TR 101 023-2 V1.1.1 (1997-04)

Technical Report

Methods for Testing and Specification (MTS); Application of object-oriented SDL features in B-ISDN specifications

ETSI

European Telecommunications Standards Institute

Reference

DTR/MTS-00037-2 (9h0i0ics.PDF)

Keywords

B-ISDN, broadband, ISDN, methodology, SDL

Contents

Intellectual Property Rights

Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).

The present document is part 2 of a multi-part Technical Report describing how SDL can be formally used by technical bodies, as identified below:

Part 1: "Portability of SDL specifications";

Part 2: "Application of object-oriented SDL features in B-ISDN specifications".

Introduction

In the context of full formalization of some B-ISDN specifications, the use of object oriented features of SDL, as defined in SDL 92 is investigated. Since ETS 300 414 [1], which regulates the use of SDL in ETSs, does not treat the object-oriented constructs of SDL 92, a limited study was undertaken in PT86 with a task to create first suggestions on their use.

The main goals are to support re-usability and specialization while maintaining a high level of readability.

The SDL model used as a case study in the present document is the Q.2931 UNI (User and Network side) developed by the PEX and PT 87. This model uses no aspects of SDL 92. The purpose of this model is to act as a detailed specification of Q.2931 and to provide a basis for Computer-Aided Test case Generation (CATG). The following requirements were noted as being applicable to this study:

- The Q.2931 specification models both sides of the UNI (i.e. the User and the Network sides) while in test case generation, for example, architectures with just one side (i.e. User *or* Network) are needed. That is, we need a base specification from which different configurations can easily be derived.
- The CATG model requires a greater degree of detail. The use of specialization may be applicable here.
- Integration of new features from CS-2 may be facilitated by using object-orientation.

At the same time it was also felt interesting to investigate creation of models where interworking of B-ISDN and IN can be analysed.

1 Scope

This Technical Report (TR) describes results of initial studies of the use of object-oriented SDL features in ETSI deliverables. The purpose of this document is to show some ways of using object-orientation to facilitate re-use of specifications. The ideas presented are applicable to all standards that specify behaviour, such as protocols and services. This Technical report complements the methodologies described in ETS 300 414 [1] and ETR 298 [2].

2 References

References may be made to:

a) specific versions of publications (identified by date of publication, edition number, version number, etc.), in which case, subsequent revisions to the referenced document do not apply; or

b) all versions up to and including the identified version (identified by "up to and including" before the version identity); or

c) all versions subsequent to and including the identified version (identified by "onwards" following the version identity); or

d) publications without mention of a specific version, in which case the latest version applies.

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] ETS 300 414 (1995): "Methods for Testing and Specification (MTS); Use of SDL in European Telecommunication Standards; Rules for testability and facilitating validation".

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

3 Abbreviations

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

SDL	Specification and Description Language
UNI	User Network Interface
PEX	Permanent EXperts (Protocol description methodology experts from the ETSI Secretariat competence centre)
CATG	Computer-Aided Test case Generation
CS-2	Capability Set 2
IN	Intelligent Network
INAP	Intelligent Network Application Protocol

4 SDL 92 concepts used

In this study the following constructs of SDL 92 are felt useful:

- packages;
- system, block and process type definitions;
- virtuality constraints as indicated by keywords **virtual**, **redefined** and **finalized**;
- gate definitions;

- specialization of types using keywords **inherits** and **atleast**;
- system, block and process instances as objects of a given class (type).

In addition the use of continuous signals (SDL 88) is treated in relation to analysis of parameters received with a signal.

5 Analysis of re-use based on object-oriented SDL

5.1 Goals

When planning for re-use it is possible to aim at several goals. The goals that were investigated in this study are as follows:

- create one system type definition that is complete (including instantiation of blocks and processes) that can be further specialized;
- create several system type definitions based on common definitions of block and process types, data, signal and signallist definitions. Several system type definitions can be used for different purposes such as validation, test case generation etc. Instantiation of block and process types is different in different system types. As in the first situation definitions could be further specialized;
- provided two systems such as B-ISDN UNI and IN INAP interworking needs to be investigated, definitions of relevant types should be imported so that a combined system type is defined.

Common to all situations should be that any definition such as block or process types etc. should be given in one place only.

What can and cannot be achieved, together with some advantages or possible problems will be presented through two examples.

In parallel to this evaluation the problem of analysis of parameters of incoming messages is addressed in example III.

5.2 Example I

Figure 1 represents an illustration of system type definition. The approach briefly described here is natural and uses structuring principles that are characteristic for SDL language, notably information hiding principles (definitions visible only where needed). Other positive characteristics are as follows:

- system is defined containing two block type definitions for Network and User side respectively;
- not shown are data, signal and signallist definitions that should be within the system definition;
- block types are virtual, which means that the system that inherits from this one can redefine their gate definitions and their internal structure;
- provided block definitions also contain virtual process type definitions, they can also be specialized (changes in process diagrams and gate definitions).

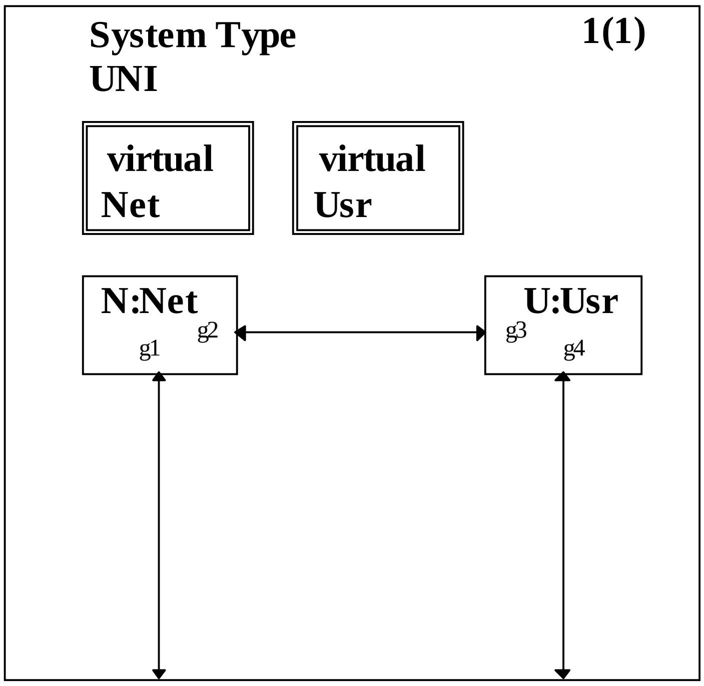


Figure 1: System type definition

This approach is very suitable for situations where one capability set is described in one system definition and enriched capability set is defined as its specialization.

In terms of other aspects of re-usability this approach has some limitations:

- only the system type definition that is visible at the first level of some package can be re-used and specialized;
- the specialized system inherits the block instances and channels, nothing can be removed or reconnected in some other way (additional channels as well as block types and block instances are, of course, possible);
- parts of this system (block or process types) cannot be re-used in other systems;
- parts of other systems, defined in some other packages in the same manner, cannot be added to this system.

In spite of the above mentioned limitations, this approach has enough good characteristics to be used in many situations. Last but not least, the feeling is that the readability of documents containing such specifications may be better than when other approaches are used.

5.3 Example II

In example two, basically a very similar system is defined using a different approach. The full example that illustrates this approach is given in annex A. The names of entities given in brackets can be found in the annex A. The approach is as follows:

- two block types representing the network (Q2931_Net) and user side (Q2931_Usr) are defined outside the system in the package (Q2931paDefs);
- in order to do that, data, signals and signallists need to be defined outside the system in the package as well;
- being defined outside of the type definition, the block types cannot be defined as virtual;
- block type definitions contain virtual process type definitions (Network and User respectively);
- in the same manner two additional blocks are defined representing the user (UserAppl) and the network (NetAppl) applications respectively;

- a system type (Q2931_UNI_blocksOnly) is defined that contains only virtual block type definitions for network (Q2931_N) and user (Q2931_U) side and network (NetApplBlock) and user (UserApplBlock) application. These virtual type definitions are bound to matching block type definitions at the package level using the keyword **atleast**;
- this system type can be used to create new system types that inherit all virtual type definitions and add different block instances and channels;
- in this example a system (Q2931_UNI_chanAdded) with block instances for network and user side are specified and connected with channels to one another and to the environment;
- another system type can be defined that also has block instances for user and network applications connected with channels to user and network block instances, thus creating a closed system. This may, for example be suitable for validation or test case generation.

Suppose that an enriched capability set needs to be specified so that elements of this specification are inherited and new data, signals, signallists and type re-definitions are added. This can be done in a new package (Q2931paRedefs) where:

- block types defined at package level inherit from their matching block type definitions from the basic package adding new elements to gate definitions (dashed gate symbols) and re-defining process type definitions (Q2931_Net2 and Q2931_Usr2);
- new data, signal and signallist definitions are specified in the package;
- a new system type (Q2931_UNI_chanAdded_R) is defined that inherits system type (Q2931_UNI_chanAdded) from basic capability set package (Q2931paDefs) and contains redefined block type definitions for network and user side and network and user application. These redefined type definitions are bound to matching block type definitions at the package level using again the keyword **atleast**;
- the system contains also dashed block instances connected with a channel carrying signals that are added.

In terms of re-usability this approach has the following advantages:

- each type definition is defined in exactly one place;
- that definition is on the package level which means that it can be used in several system type definitions that can be used for different purposes;
- similar parts defined in other packages in the same manner can be combined into system type definitions useful for interworking investigations.

In principle, process type definitions as well as procedure definitions can also be defined outside of system and block type definitions at the package level. For this particular example it was felt that this was not necessary.

Readability of specifications based on this approach appears to be somewhat reduced. On the other hand the fact that definitions are never duplicated and the size of overall specifications may be reduced is surely a good characteristic.

5.4 Example III

Example III addresses a completely different problem. A very common situation, notably present in B-ISDN standard Q.2931 is the need to specify in SDL the analysis of parameter values received with a particular signal. This situation is sometimes specified so that a rather complicated tree of decisions follows the signal reception. This very often takes much more space than one page and combined with many joins leads to very poor readability and maintainability. The use of procedures sometimes can help, but the situation gets complicated if signals need to be sent from within procedures.

Also, sometimes instead of actual analysis of input parameters, various cases are modelled in a non-deterministic way using informal text in decisions. This decouples input values from decisions that follow and may lead to potential problems in particular in relation to testing. To illustrate the problems, an information element is said to be optional, but should be sent forward if present in received message. The latter request may not be met if decisions are not based on received parameters.

One approach that might help is shown in figure 2.

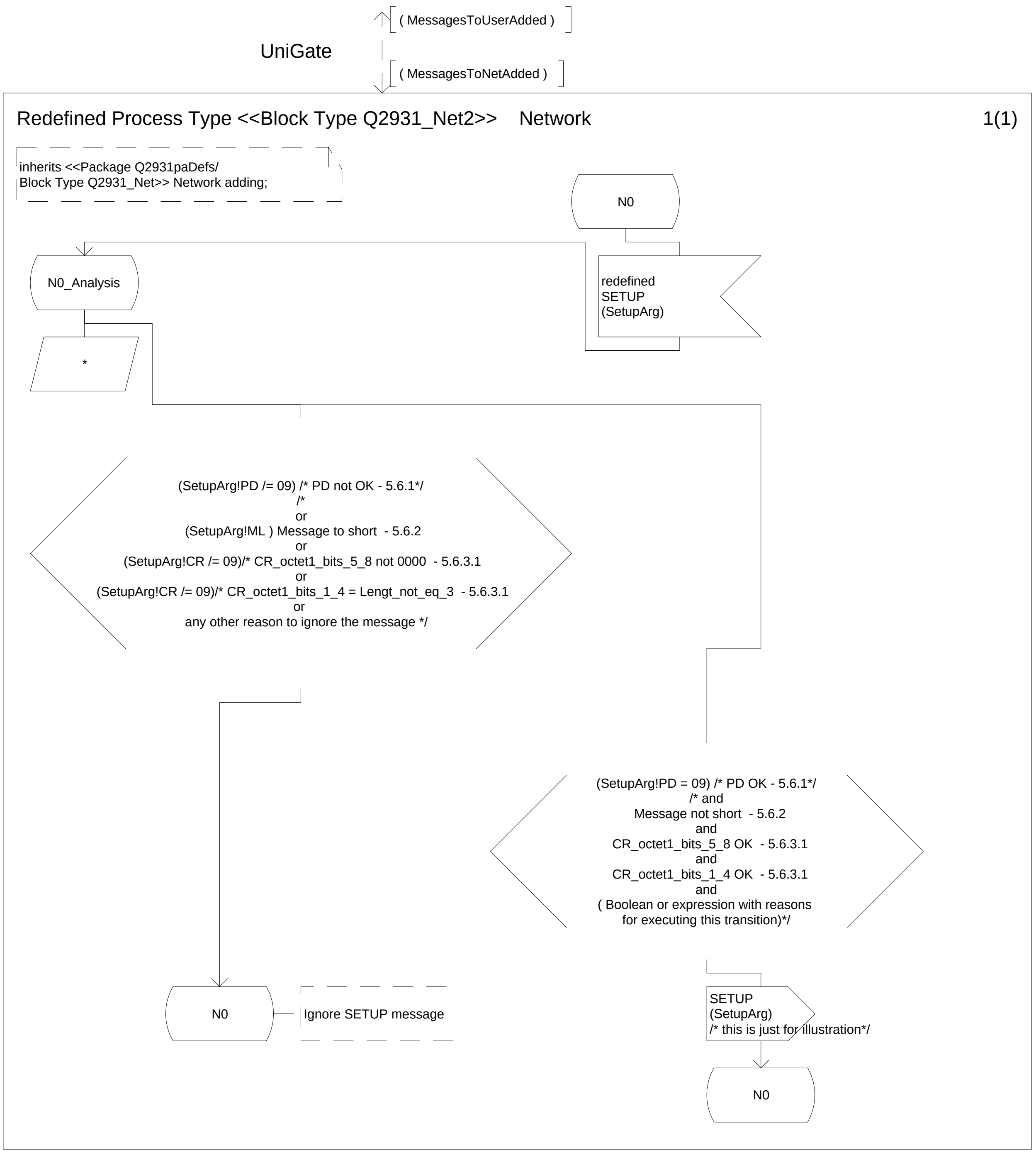

(MessagesToUserAdded)
UniGate
(MessagesToNetAdded)
Redefined Process Type <<Block Type Q2931_Net2>> Network
1(1)
inherits <<Package Q2931paDefs/
Block Type Q2931_Net>> Network adding;
N0
redefined
SETUP
(SetupArg)
N0_Analysis
*
(SetupArg!PD /= 09) /* PD not OK - 5.6.1*/
/*
or
(SetupArg!ML) Message to short - 5.6.2
or
(SetupArg!CR /= 09)/* CR_octet1_bits_5_8 not 0000 - 5.6.3.1
or
(SetupArg!CR /= 09)/* CR_octet1_bits_1_4 = Lengt_not_eq_3 - 5.6.3.1
or
any other reason to ignore the message */
(SetupArg!PD = 09) /* PD OK - 5.6.1*/
/* and
Message not short - 5.6.2
and
CR_octet1_bits_5_8 OK - 5.6.3.1
and
CR_octet1_bits_1_4 OK - 5.6.3.1
and
(Boolean or expression with reasons
for executing this transition)*/
N0
Ignore SETUP message
SETUP
(SetupArg)
/* this is just for illustration*/
N0

Figure 2: Use of continuous signals for signal parameter analysis

In this example immediately upon reception of a SETUP signal a new state is entered where a number of conditions on message parameters are expressed as Boolean expressions in continuous signals. The condition that evaluates to TRUE will trigger the transition that follows. In this manner branching in one place is used instead of a tree of decision statements. This, in spite of possibly complex Boolean expressions, may prove to be far more readable reducing at the same time considerably the size of specifications.

In combination with object orientation this approach is even more powerful. When a process is redefined every transition following a continuous signal with a keyword virtual can be redefined, and new transitions can also be added.

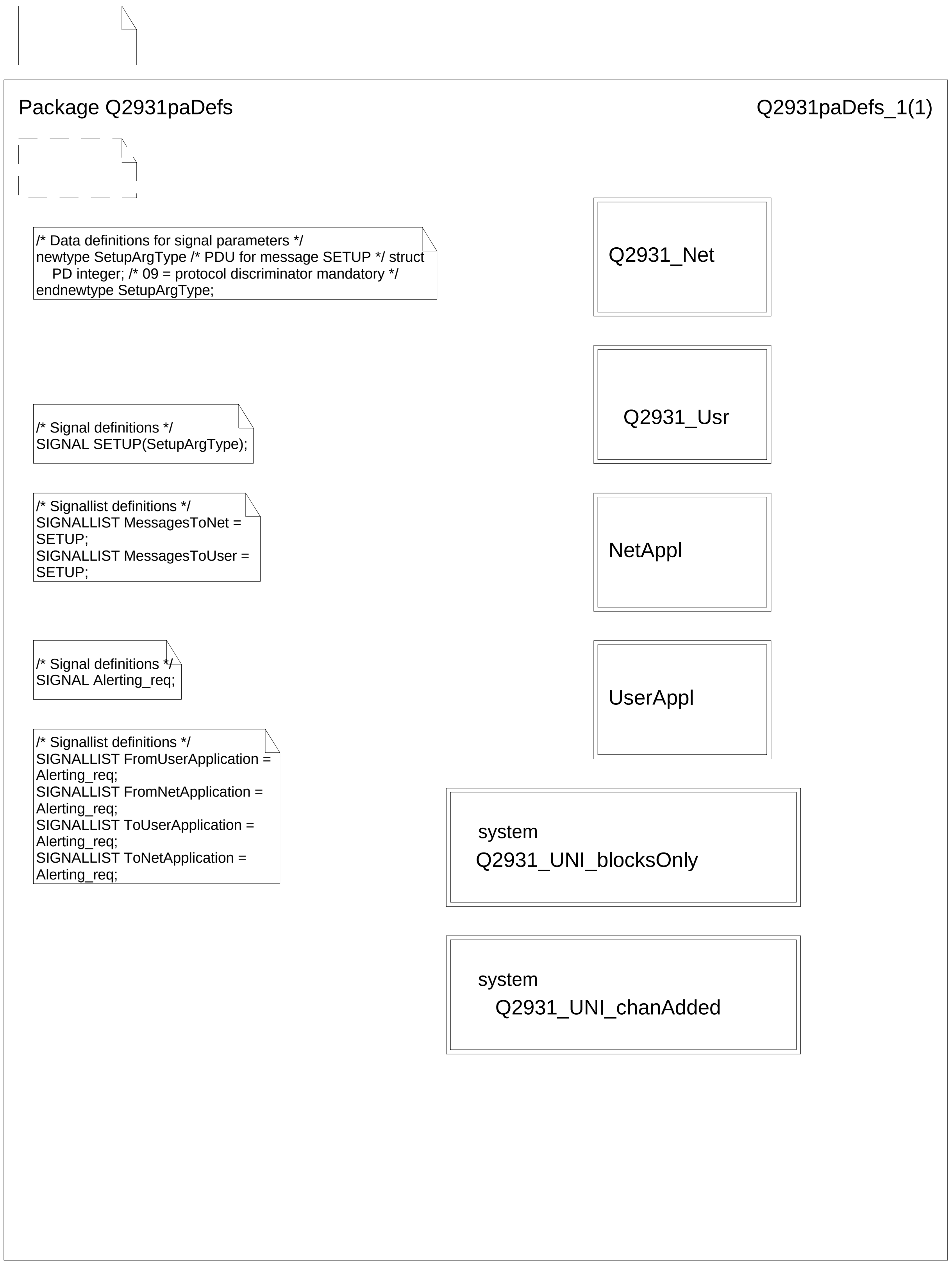
Expressions in continuous signals should be carefully built since at least one transition must be executed and if several can evaluate to TRUE at the same time, priorities may have to be added to them. Note also the SAVE symbol with asterisk, which should hold any pending messages in the input queue until analysis is completed and an appropriate state is entered.

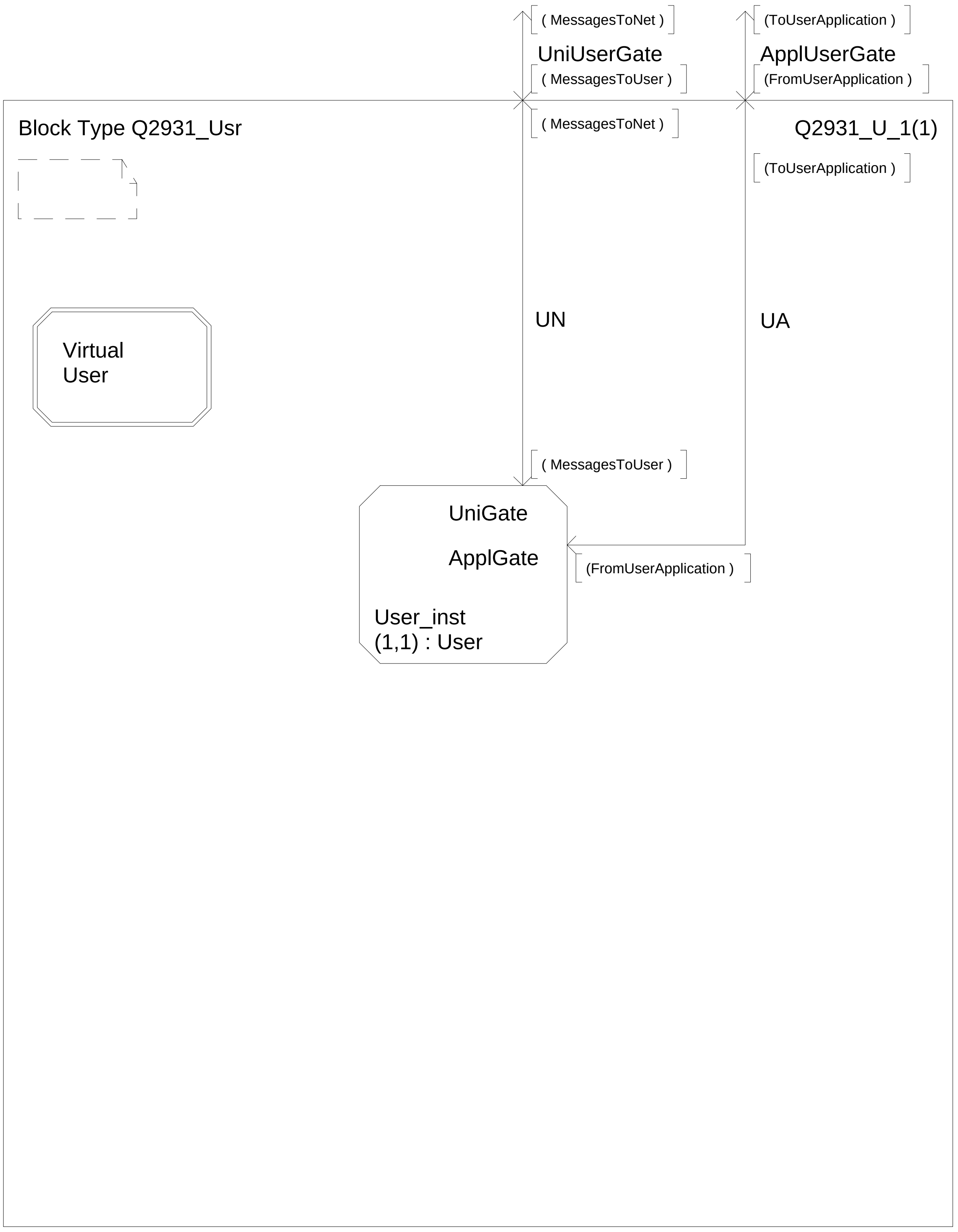
6 Conclusions and further studies

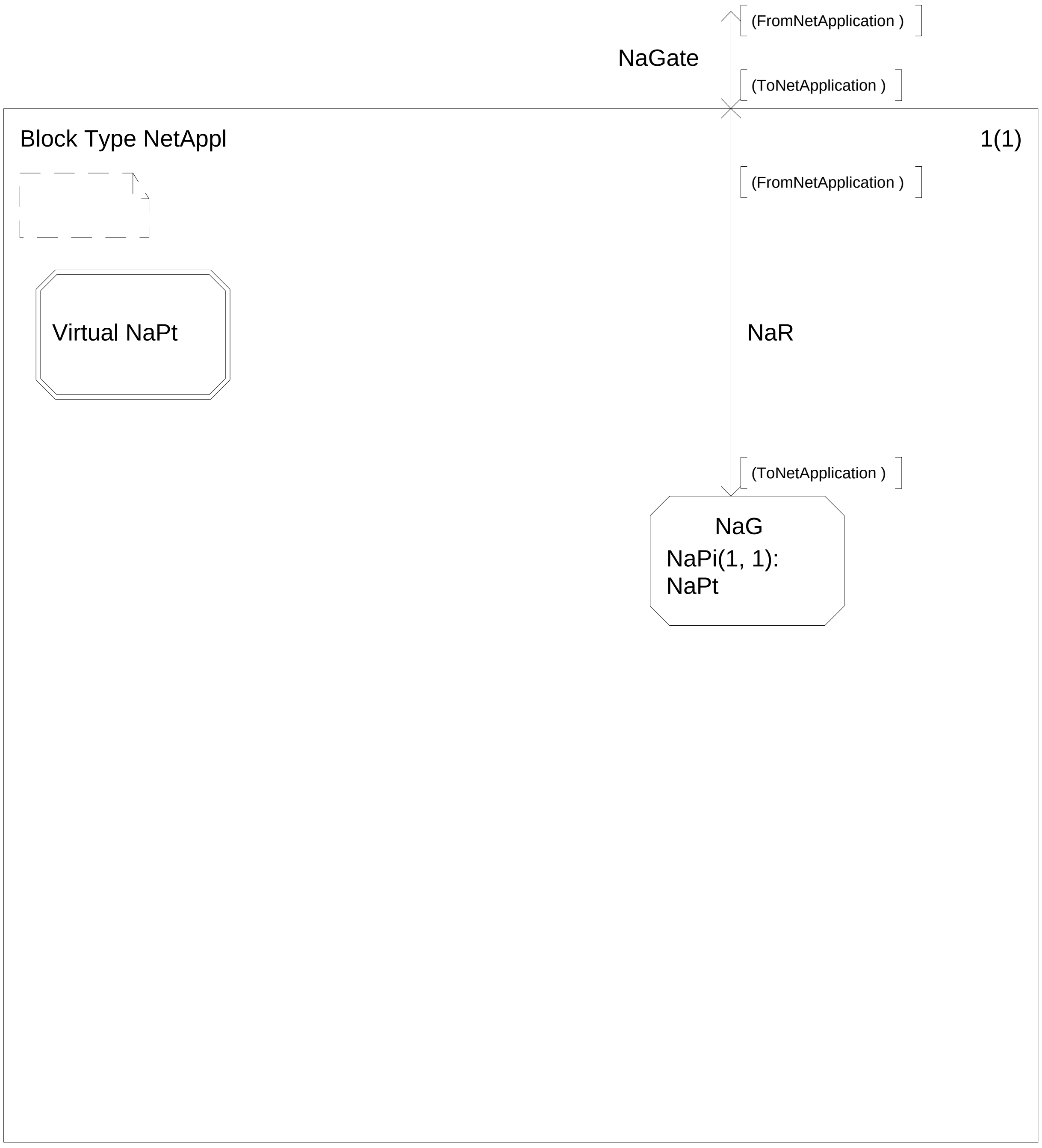
The results of this limited study indicate that re-use of specifications based on the use of object-oriented features can be achieved. In this study some possible goals are identified and some possible solutions are proposed with preliminary evaluation of advantages and drawbacks. At the same time, other goals could also be identified leading to different approaches to re-use.

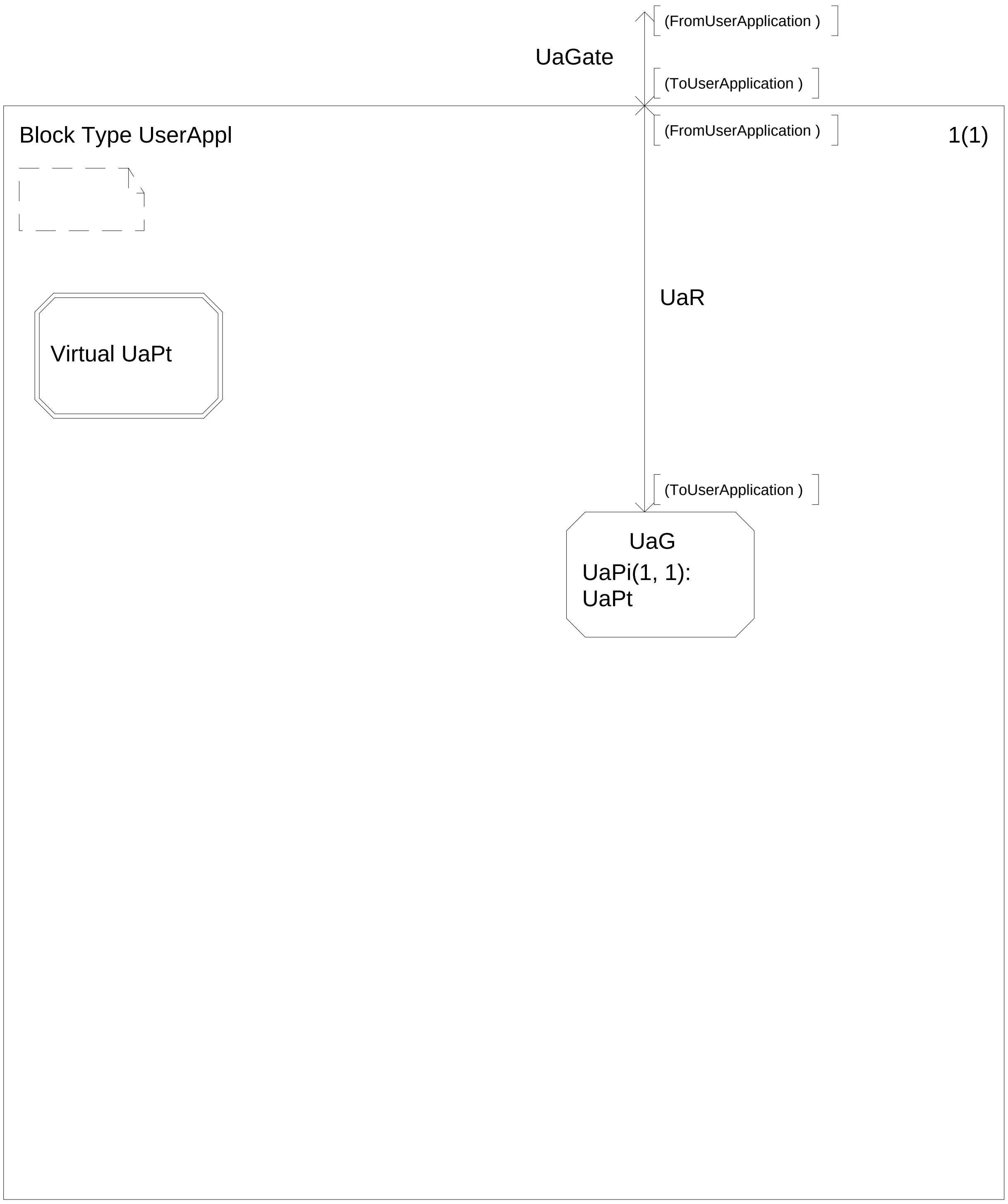
In order to achieve even the minimal level of re-use based on object-orientation ETSI standard developers will need support and guidance. Further studies that would extend the ideas presented in the present document may create a needed framework.

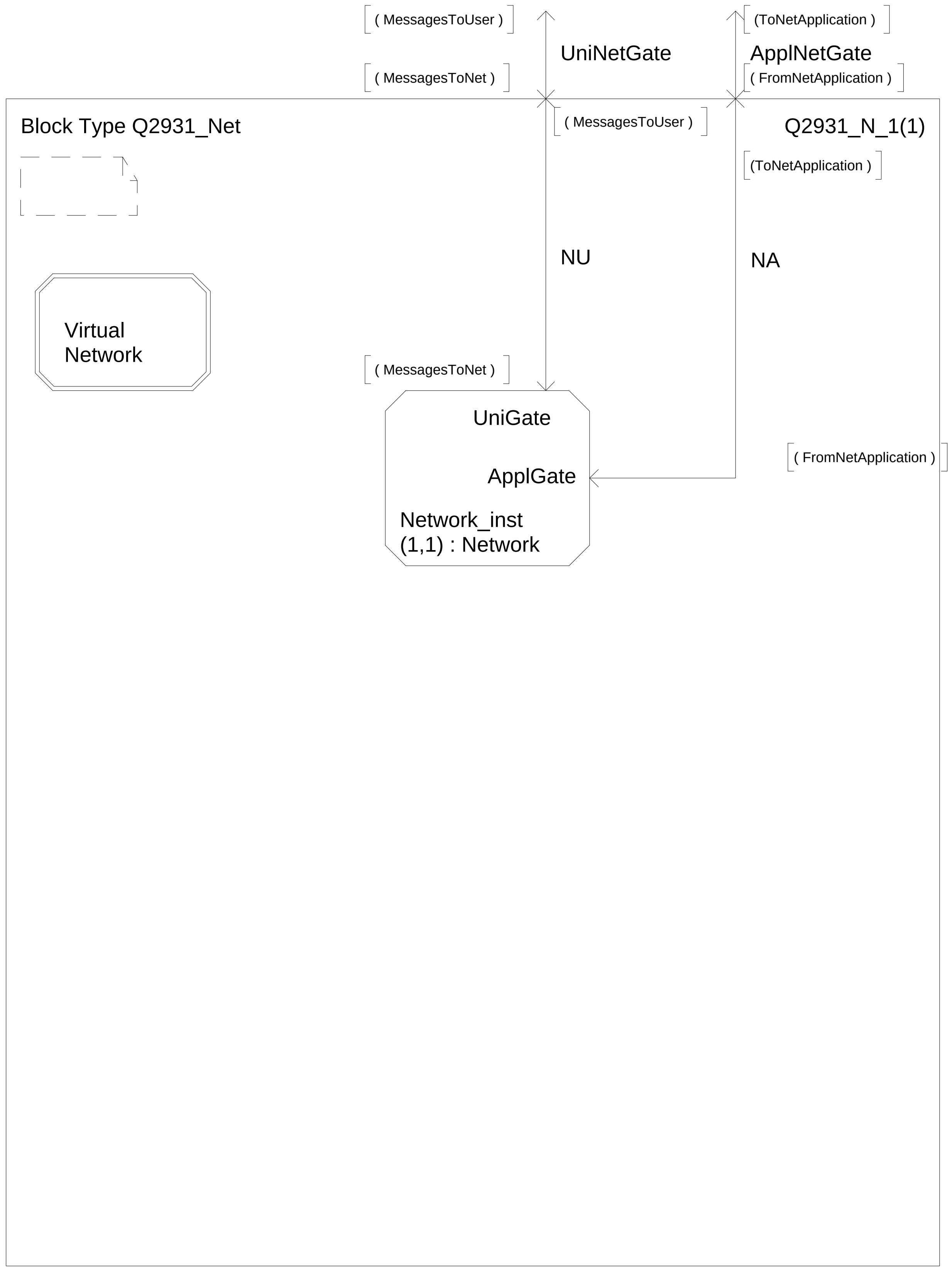
Annex A: SDL diagrams for Example II

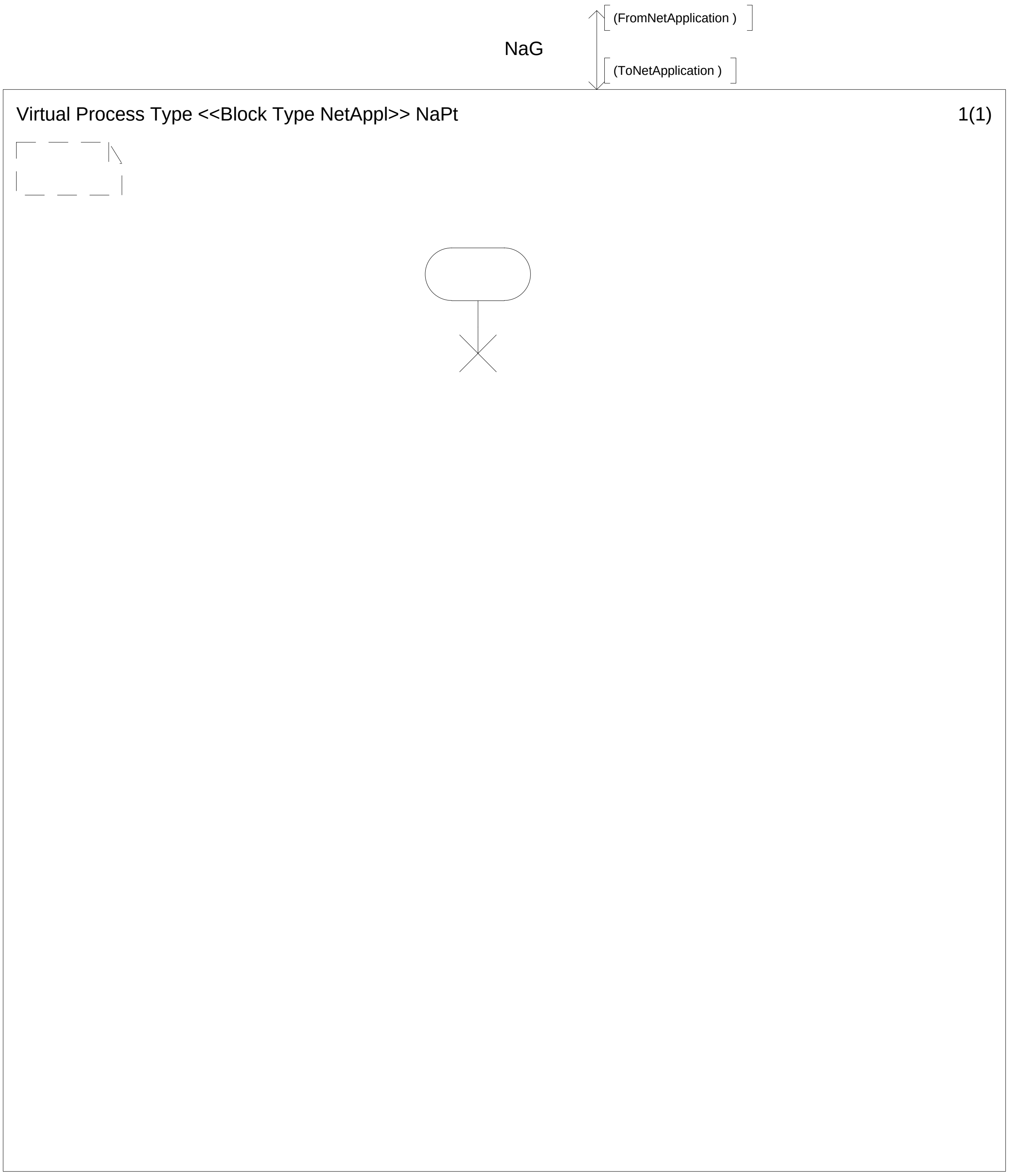

Package Q2931paDefs
Q2931paDefs_1(1)
/* Data definitions for signal parameters */
newtype SetupArgType /* PDU for message SETUP */ struct
PD integer; /* 09 = protocol discriminator mandatory */
endnewtype SetupArgType;
/* Signal definitions */
SIGNAL SETUP(SetupArgType);
/* Signallist definitions */
SIGNALLIST MessagesToNet = SETUP;
SIGNALLIST MessagesToUser = SETUP;
/* Signal definitions */
SIGNAL Alerting_req;
/* Signallist definitions */
SIGNALLIST FromUserApplication = Alerting_req;
SIGNALLIST FromNetApplication = Alerting_req;
SIGNALLIST ToUserApplication = Alerting_req;
SIGNALLIST ToNetApplication = Alerting_req;
Q2931_Net
Q2931_Usr
NetAppl
UserAppl
system Q2931_UNI_blocksOnly
system Q2931_UNI_chanAdded

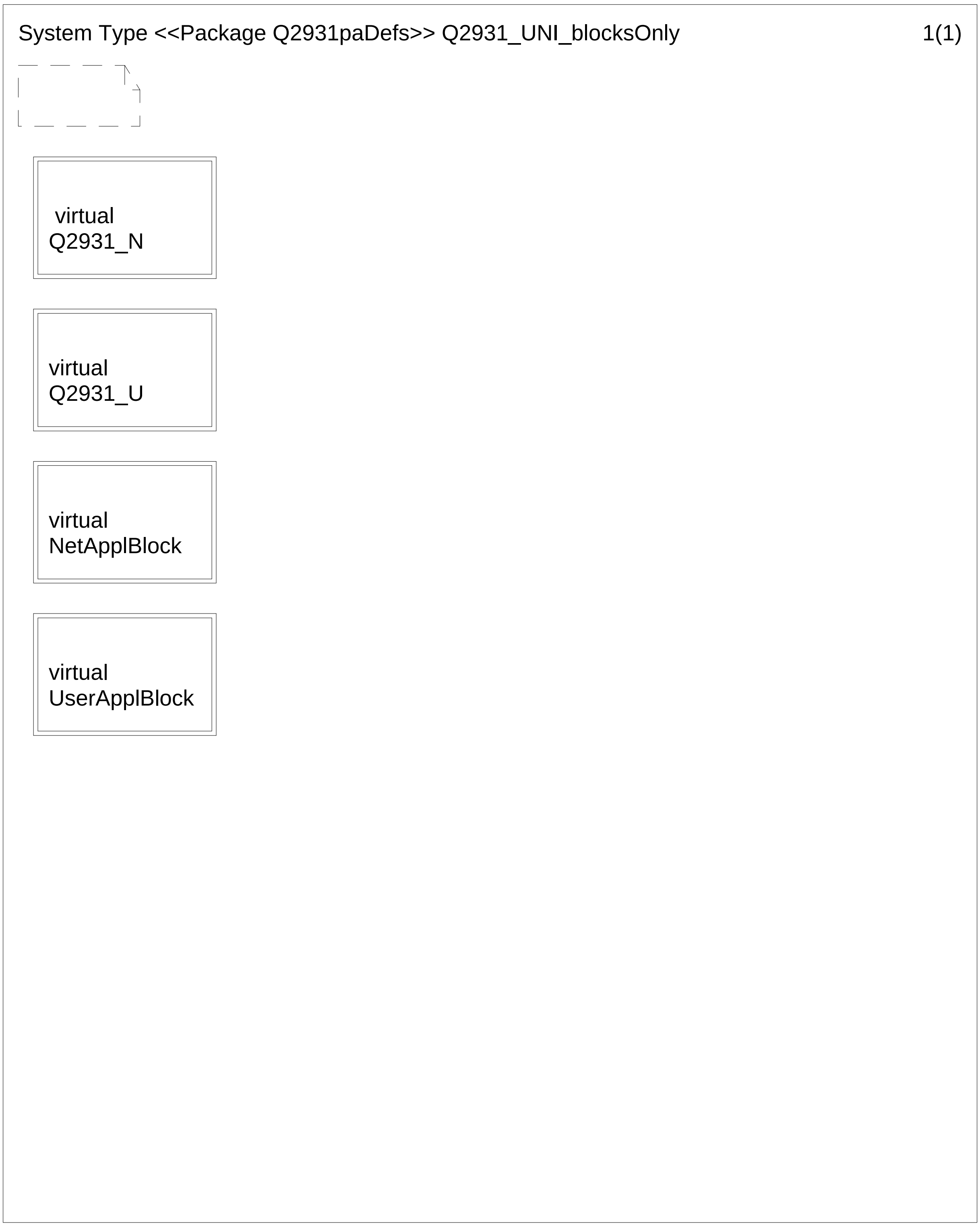

(MessagesToNet)
UniUserGate
(MessagesToUser)
(ToUserApplication)
ApplUserGate
(FromUserApplication)
Block Type Q2931_Usr
Q2931_U_1(1)
(MessagesToNet)
(ToUserApplication)
Virtual
User
UN
UA
(MessagesToUser)
UniGate
ApplGate
(FromUserApplication)
User_inst
(1,1) : User

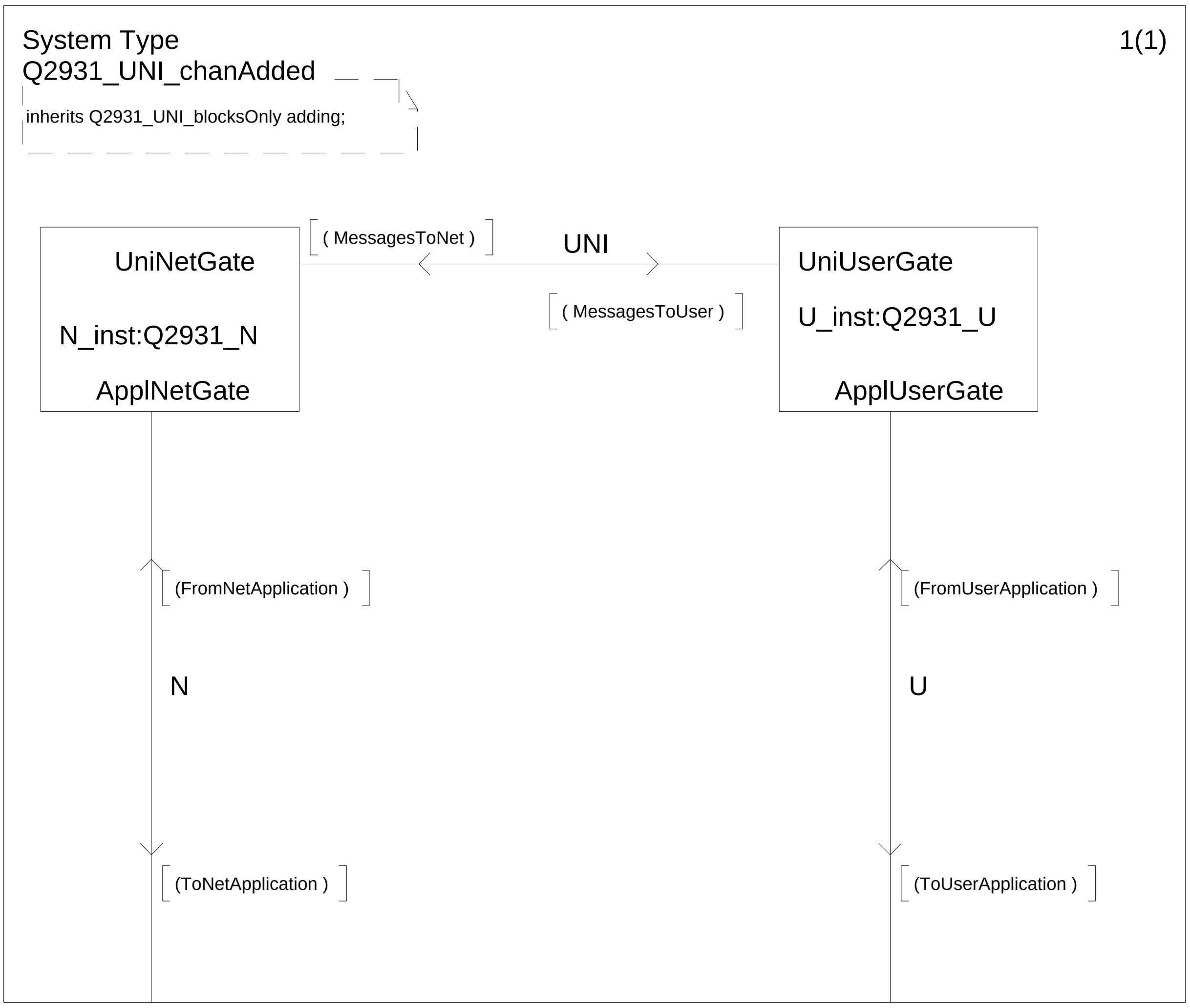

(FromNetApplication)
NaGate
(ToNetApplication)
Block Type NetAppl
1(1)
(FromNetApplication)
Virtual NaPt
NaR
(ToNetApplication)
NaG
NaPi(1, 1):
NaPt

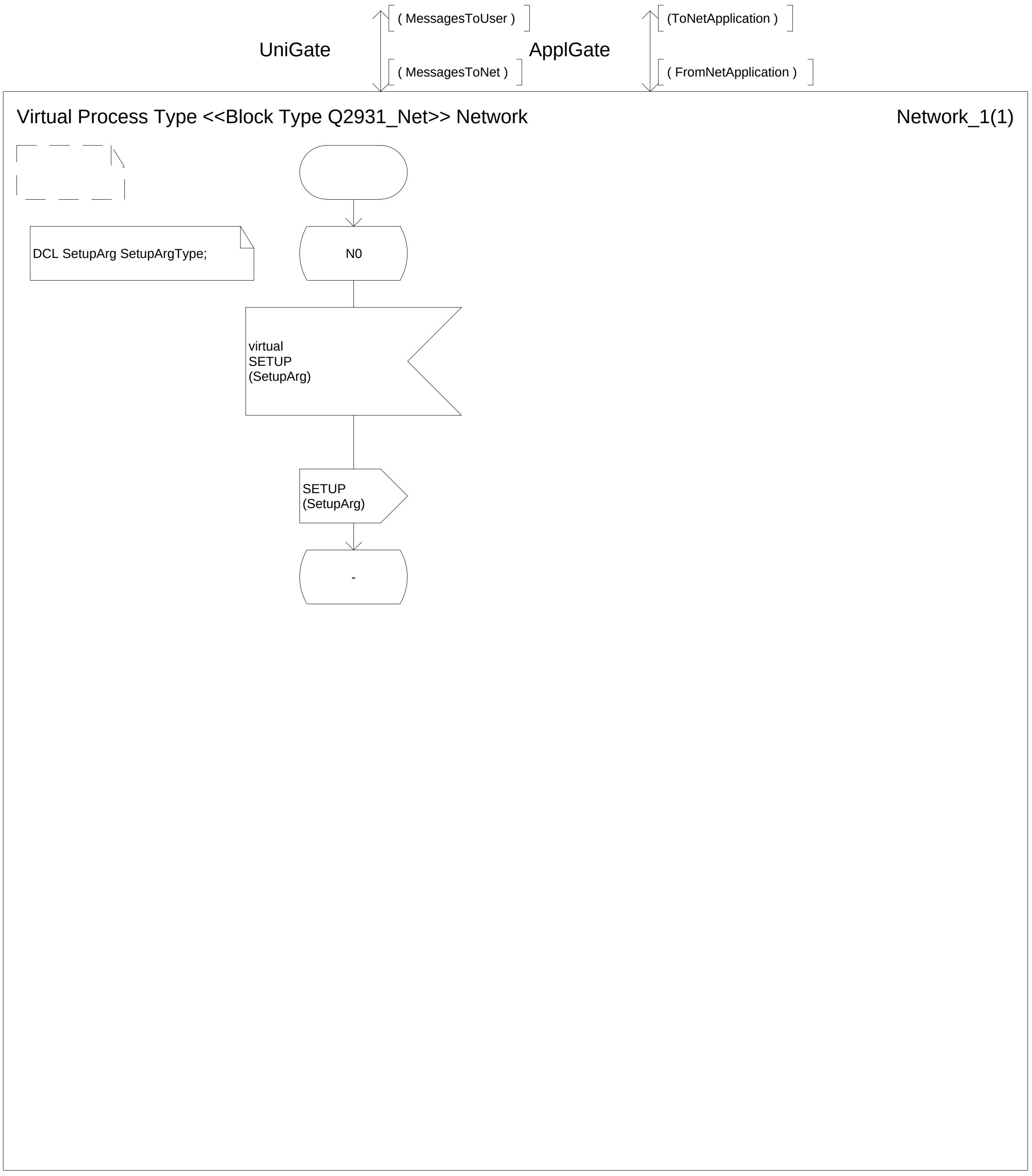

(FromUserApplication)
UaGate
(ToUserApplication)
Block Type UserAppl
(FromUserApplication)
1(1)
Virtual UaPt
UaR
(ToUserApplication)
UaG
UaPi(1, 1):
UaPt

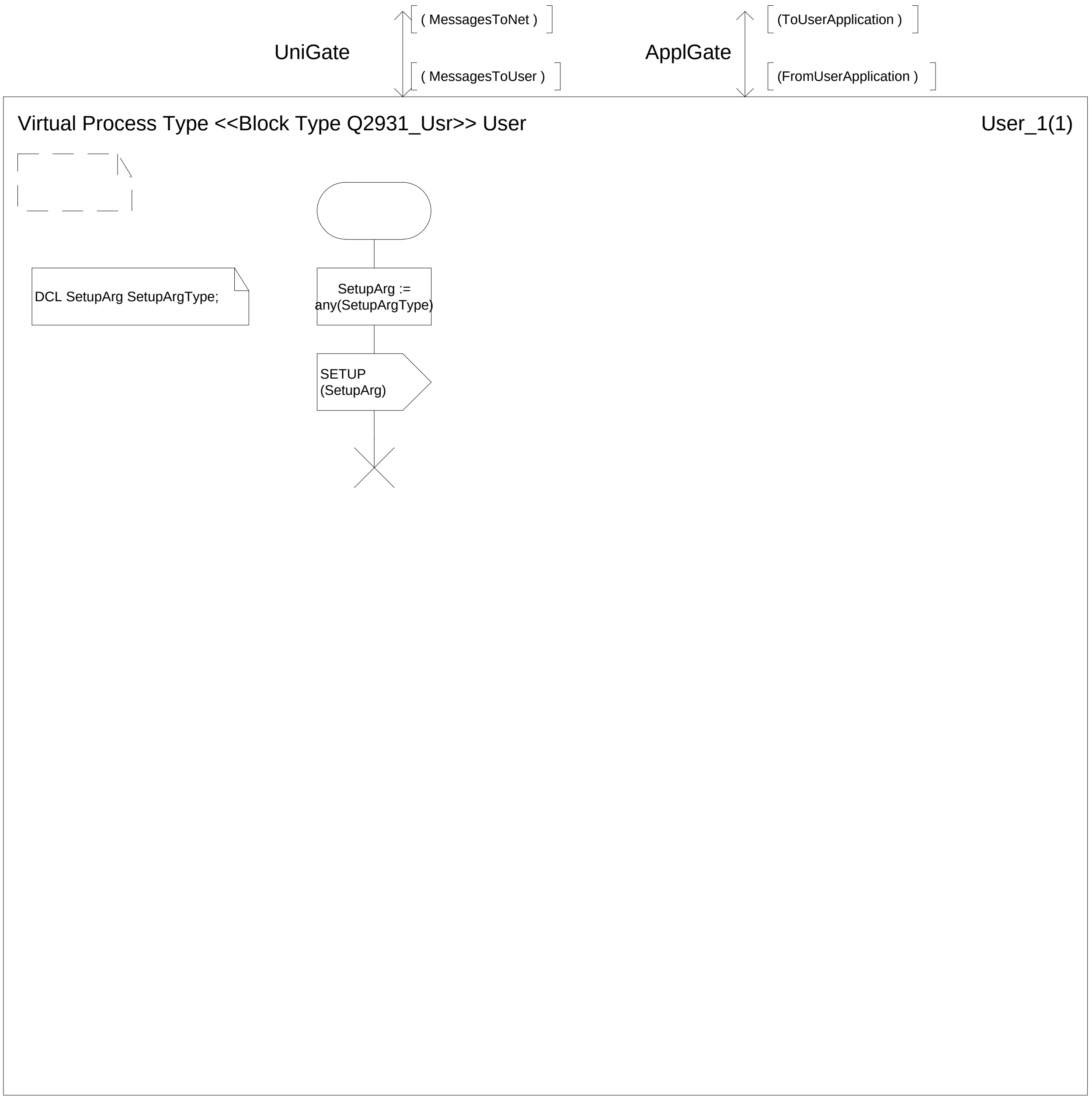

(MessagesToUser)
UniNetGate
(MessagesToNet)
(ToNetApplication)
ApplNetGate
(FromNetApplication)
Block Type Q2931_Net
Q2931_N_1(1)
(MessagesToUser)
(ToNetApplication)
NU
NA
Virtual Network
(MessagesToNet)
UniGate
ApplGate
(FromNetApplication)
Network_inst (1,1) : Network

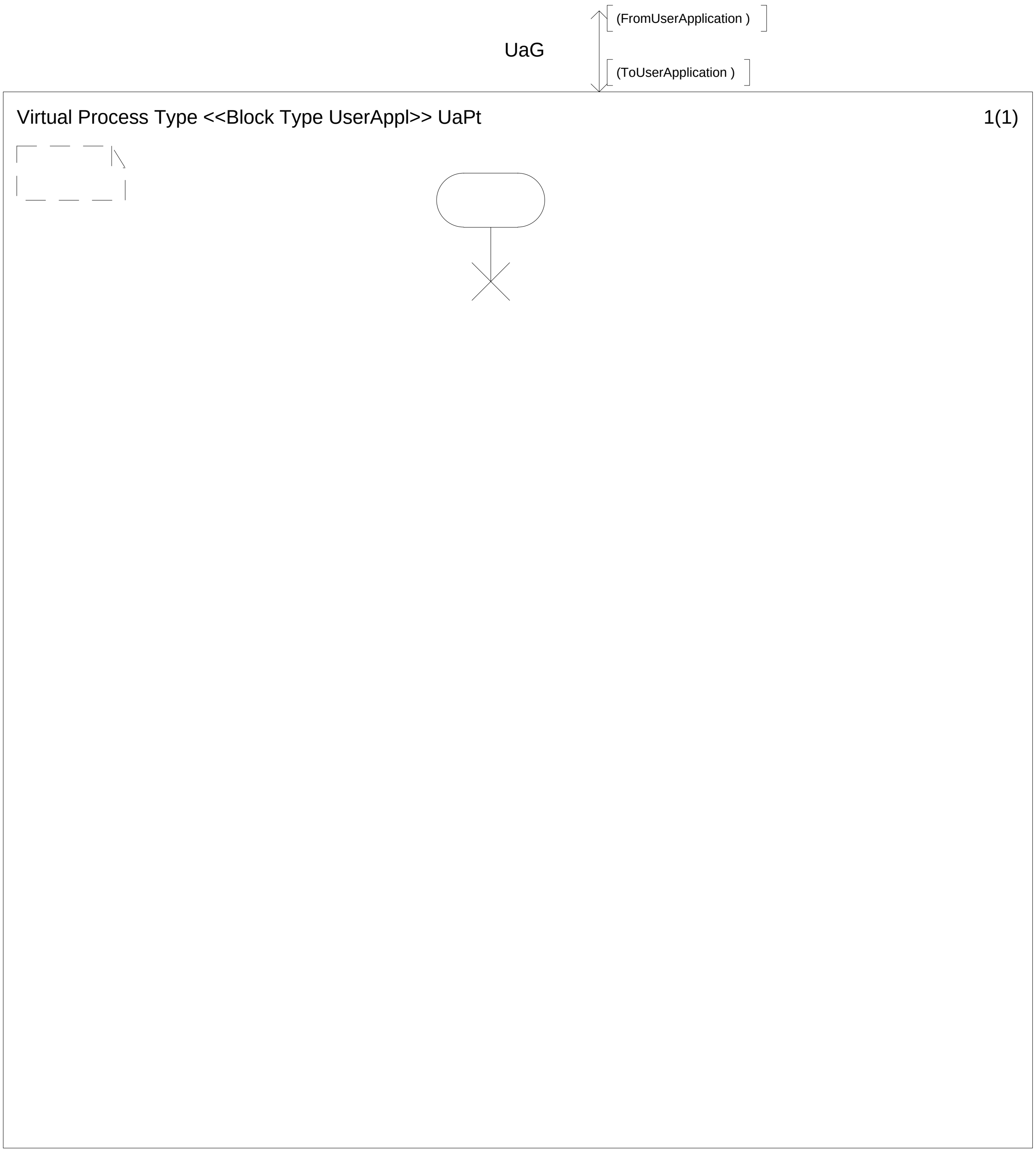

(FromNetApplication)
NaG
(ToNetApplication)
Virtual Process Type <<Block Type NetAppl>> NaPt
1(1)

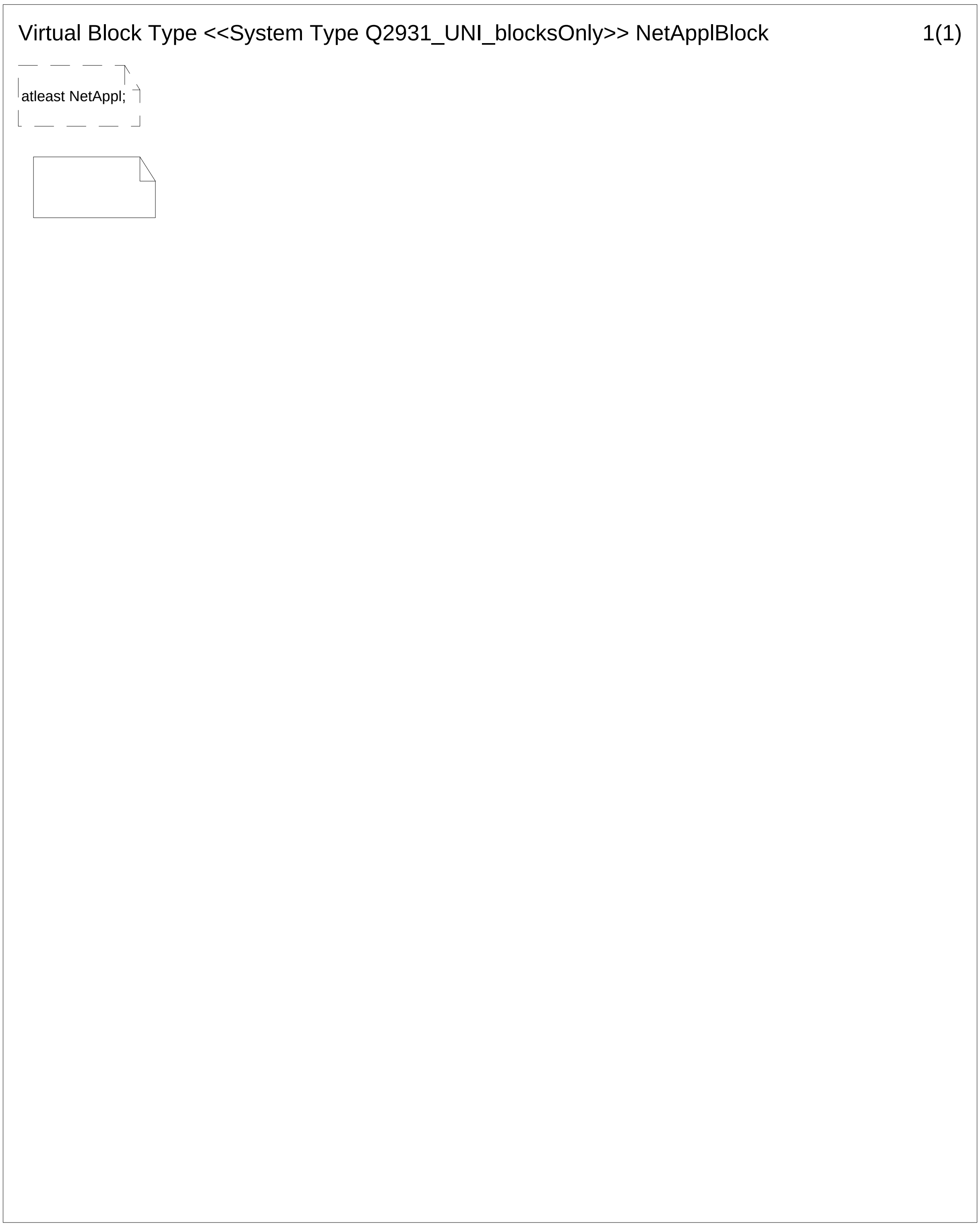

System Type <<Package Q2931paDefs>> Q2931_UNI_blocksOnly
1(1)
virtual Q2931_N
virtual Q2931_U
virtual NetApplBlock
virtual UserApplBlock

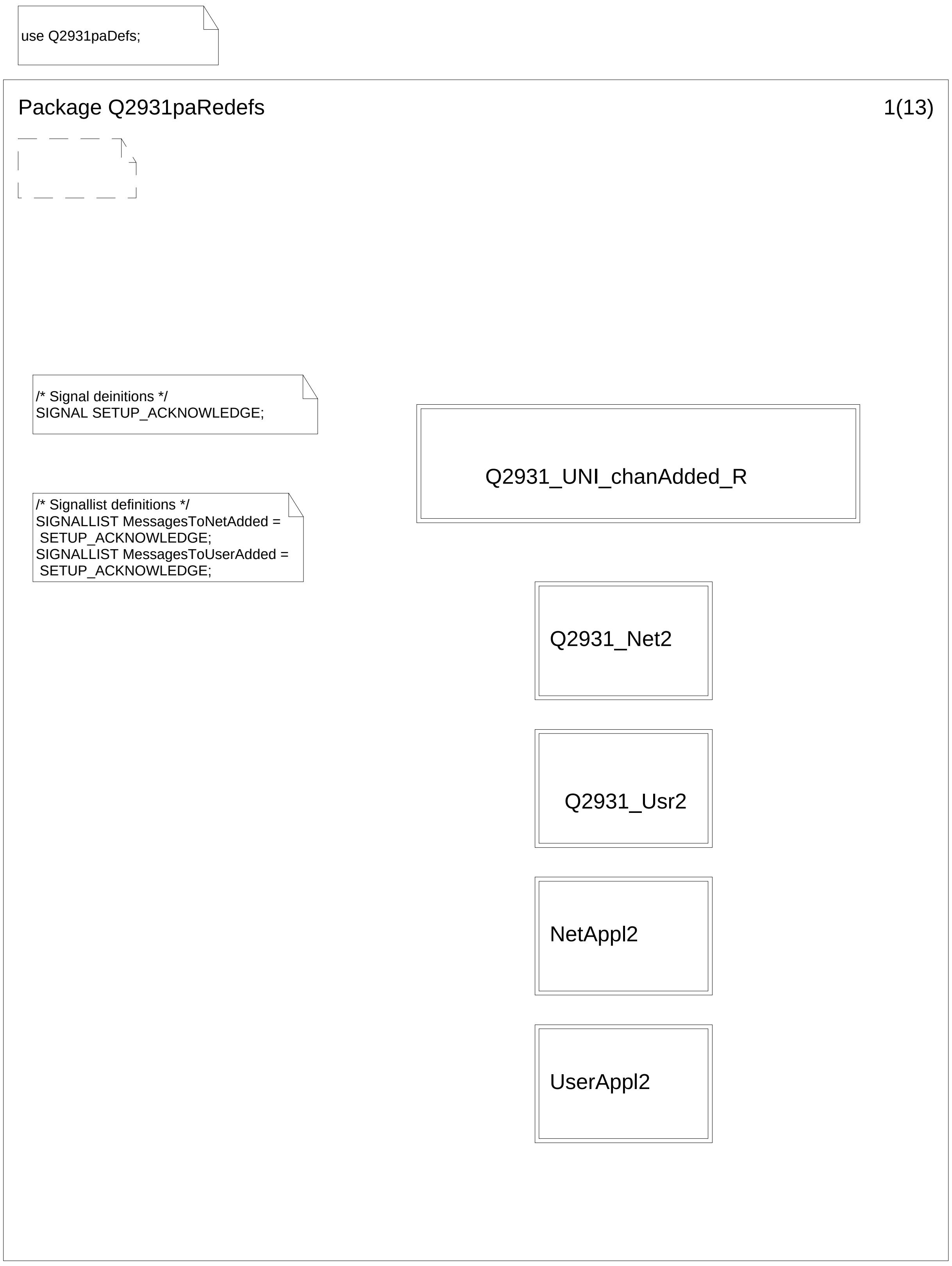

System Type
Q2931_UNI_chanAdded
1(1)
inherits Q2931_UNI_blocksOnly adding;
UniNetGate
N_inst:Q2931_N
ApplNetGate
(MessagesToNet)
UNI
(MessagesToUser)
UniUserGate
U_inst:Q2931_U
ApplUserGate
(FromNetApplication)
N
(ToNetApplication)
(FromUserApplication)
U
(ToUserApplication)


(MessagesToUser)
UniGate
(MessagesToNet)
(ToNetApplication)
ApplGate
(FromNetApplication)
Virtual Process Type <<Block Type Q2931_Net>> Network
Network_1(1)
DCL SetupArg SetupArgType;
N0
virtual
SETUP
(SetupArg)
SETUP
(SetupArg)
-


(MessagesToNet)
UniGate
(MessagesToUser)
(ToUserApplication)
ApplGate
(FromUserApplication)
Virtual Process Type <<Block Type Q2931_Usr>> User
User_1(1)
DCL SetupArg SetupArgType;
SetupArg :=
any(SetupArgType)
SETUP
(SetupArg)


(FromUserApplication)
UaG
(ToUserApplication)
Virtual Process Type <<Block Type UserAppl>> UaPt
1(1)


Virtual Block Type <<System Type Q2931_UNI_blocksOnly>> NetApplBlock
1(1)
atleast NetAppl;

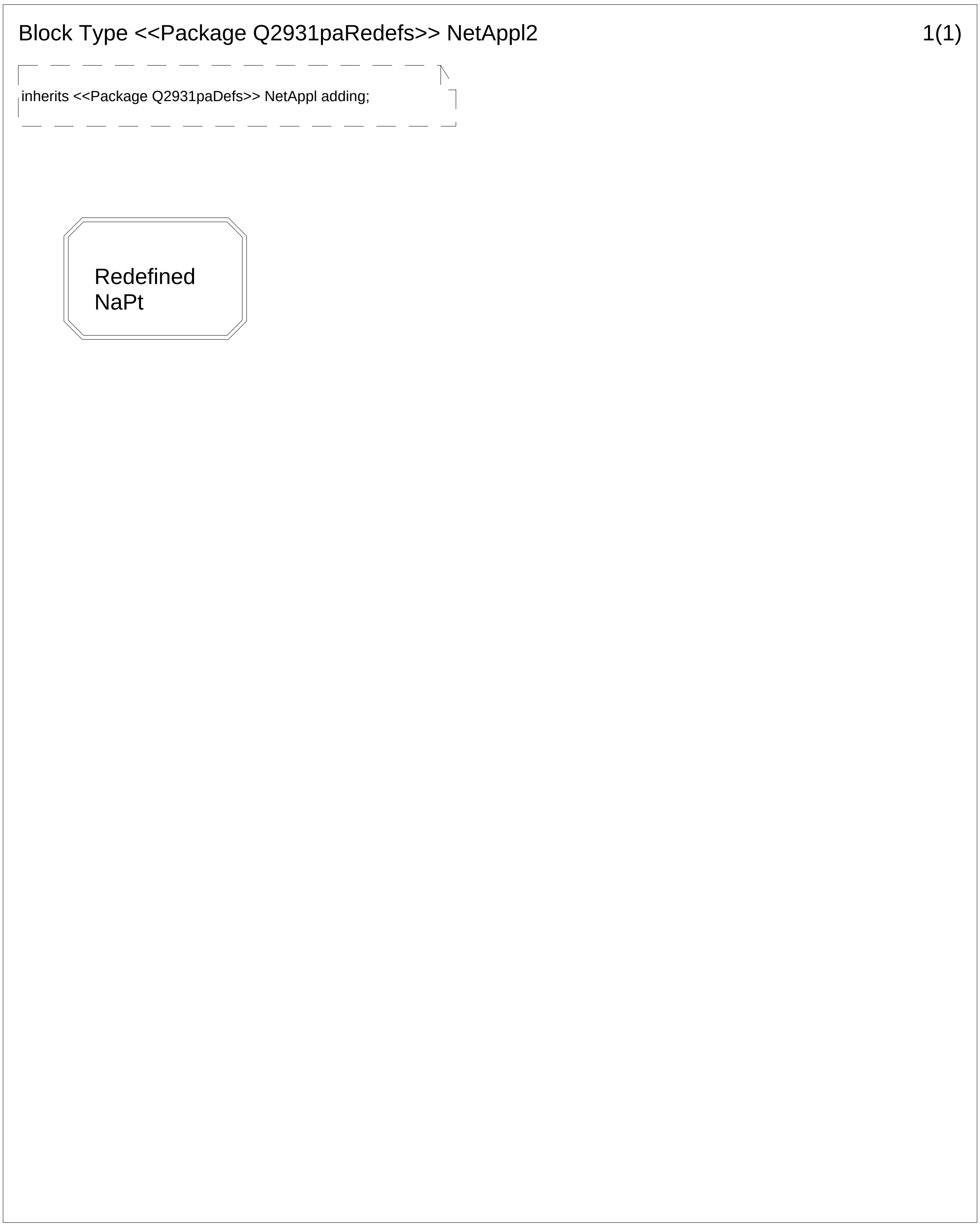


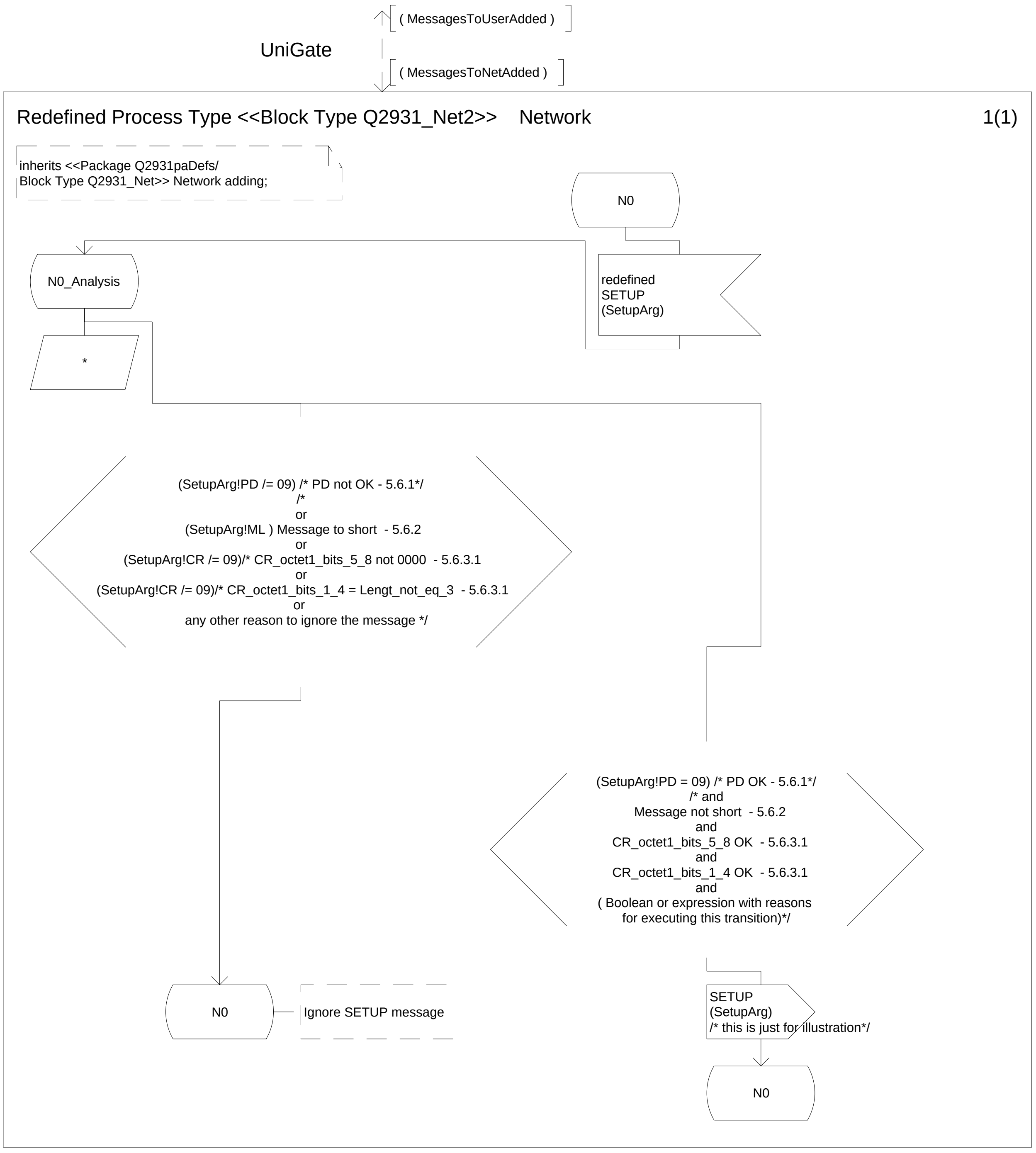
```
use Q2931paDefs;
```

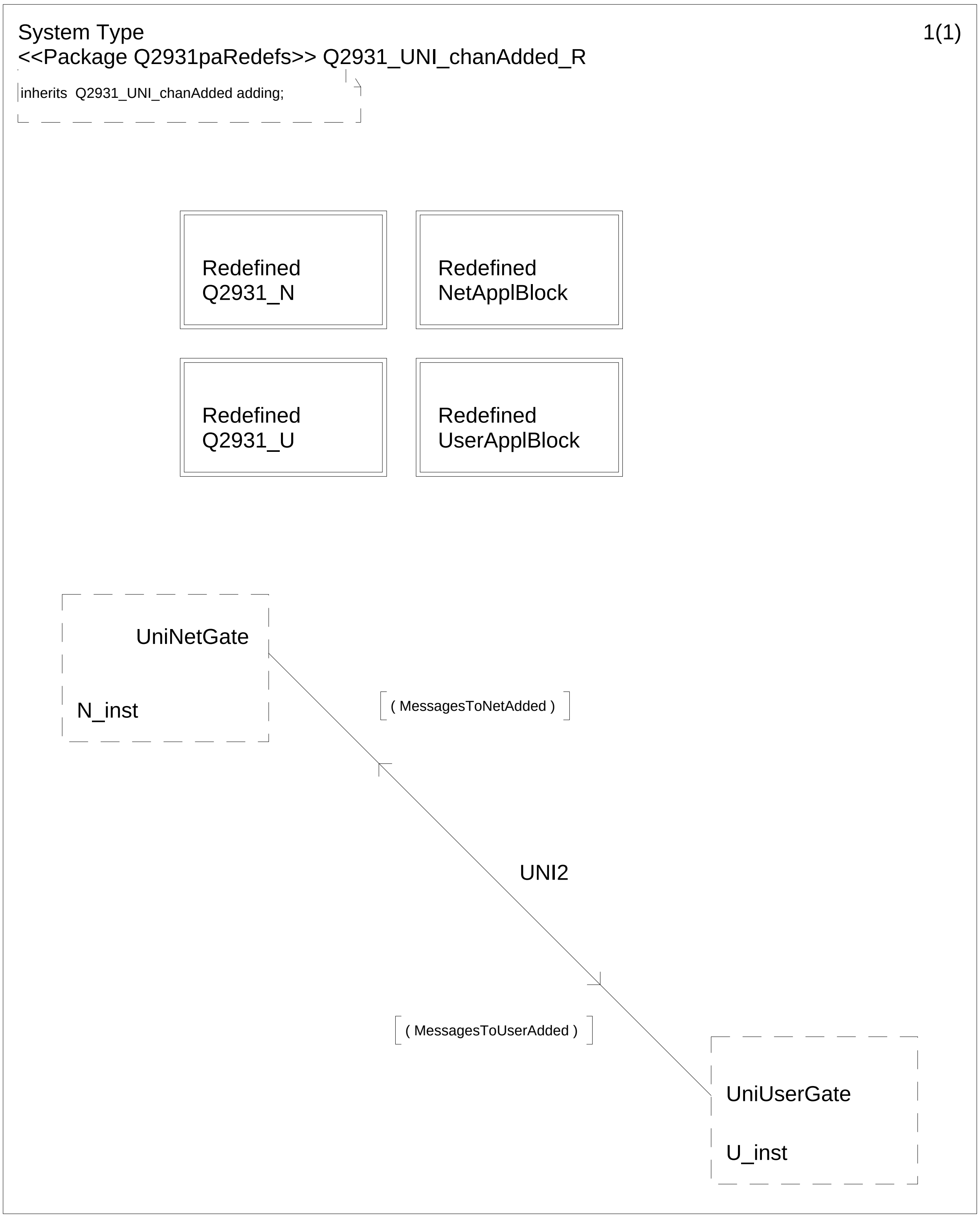
Package Q2931paRedefs 1(13)

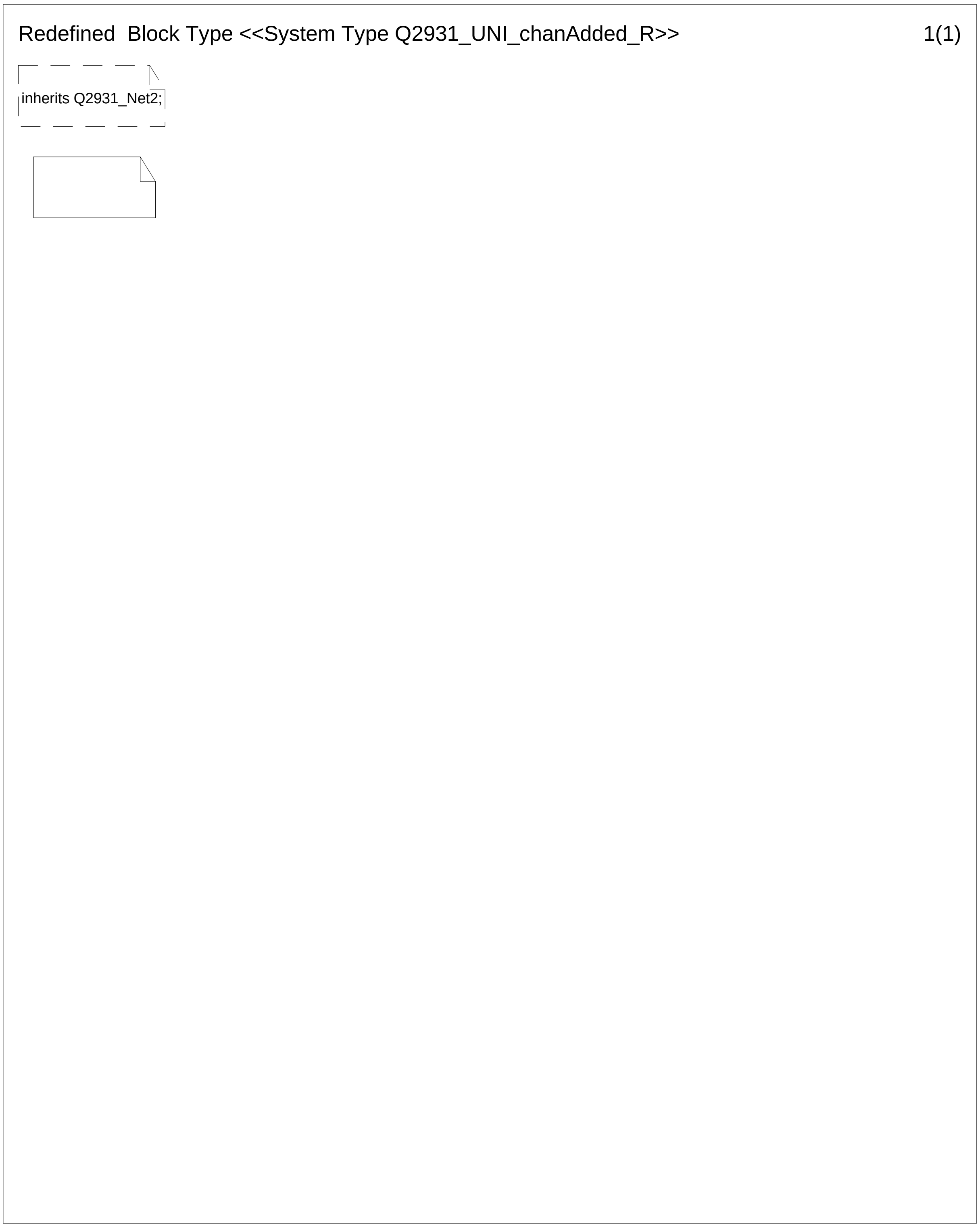
```
/* Signal deinitions */
SIGNAL SETUP_ACKNOWLEDGE;
```

```
/* Signallist definitions */
SIGNALLIST MessagesToNetAdded =
 SETUP_ACKNOWLEDGE;
SIGNALLIST MessagesToUserAdded =
 SETUP_ACKNOWLEDGE;
```


Block Type <<Package Q2931paRedefs>> NetAppl2
1(1)
inherits <<Package Q2931paDefs>> NetAppl adding;
Redefined
NaPt


(MessagesToUserAdded)
UniGate
(MessagesToNetAdded)
Redefined Process Type <<Block Type Q2931_Net2>> Network
1(1)
inherits <<Package Q2931paDefs/
Block Type Q2931_Net>> Network adding;
N0
redefined
SETUP
(SetupArg)
N0_Analysis
*
(SetupArg!PD /= 09) /* PD not OK - 5.6.1*/
/*
or
(SetupArg!ML) Message to short - 5.6.2
or
(SetupArg!CR /= 09)/* CR_octet1_bits_5_8 not 0000 - 5.6.3.1
or
(SetupArg!CR /= 09)/* CR_octet1_bits_1_4 = Lengt_not_eq_3 - 5.6.3.1
or
any other reason to ignore the message */
(SetupArg!PD = 09) /* PD OK - 5.6.1*/
/* and
Message not short - 5.6.2
and
CR_octet1_bits_5_8 OK - 5.6.3.1
and
CR_octet1_bits_1_4 OK - 5.6.3.1
and
(Boolean or expression with reasons
for executing this transition)*/
N0
Ignore SETUP message
SETUP
(SetupArg)
/* this is just for illustration*/
N0


System Type
<<Package Q2931paRedefs>> Q2931_UNI_chanAdded_R
1(1)
inherits Q2931_UNI_chanAdded adding;
Redefined Q2931_N
Redefined NetApplBlock
Redefined Q2931_U
Redefined UserApplBlock
UniNetGate
N_inst
(MessagesToNetAdded)
UNI2
(MessagesToUserAdded)
UniUserGate
U_inst


Redefined Block Type <<System Type Q2931_UNI_chanAdded_R>>
1(1)
inherits Q2931_Net2;

History

Document history		
V1.1.1	April 1997	Publication

ISBN 2-7437-1291-0
Dépôt légal : Avril 1997