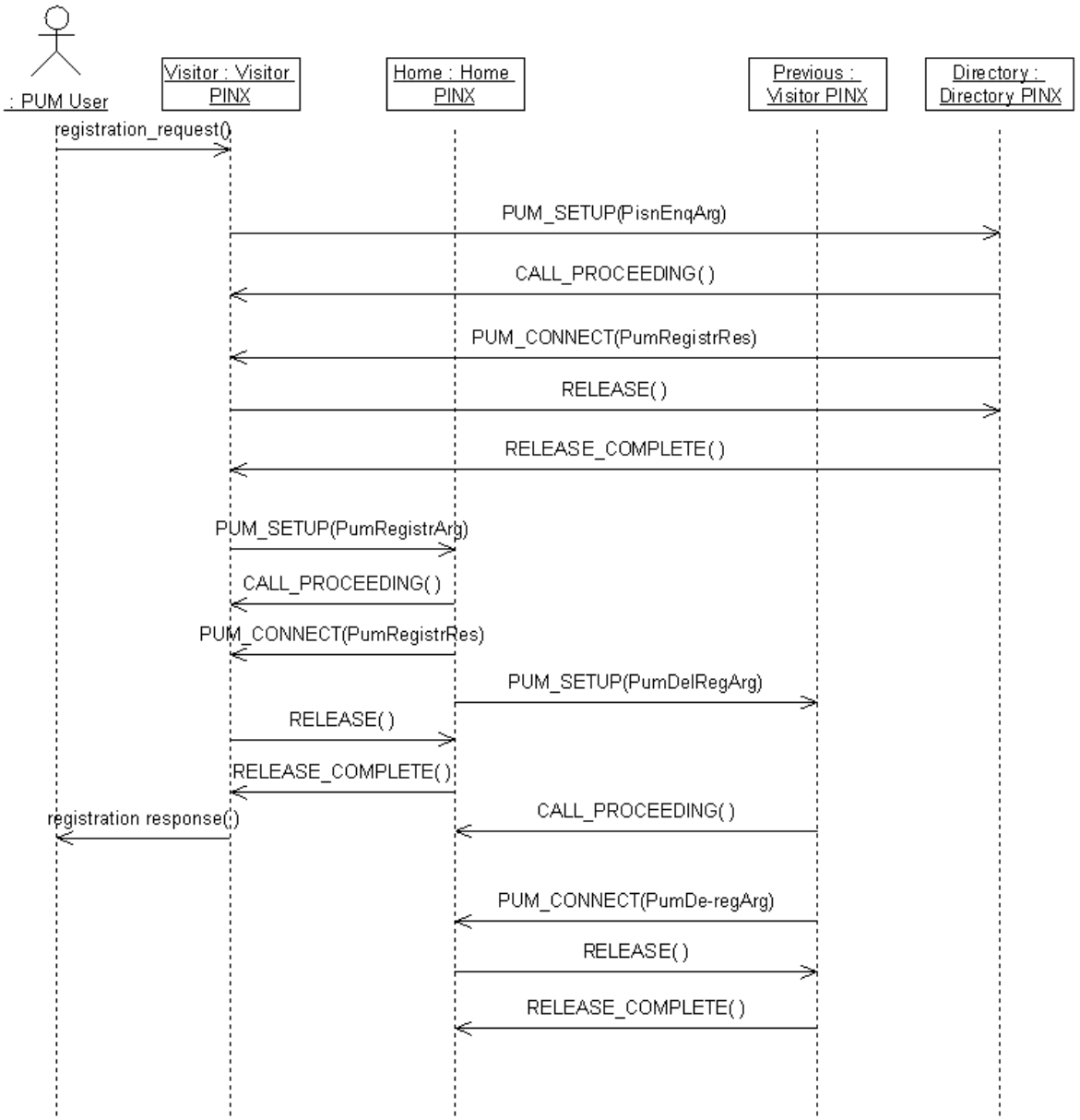

Figure 54: Example sequence diagram showing registration using Alternative Identifier

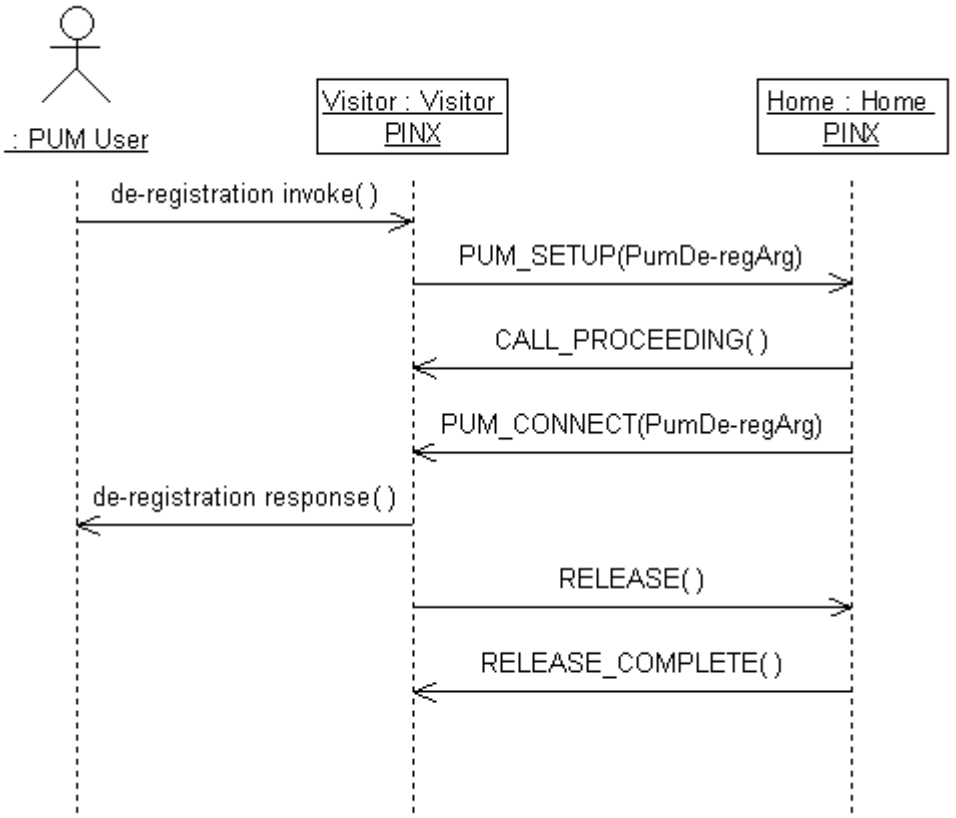

Figure 55: Example sequence diagram showing de-registration

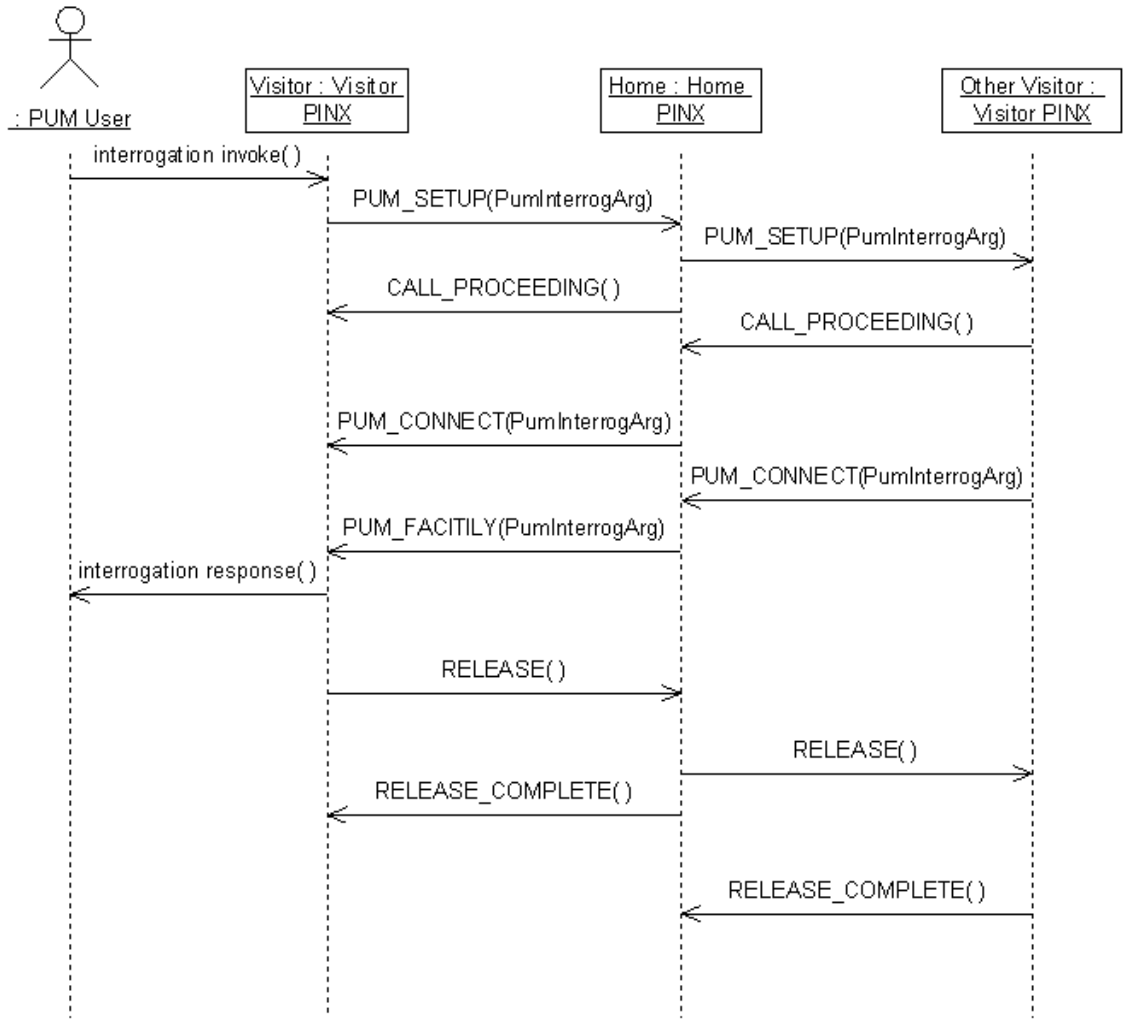

Figure 56: Example sequence diagram showing PUMR interrogation

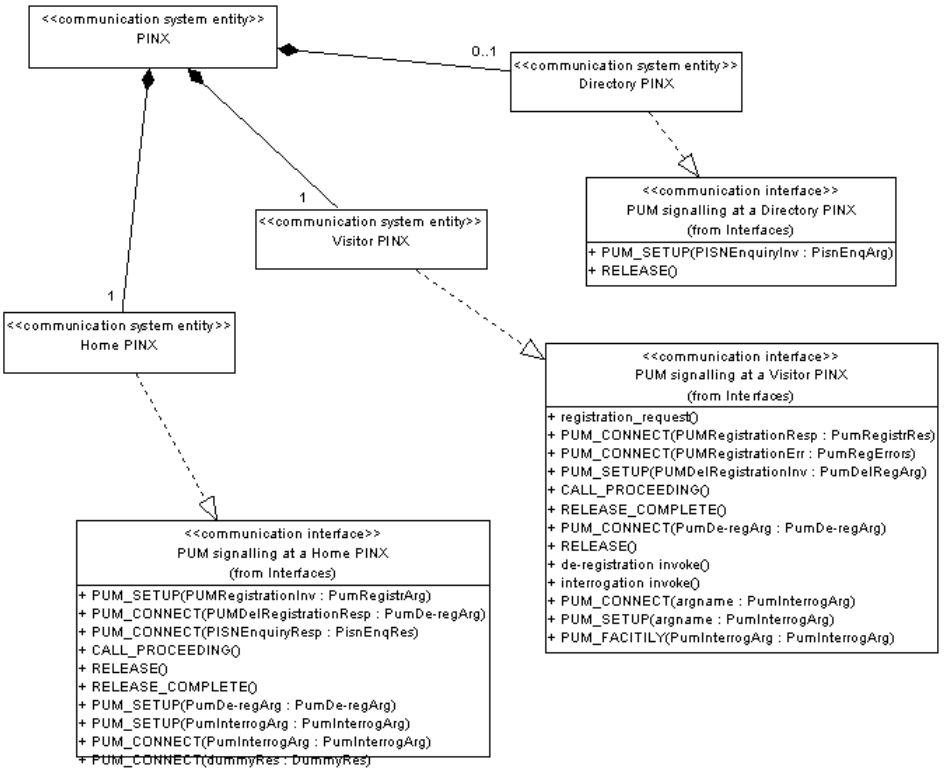

Figure 57: PUMR detailed Domain Model

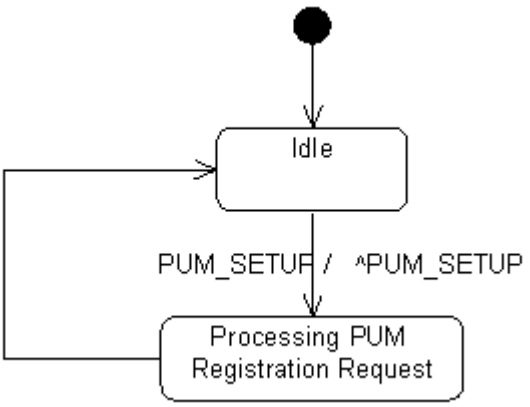

Figure 58: Statechart diagram showing the registration processing at the Home PINX

NOTE: Figure 58 and Figure 59 are examples of a statechart diagram and a sub-diagram that could be developed for the PUMR supplementary service. The Specification Model is incomplete at this point and does not include any further statecharts.

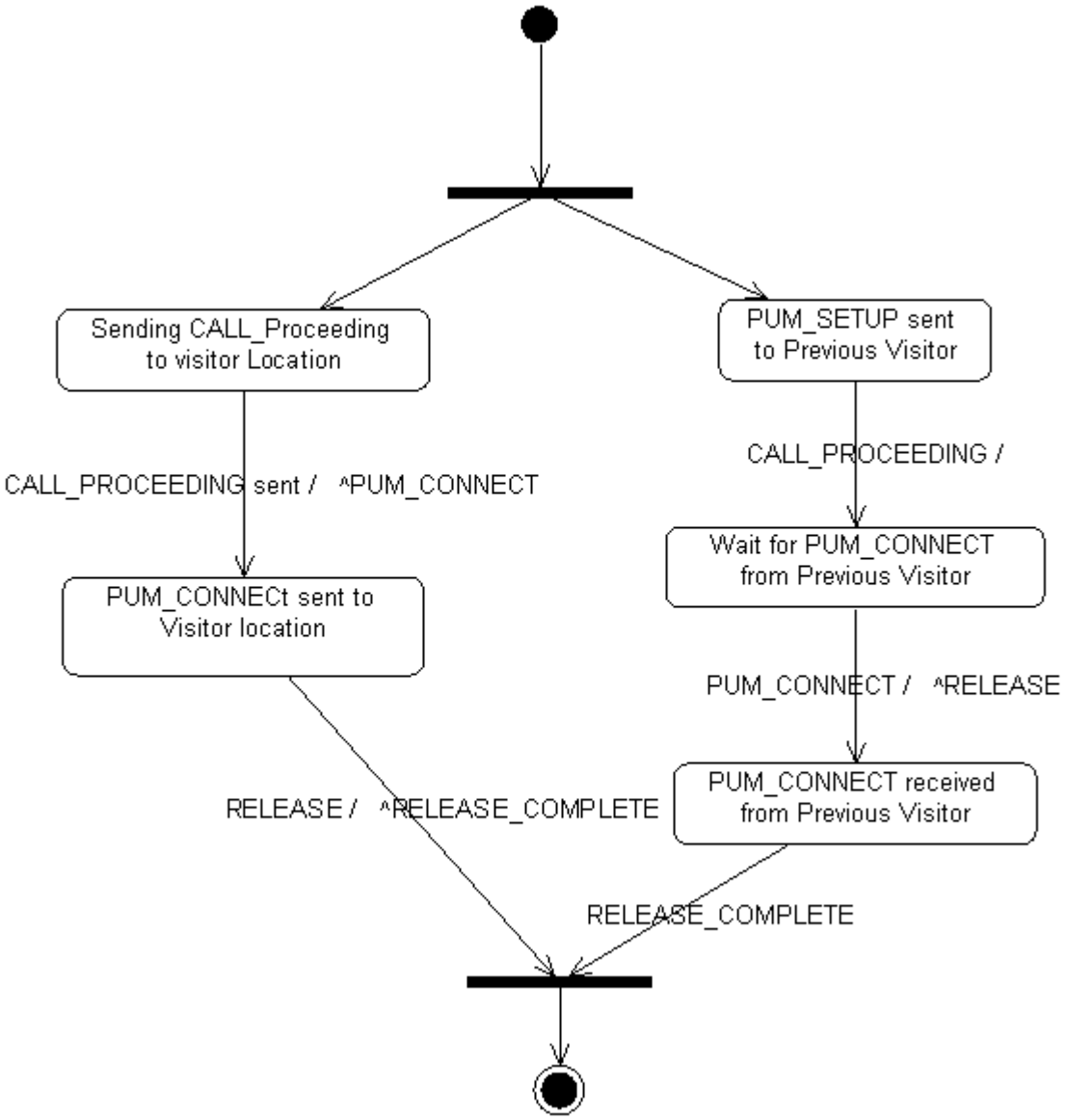

Figure 59: Statechart sub-diagram showing the detailed processing of a registration request at the Home PINX

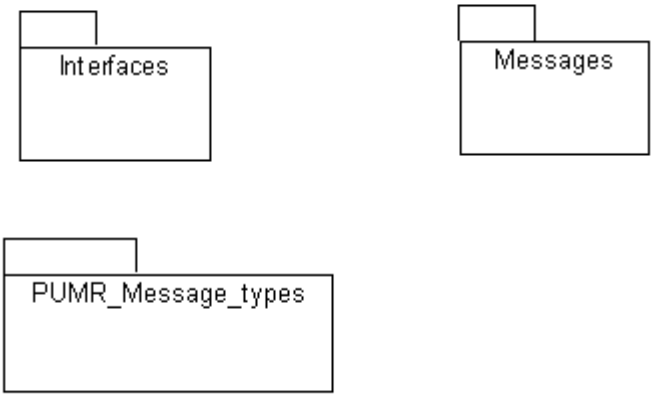

Figure 60: PUMR message-specific packages

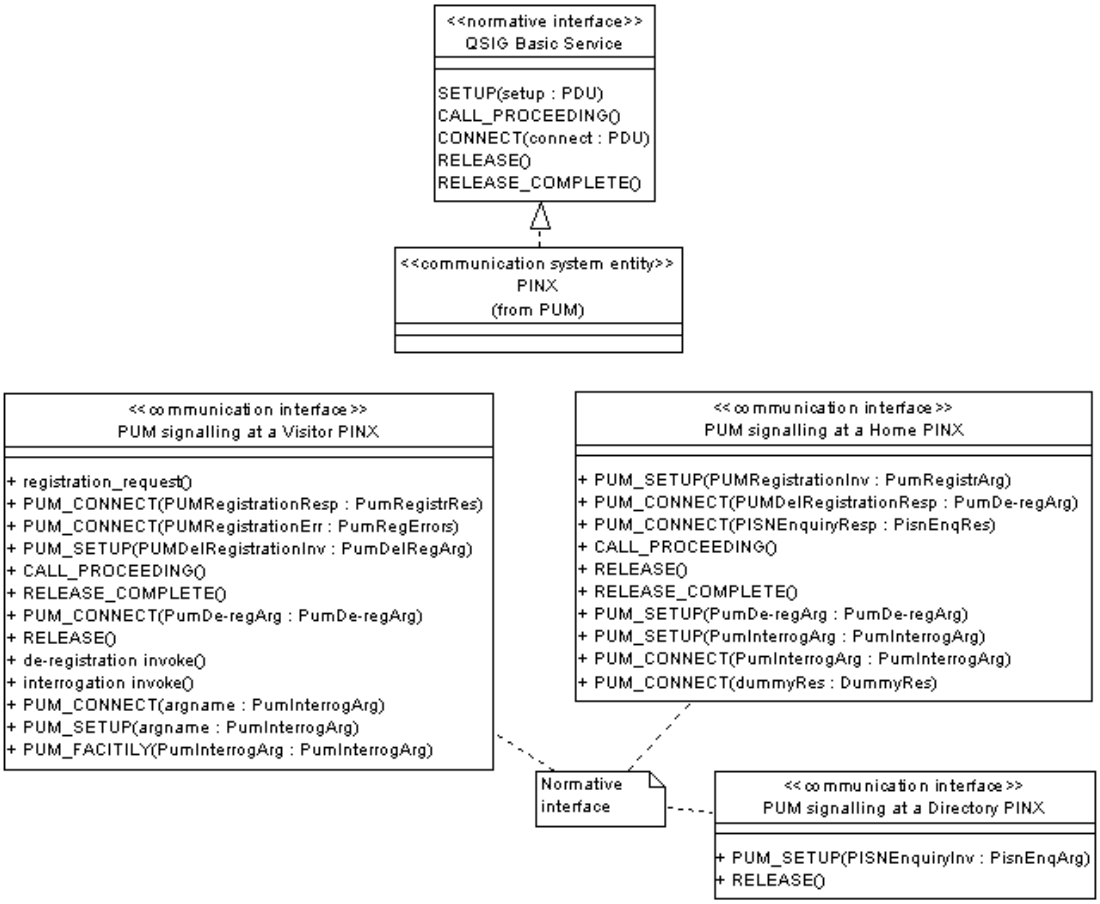

Figure 61: Identification of PUMR signalling at the QSIG interfaces

Figure 62: Identification of the two QSIG signals used for carrying PUMR message information

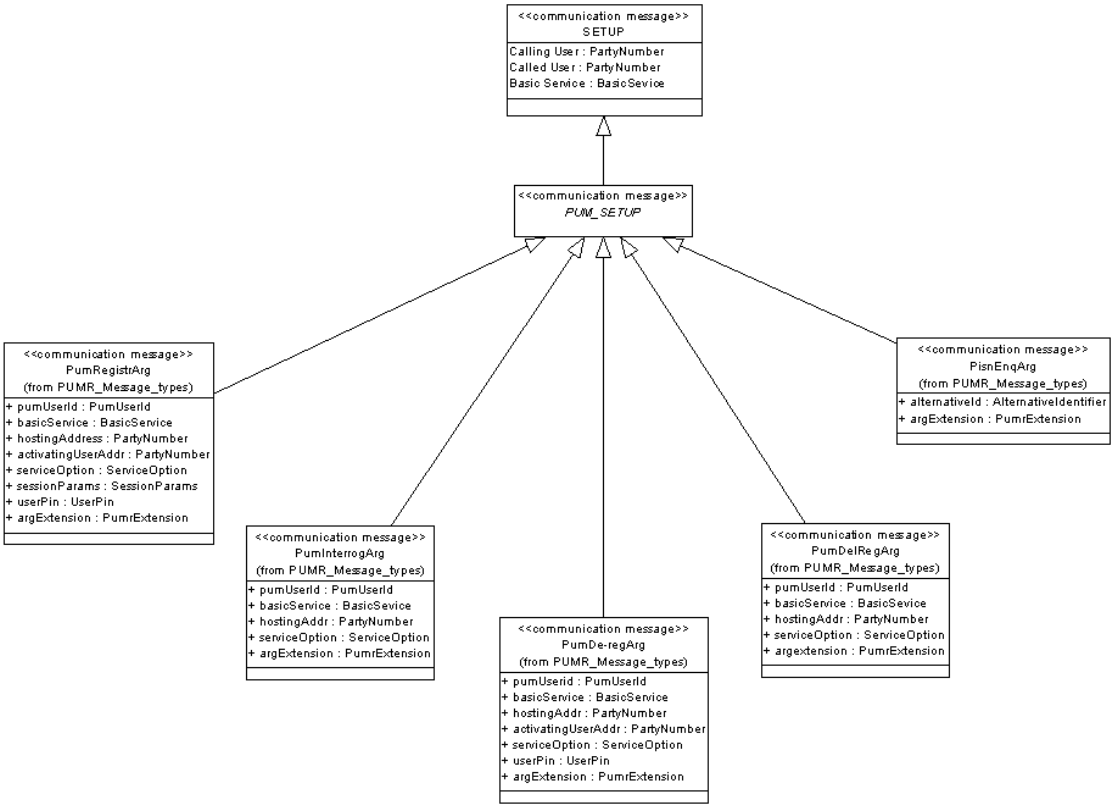

Figure 63: PUMR message contents carried in the SETUP signal

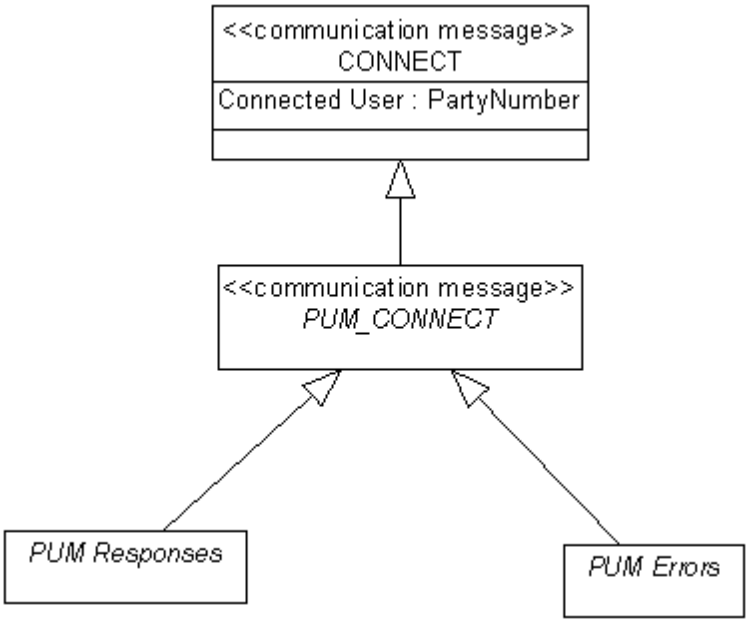

Figure 64: PUMR message types carried in the CONNECT signal

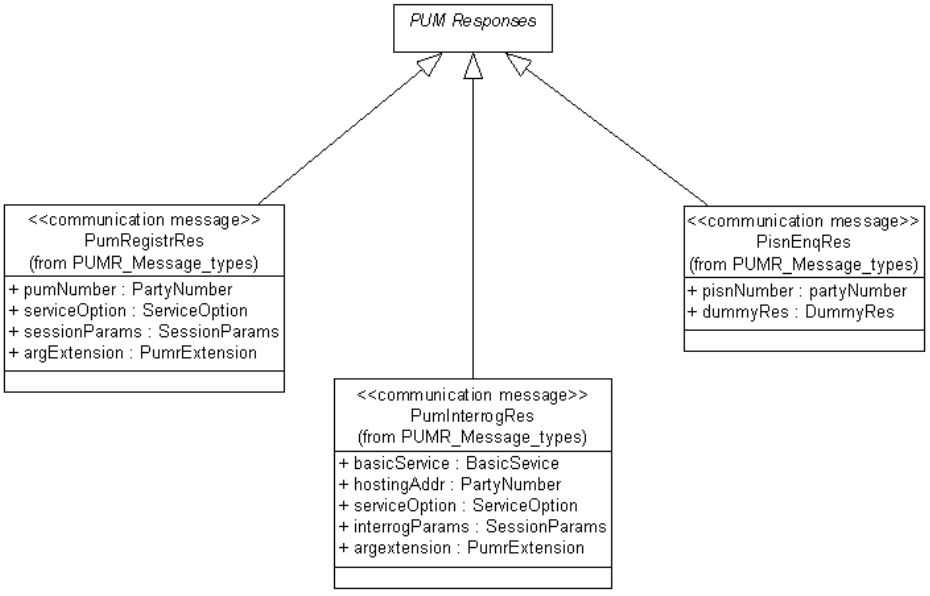

Figure 65: Contents of the PUMR response messages

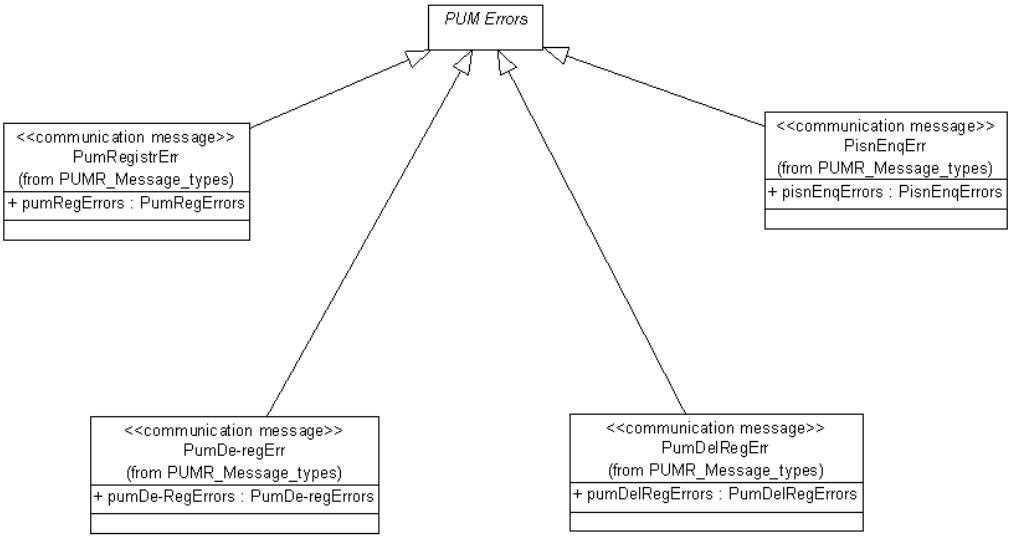

Figure 66: Contents of the PUMR error messages

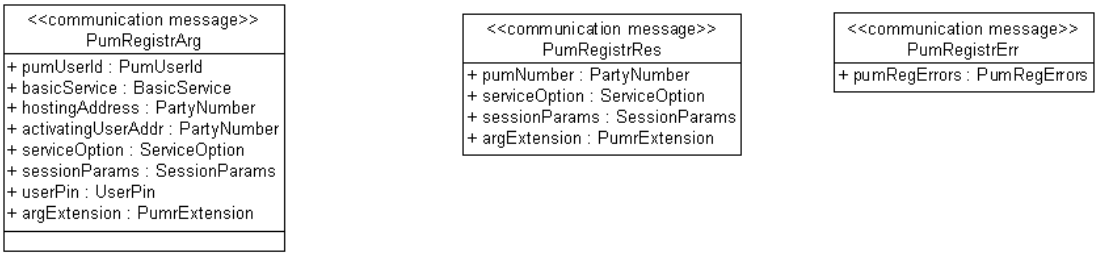

Figure 67: PUMR registration message types

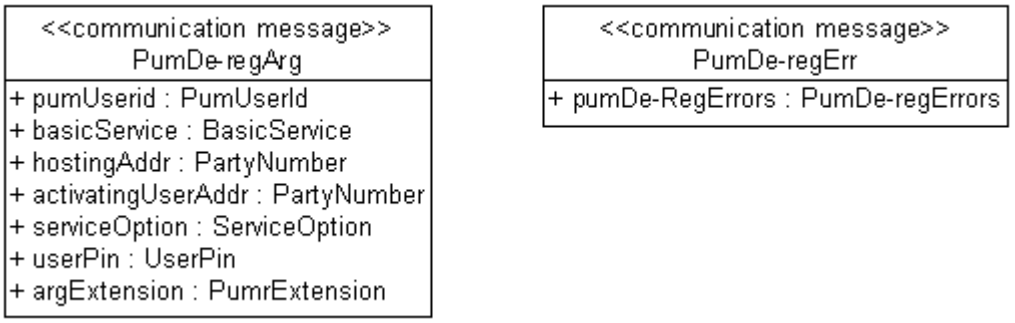

Figure 68: PUMR de-registration message types

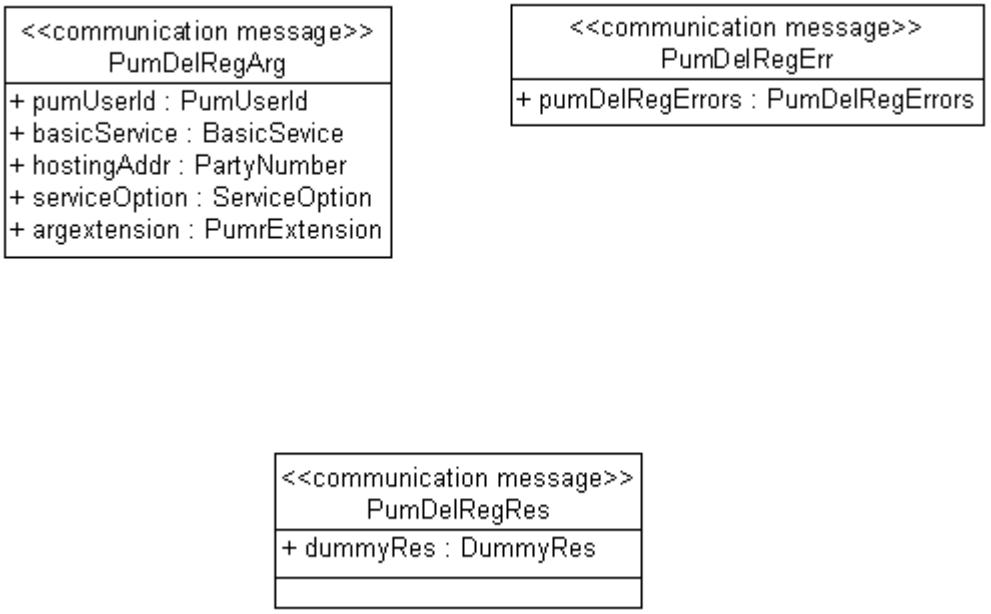

Figure 69: PUMR delete registration message types

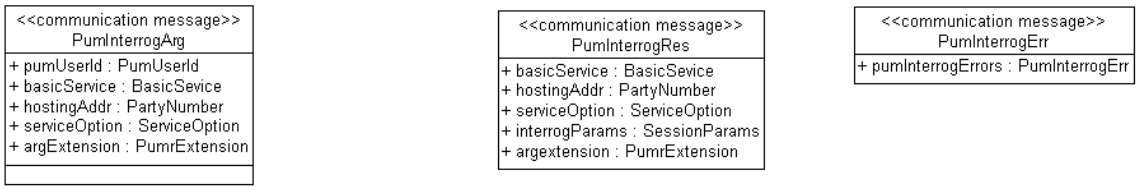

Figure 70: PUMR interrogation message types

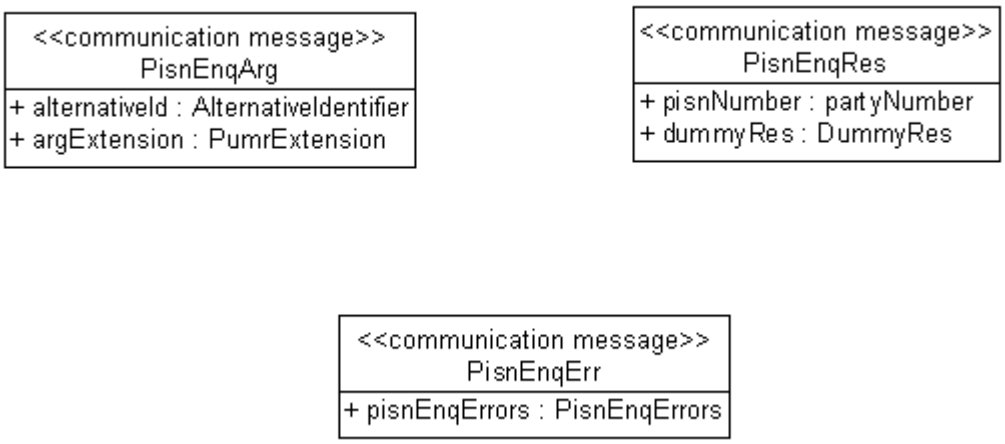

Figure 71: PISN enquiry message types

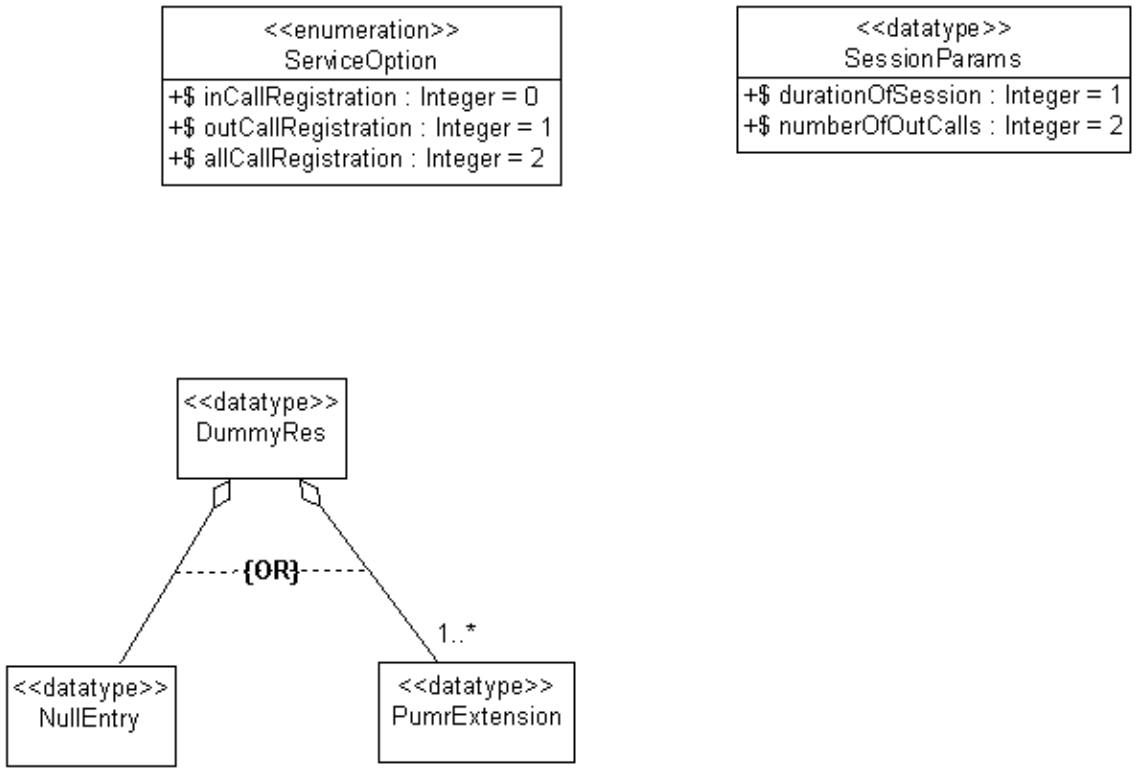

Figure 72: PUMR general data types

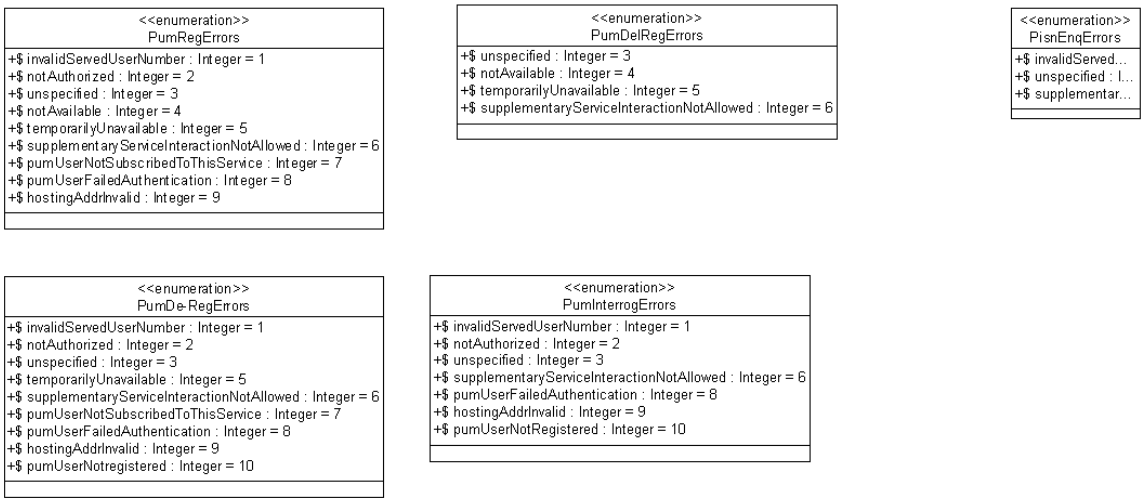

Figure 73: PUMR error codes

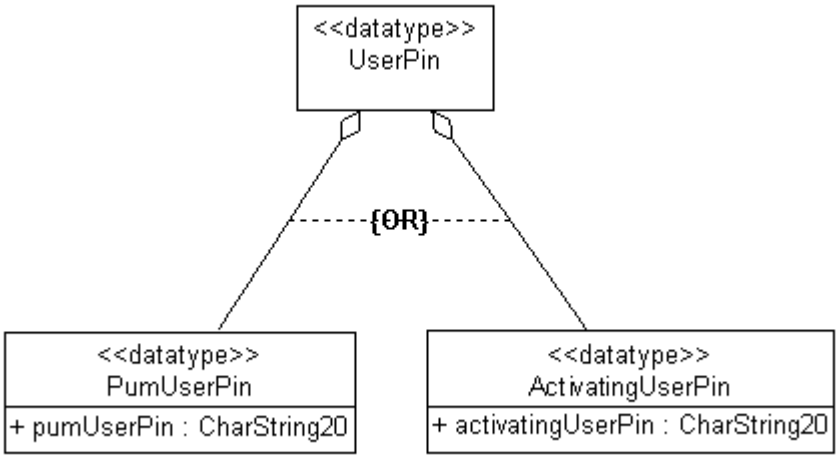

Figure 74: Type specification of PUM user PIN

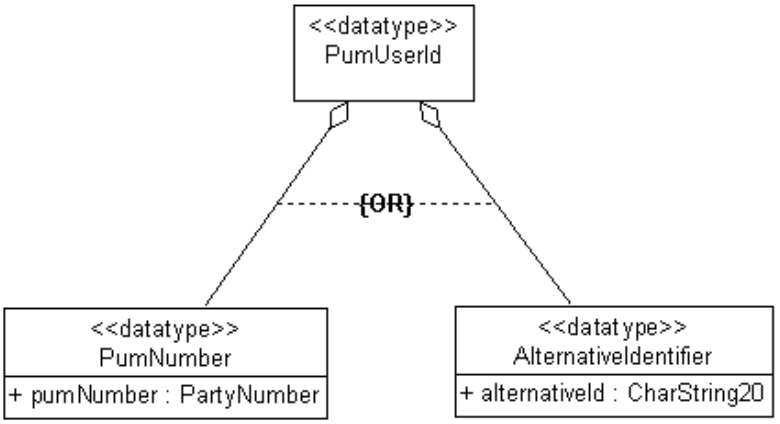

Figure 75: Type specification of PUM user identifier

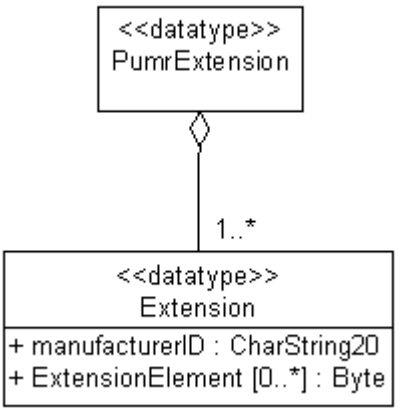

Figure 76: Type specification of PUMR message extension

Figure 77: QSIG message packages not specific to PUMR

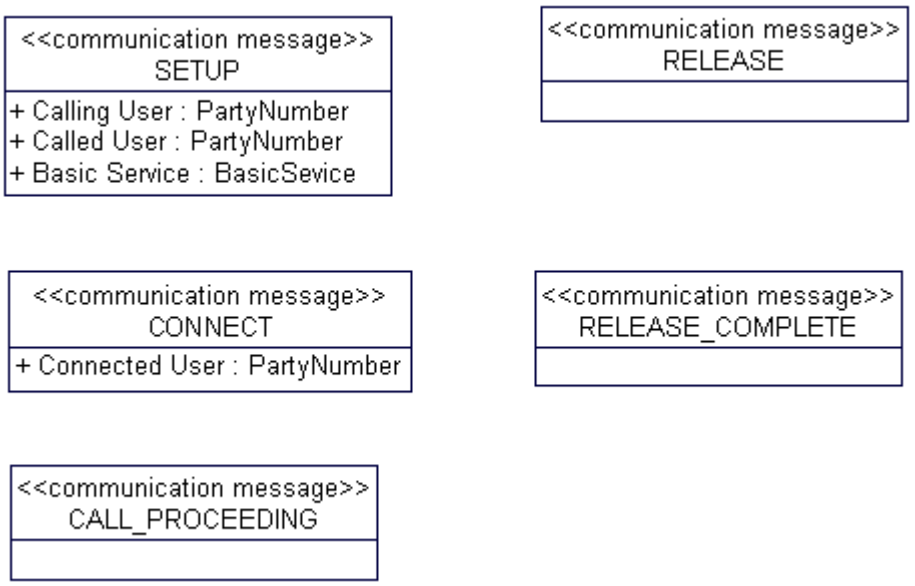

Figure 78: QSIG basic service messages

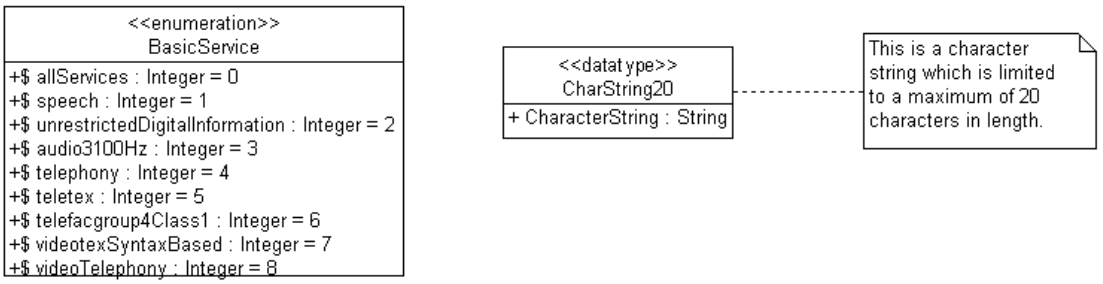

Figure 79: QSIG general data types

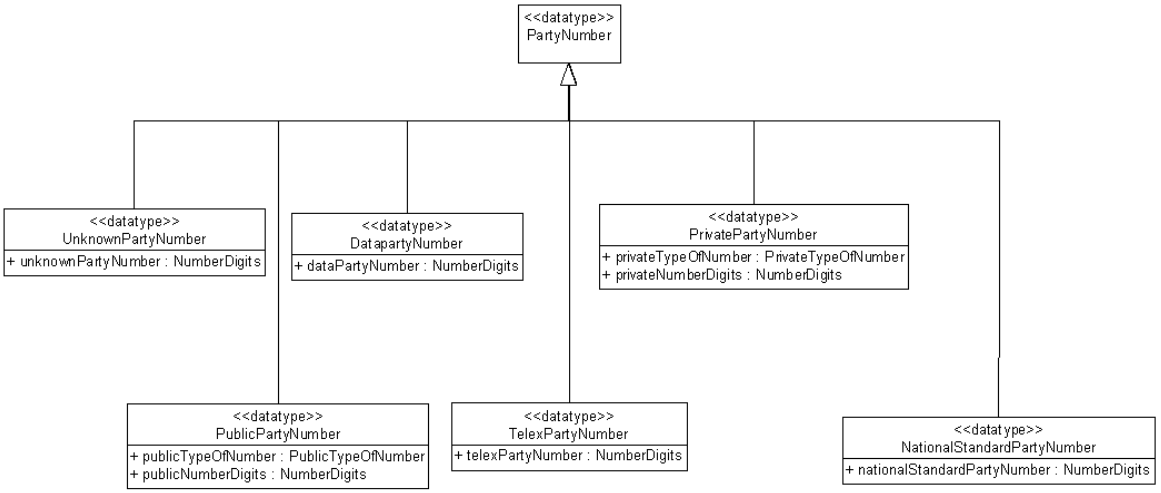

Figure 80: Type specification of QSIG party number

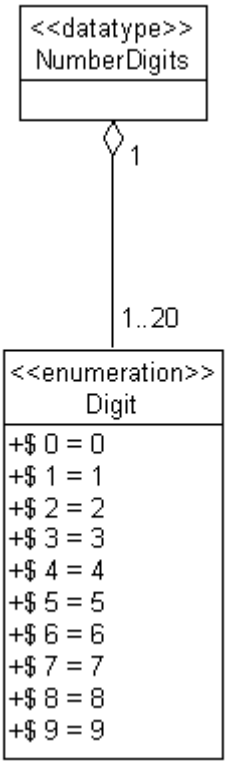

Figure 81: Type specification of QSIG digit string

A.2.4 Testing Model

The Testing Model is derived from the Context and Specification Model and bridges the gap between the functional specification and its associated test suite.

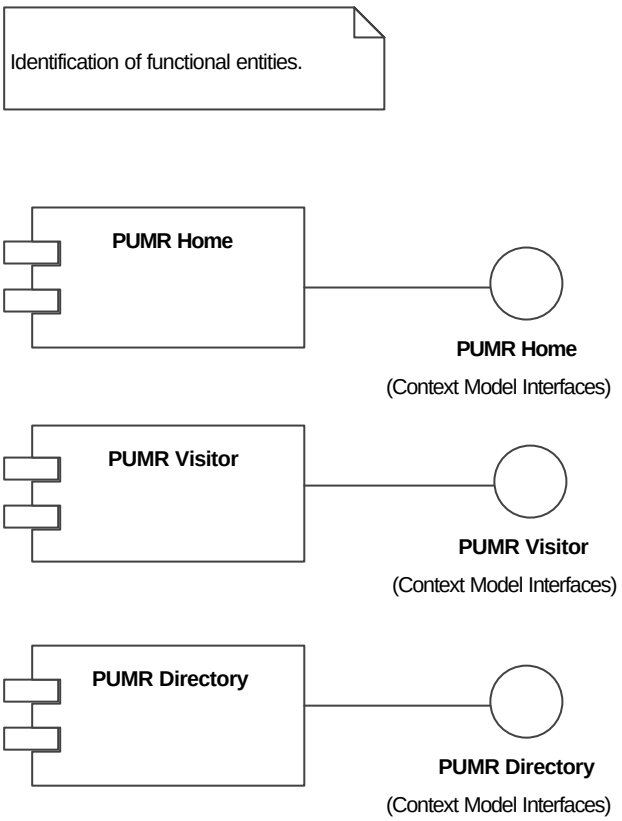

Figure 82: Functional entities

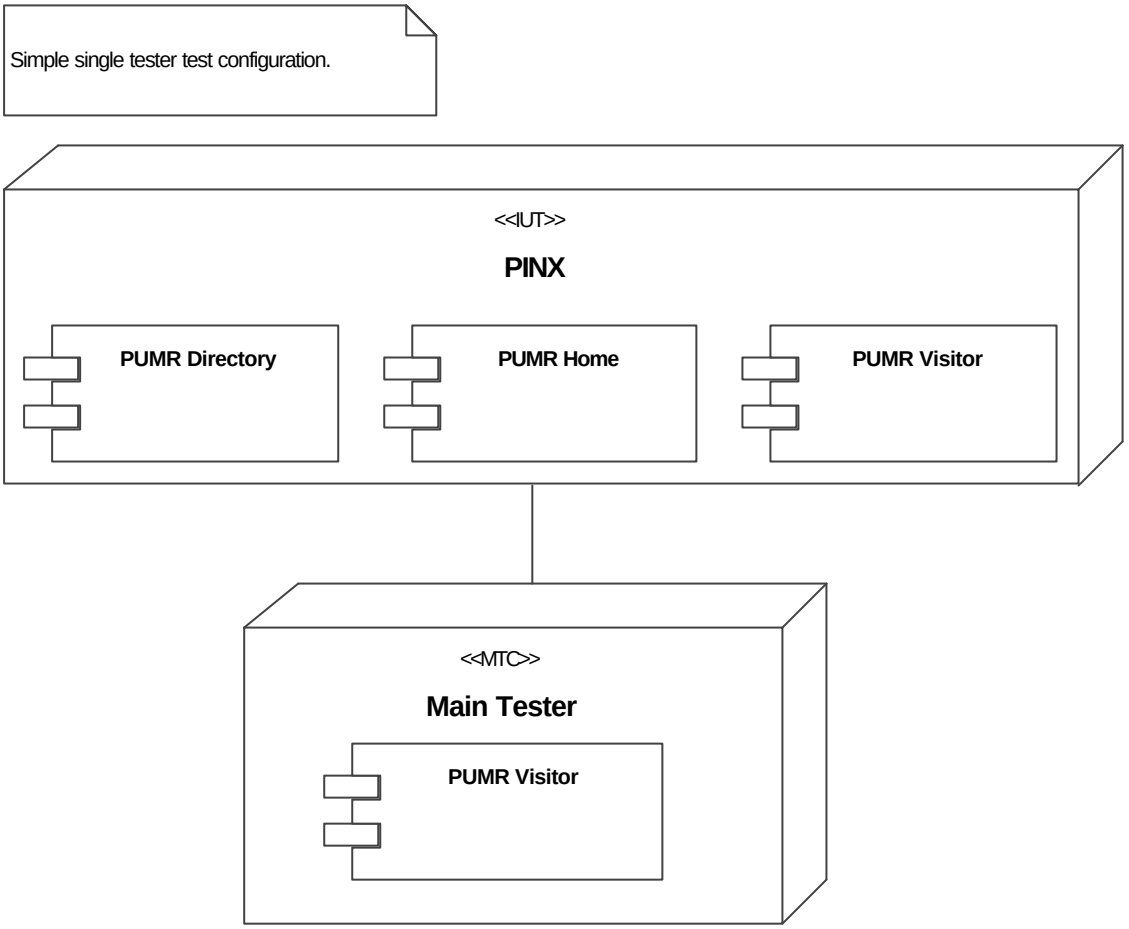

Figure 83: Test configuration with a single tester

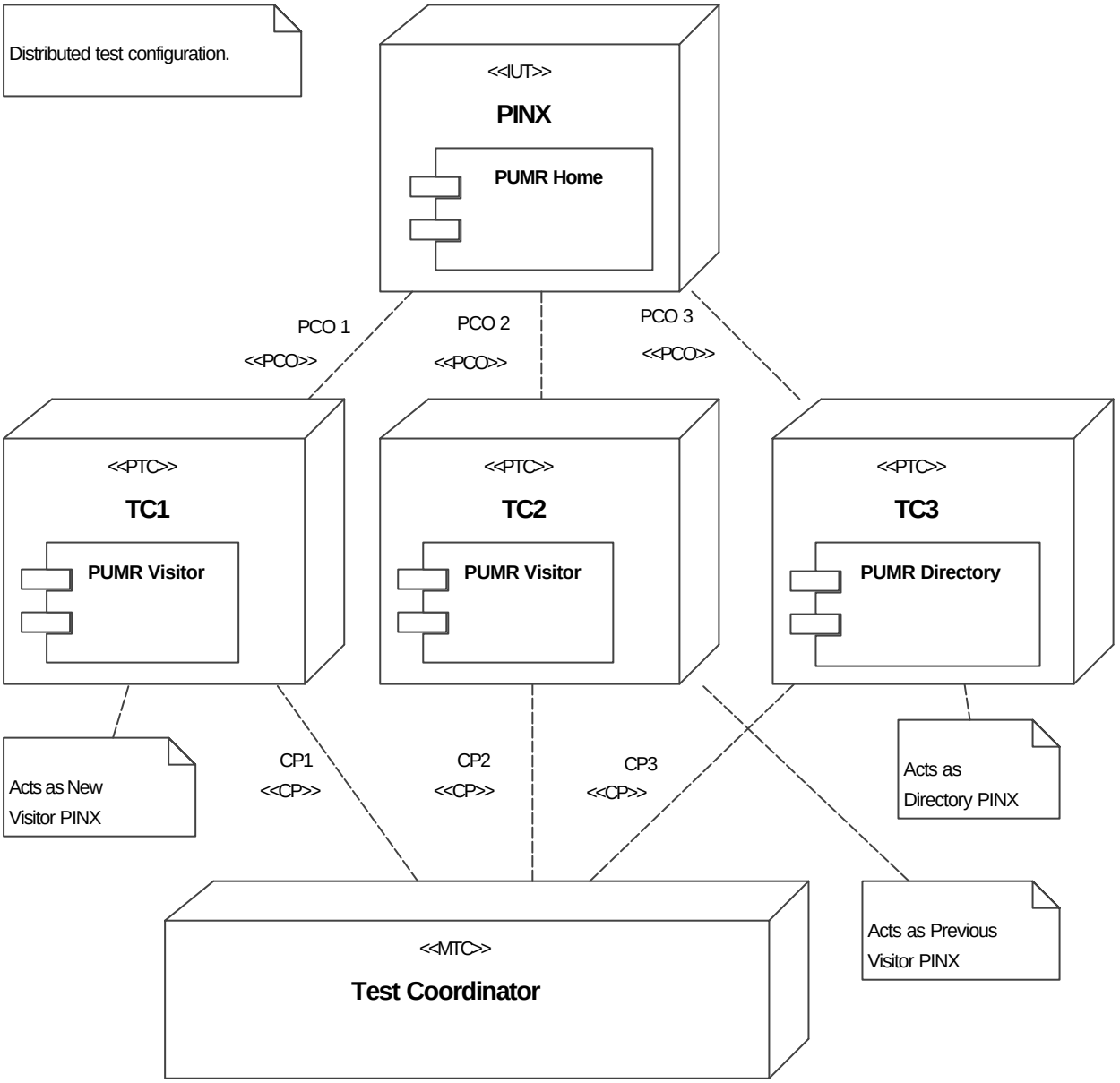

Figure 84: Test configuration with distributed testers

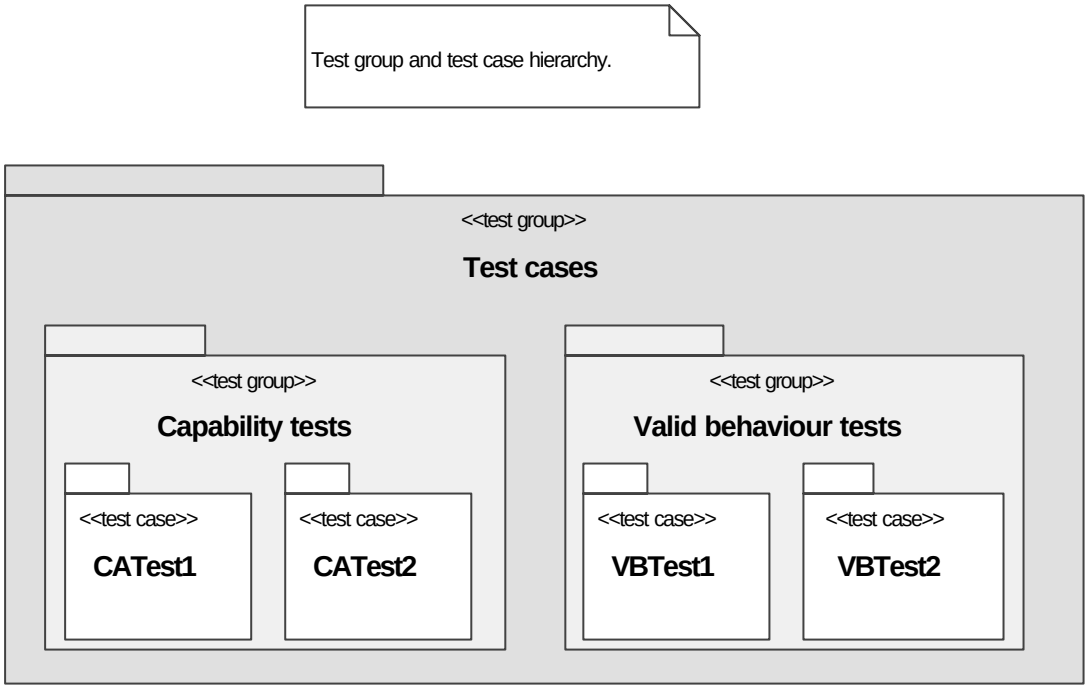

Figure 85: Test case structure

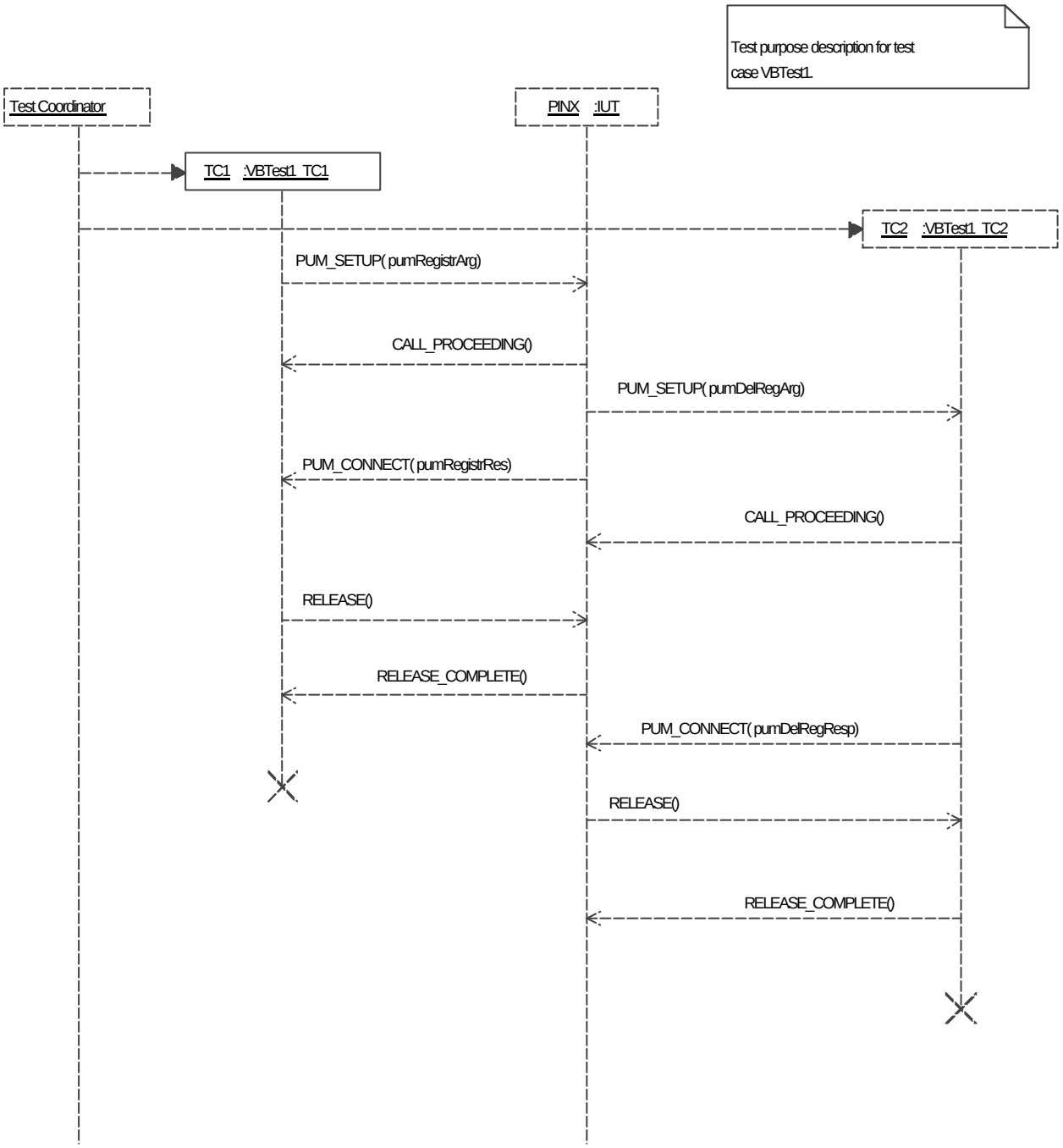

Figure 86: Test purpose description

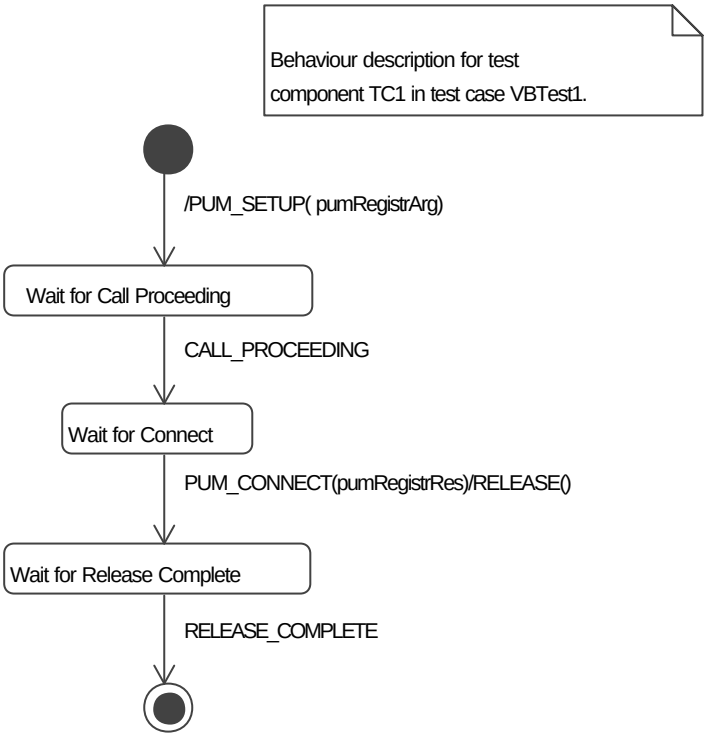

Figure 87: Behaviour description for test component TC1

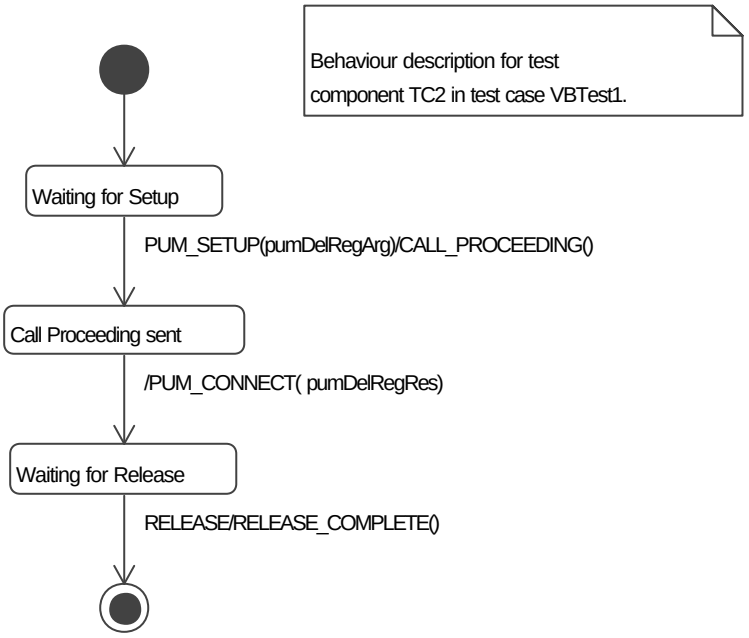

Figure 88: Behaviour description for test component TC2

History

Document history		
1.0	Sept 1999 – May 2000	First draft
2.0	May 2000 – Sept 2000	Revised structure of the document to accommodate UML Activity Diagrams describing the individual elements of the standards development process.