1. DSS2

Author: Timo Pärnänen

There are two possible trends to implement ATM signalling protocols: according to specifications by ATM Forum or by ITU-T. During the first year of TOVE project, an user-network interface (UNI) layer 3 was implemented according to ITU-T specifications. In the future, also network-network interface by ITU-T and some specifications of ATM Forum will be implemented.

1.1 1 Introduction

This implementation, Digital Signalling Subscriber No.2 (DSS2), is based on ITU-T standard Q.2931. It specifies the user-network interface for basic call/connection control in Broadband Integrated Services Digital Network (B-ISDN). Part of the specification was implemented in the call control module (see the call control documentation).

2 Architecture

Q.2931 specification consist of four modules: co-ordination, reset-restart, reset-response and actual Q.2931 signalling protocol. In the first release, the co-ordination and Q.2931 module were implemented. Both of these are placed in the same programming module (in CVS, Concurrent Versions System) called dss2. The dss2 module includes following classes:

- dss2Protocol
- dss2CoOrdProtocol
- dss2State and inherited classes for all actual states in DSS2 state machine
- dss2CoOrdBaseState and four inherited classes for actual states in co-ordination state machine
- dss2Pdus and inherited classes for each PDUs
- eight primitive classes for requesting and indicating link status
- dss2Accessor joined to the mux to get a call reference value from incoming PDUs

Figure 1 shows relations between objects.

From the switch's point of view there are two sides of the DSS2 protocol, originating and terminating sides. The former one receives the SETUP message from down direction or from the IN architecture and starts call, the latter one sends the message to the destination (to the next switch or user) equipment. An appendix X has signalling diagram examples of call/connection establishment and clearing.

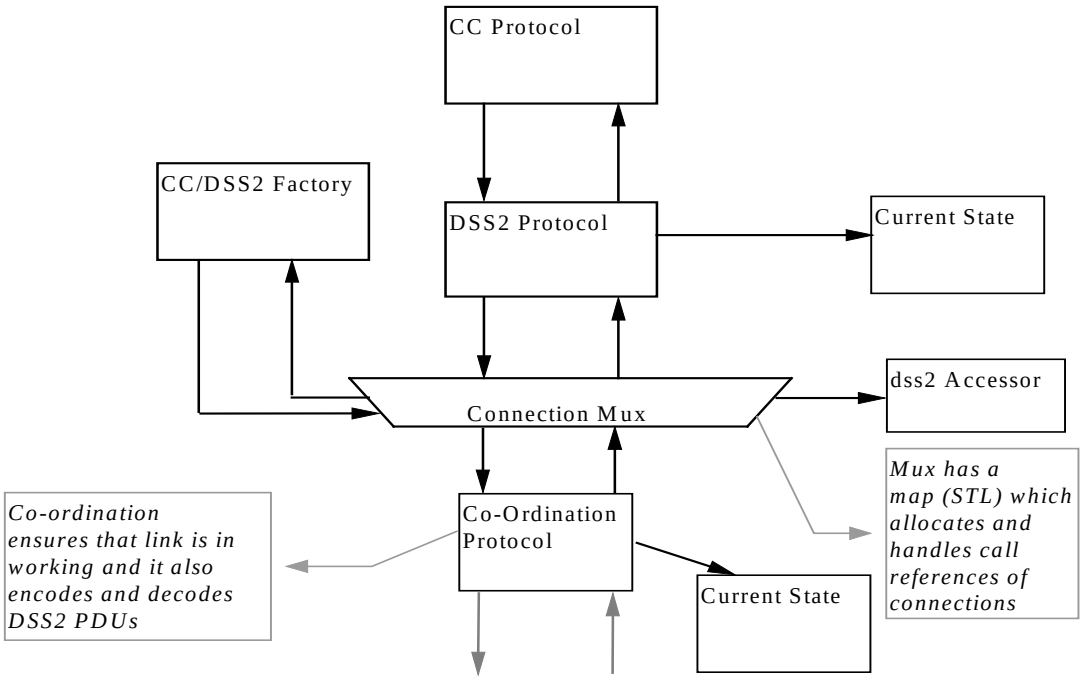

Figure 1 Relations between objects

1.2 3 Implementation details

Following chapters describe some special issues related to the implementation of the DSS2 protocol, states and messages. Because implementation is based on the OVOPS++ architecture, common implementation solutions are not documented here. These can be found from the OVOPS++ documentation.

1.2.1 3.1 Protocols and states

The DSS2 protocol accepts primitives from a call control via sigiface, correspondingly the co-ordination protocol accepts primitives from lower uni-sscf layer via uaaliface. Through the local "interface" between the DSS2 and co-ordination protocol, PDUs and few link control primitives are transferred.

The DSS2 and co-ordination protocol class includes all protocol variables and a pointer to the current state. The protocol has methods (interface) to send primitives and PDUs to upward and downward. The purpose of the co-ordination protocol is to ensure that link is up. It also decodes PDUs from incoming lower layer primitives and correspondingly encodes PDUs to outgoing primitives to down direction.

States are implemented in a bit different way like in other protocols of the TOVE project. In fact differences are in state changing methods. The reason why state changing is different is that we want use same states in many signalling protocols. Now it is used only in the Q.2931 network and user side, but in the future these same states will be used for UNI 3.1/4.0. The DSS2 network and user side have very few differences between

themselves. User side states U0, U6, U9 for example correspond to network states N0, N1, N3. The state changing from outgoingCallProceeding (N3) state to the active state is different in the user side (U9 → U8). There is one state more in the user side in this situation. Figure 2 shows that change from callInitiated state to outgoingCallProceeding state is equal in both sides, but change from outgoingCallProceeding to the active state differs.

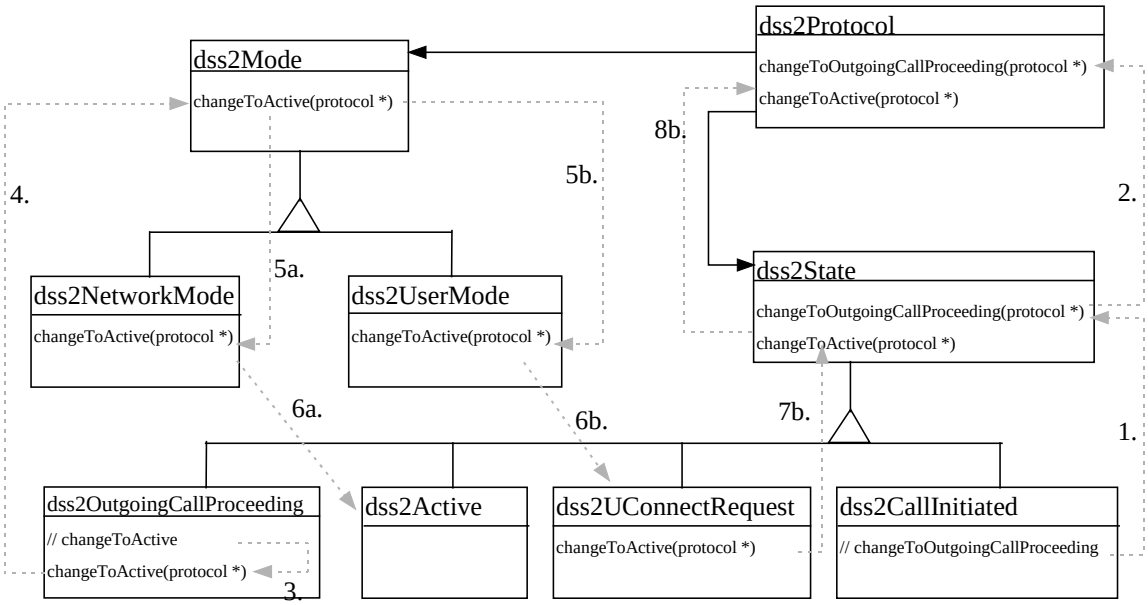

Figure 2 State changing

A following list describes stages shown in the figure 2.

1. changeToOutgoingCallProceeding method is called from dss2 base state
2. the method call is delegated to the protocol
3. changeToActive method is called from outgoingCallProceeding itself (rewritten method)
4. the method call is delegated to the mode class instead of the protocol
5. the method call is directed to the actual inherited mode class

6.a. in the network side state is changed direct to the active state

6.b. in the user side state is changed to the connectRequest state

7-8. state change from connectRequest state to the active state corresponds steps 1 and 2

In the outgoingCallProceeding state, changeToActive method has rewritten, which doesn't call the real state changing method from the protocol but from the mode object.

States are implemented using a singleton pattern as normally. A figure 3 shows state hierarchy and relations between DSS2 and co-ordination states.

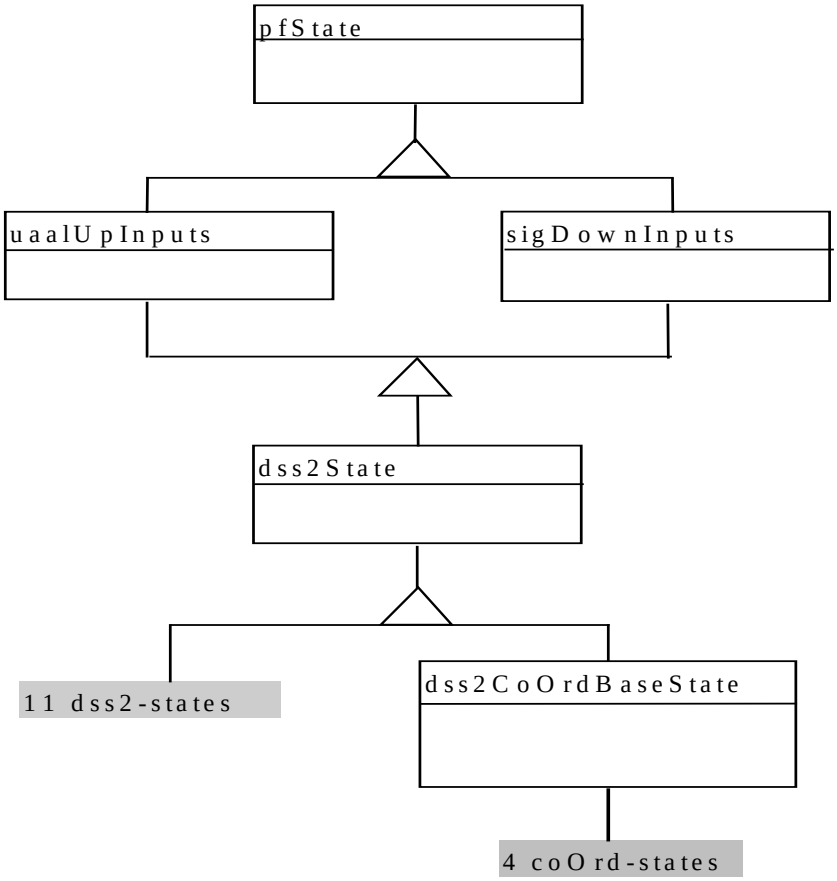

Figure 3 The state hierarchy**.**

The dss2State is a base state for the protocol state machine. It is derived virtually from uaalUpInputs and sigDownInputs whose are abstract base classes defined in upper and lower interfaces. These abstract classes inherit state machine features of the framework from the pfState base class. Actual DSS2 states and a base state for the co-ordination protocol are derived from dss2State. The dss2CoOrdBaseState is derived from dss2State because it has some same inputs (PDU inputs) than in DSS2 and it helps error handling for inherited co-ordination states.

1.2.2 3.2 Messages and information elements

PDUs are composed of information elements. In our DSS2 implementation PDUs have a common base state which includes information elements as attributes. The PDU base class have also encoding and decoding functions for all information elements. In that way derived PDU classes are very small because only few PDUs have the special actions like error handling actions (e.g. if invalid information element occurs). Figure 4 shows a message hierarchy. PDUs are derived from pfMessenger and tvBitHandler which is a special class for handling bit operations. It has methods to put/get bits to/from the frame.

This helps to implement encoding and decoding functions and makes them simply, especially in DSS2 because almost all information elements consist of small (one to three bits) fields.

Information elements are located in the siginfo module. Classes in that module implement information element fields and methods for handling them. In the base class of information elements have methods for checking if the information element is mandatory or is it allowed at all in the PDU in question. Appendix Y has a figure of the DSS2 PDU format. First nine octets are common for all DSS2 PDUs and each information elements have four common octets at the begin of them.

Normally information elements or fields of them are copied from a PDU to a primitive to up direction (call control). But in the case of SETUPpdu a pointer to the PDU is copied to the primitive, so all information in the PDU are transferred also to the call control.

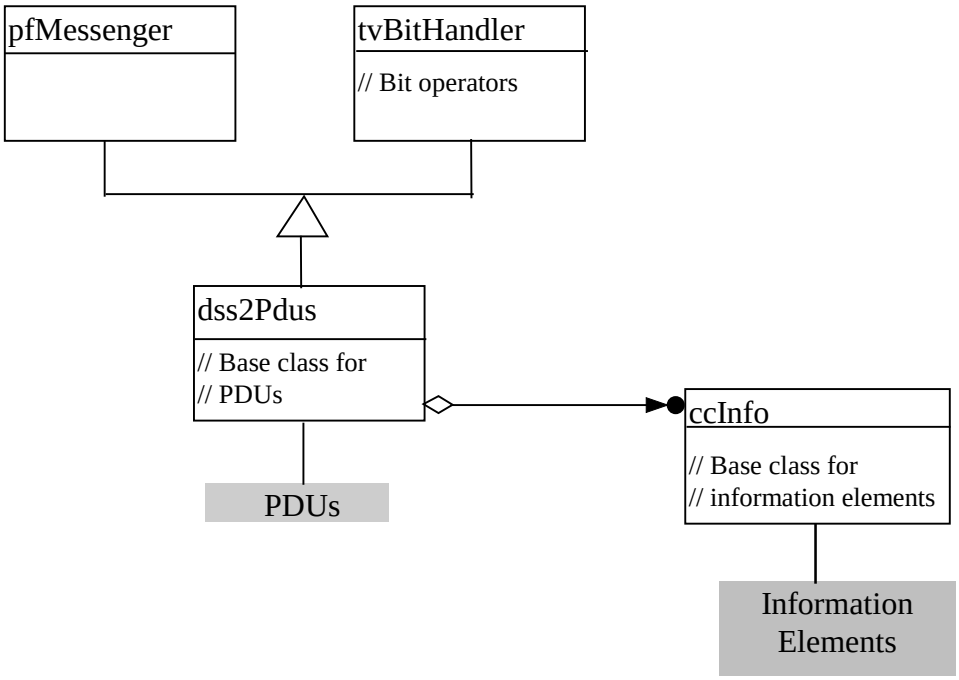

Figure 4 Message hierarchy

1.3 4 Features implemented

In the first release only mandatory (the minimal amount of elements for call/connection establishment and clearing procedures) information elements were implemented. The following list shows implemented information elements. All of them have coding functions, but some information elements are not used in the first release of the software.

- Called party number. Maximum length (network dependent) is not checked when decoding.

- Called party sub-address. Odd/even indicator not used.

- Restart indicator (not used)
- ATM traffic descriptor (not used)
- Connection identifier
- Quality of Service parameter (not used)
- Broadband bearer capability (not used)
- Call state
- Cause. Diagnostics field is not supported, but it will be decoded out from frame if presents.

An interface (sigif) between the DSS2 and the call control is of course insufficient and not very robust because of reasons mentioned above.

Next list shows other not implemented features in the first release:

- reset-start, reset-response modules
- additional PDUs. Interworking with 64 kbit/s ISDN.
- explicit error handling. Only normal error handling present.
- timers. All timers in the DSS2 and co-ordination protocol are not implemented.
- cause value is not set in all possible places.

1.4 5 Known bugs and flaws

There is one flaw founded in decoding functions of DSS2 information elements. Fixing of this could be taken quite lot of time because it may cause some general changes to DSS2 information element decoding functions.

If an information element which have optional fields is in the last one in a frame, it could be decoded wrong. This can happen when decoded information element length field doesn't match to the frame length but indicated length is enough long for mandatory fields. So the very last information element in the frame will be accepted even if information element's length value is greater than the length of octets left in the frame. The consequences are : information element is accepted and last fields are set to zero (if frame is empty the return value of a getFirst method is zero) but error isn't noticed.

1.5 6 Future development

When a data communication software is designed and implemented not all details can be considered at first time. When object-oriented programming is used, it's easy to develop the software again and make it clearer and more efficient by modifying the old code.

Following chapters describe problems and some ideas to solve them founded in the first programming phase of the Q.2931.

1.5.1 6.1 Co-Ordination

At the very first implementation, a problem occurs when a decoded PDU arrived in the mux. If a call reference of the PDU doesn't match any of values stored in the mux and the PDU is not a SETUPpdu or a call reference flag of SETUPpdu was set incorrectly, the PDU stayed at the mux and because it didn't reach any protocol it will never be deleted. The co-ordination should also send a STATUSpdu to downward. The solution for the problem that how co-ordination knows the final destination of PDU was boolean flag in a transporter. The flag was checked at the co-ordination when the method call returned.

Because the previous solution wasn't very nice, the implementation was modified so that the co-ordination protocol has a pointer to the mux implementation. Now the co-ordination can check the call reference before it sends PDU to the mux and it knows the final target of the PDU. But an idea of the OVOPS++ is to use proxies to implementations and doesn't let any other conduits to use direct implementations of other conduits. That is why major idea for the solution of this problem in next release is that the mux and the co-ordination will be combined together. Then the co-ordination is same time the protocol with a state machine and the mux with an accessor. More about this idea can be founded from the OVOPS++ documentation.

1.6 6.2 Access to information element fields

The PDU consist of information elements and each of them are composed of many fields (usually long integer values) as figure 4 showed. Because the lack of time to implement a better solution, information elements are now as publics in the PDU base class. Information element fields can be used direct from information element when the pointer to the PDU is available.

```
pdu→_connectionId.getVCI();
```

Better solution would be that the PDU has methods whose delegate access method calls to information elements.

```
pdu→getVCI();
```

But this implementation way will cause that the PDU will have several tens of methods (access methods for every fields in every information elements). The future solution may be founded in design patterns using visitors and double dispatching to find the right field from the right information element without tens of methods in the PDU.

Same experiences can be used to think structure of primitives in a sigiface and copying of parameters between PDUs and primitives.

1.7 6.3 Decoding without switch-case

It seems that identifying of PDUs and information element types, when decoding PDUs, can't be handled by any object-oriented ways and it should be solved with a switch-case structure. But one idea, is based on STL Map, was founded. Now information element objects are as attributes in the PDU. They also could been stored to STL Map with an identifier as key. When the identifier is decoded, the right information element is founded with the identifier from the map and then the right decode method can be called from the information element object instead of using the switch-case statement.

Problems may be occurred if more than one information element are instances from the same class. For example a called party number and a calling party number are both instances from phoneNumber class, but they have different coding functions. In the above idea decode method was called from the object founded in the map with the identifier, but here we need two decoding methods.

1.8 7 Statistics

Following table 1 shows hours used to implement modules related to DSS2 (table 2). It also contain hours used for a general codec module. Researching and implementing of some test prototypes have been taken also lots of time because of many new programming methods were used.

Activity	Research	Design	Coding	Reviews	Total
Duration (h)	170	200	590	0	960

Table 1 Duration of activities

Following table 2 shows statistics of the code related to an implementation of DSS2 specification. A siginfo module contains information elements and a sigif module contains primitive classes for the upper interface of the DSS2.

Module name	Lines Of Code (LOC)	Number of files	Number of classes
DSS2	5552	53	54
SIGINFO	825	3	11
SIGIF	1375	4	34

Table 2 Metrics

1.9 8 References

Object Space Inc., *Systems<ToolKit> UNIX Reference Manual*, 1994-1995.

Object Space Inc., *Systems<ToolKit> UNIX User's Manual*, 1994-1995.

ITU-T Recommendation Q.2931, 1995.

TOVE project, *OVOPS++ document.*

Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns, Elements of Object-Oriented Software*, Addison-Wesley, 1995.

1.10 Appendix I

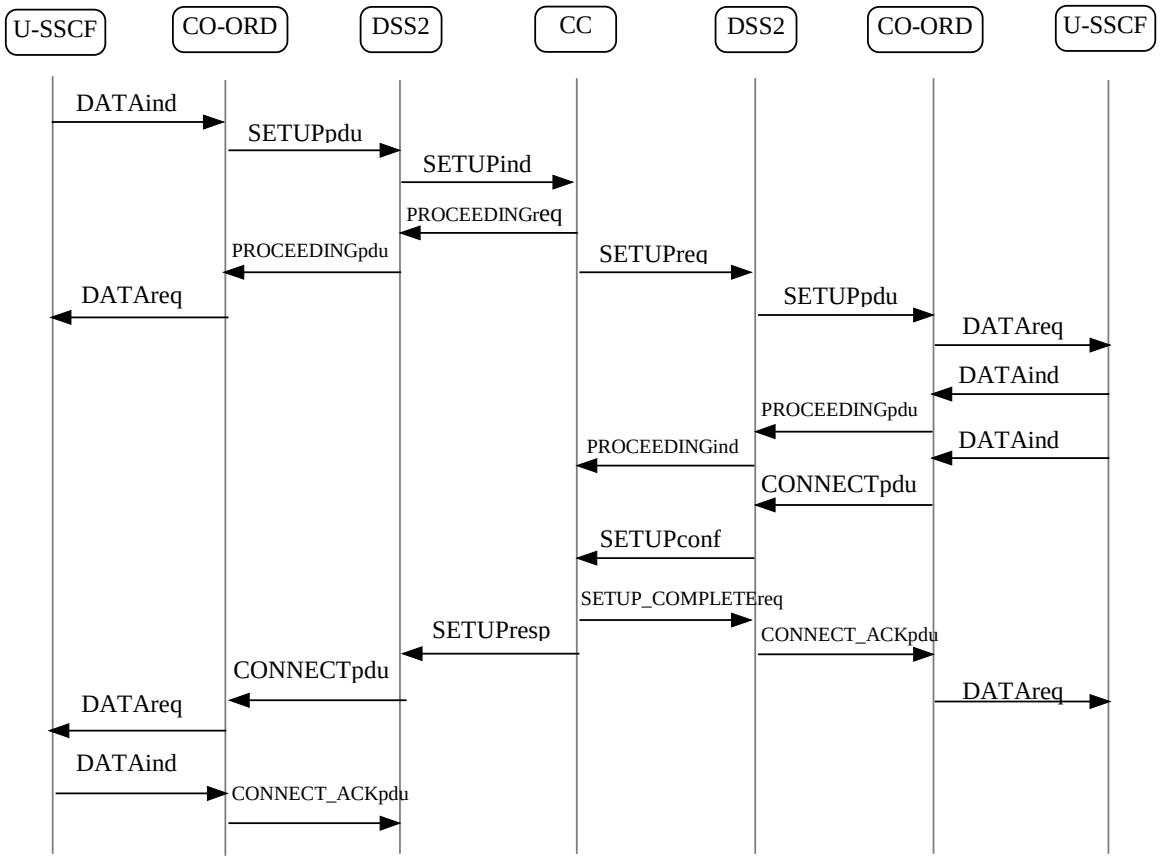

Figure 2 Normal call/connection establishment from the switch point of view.

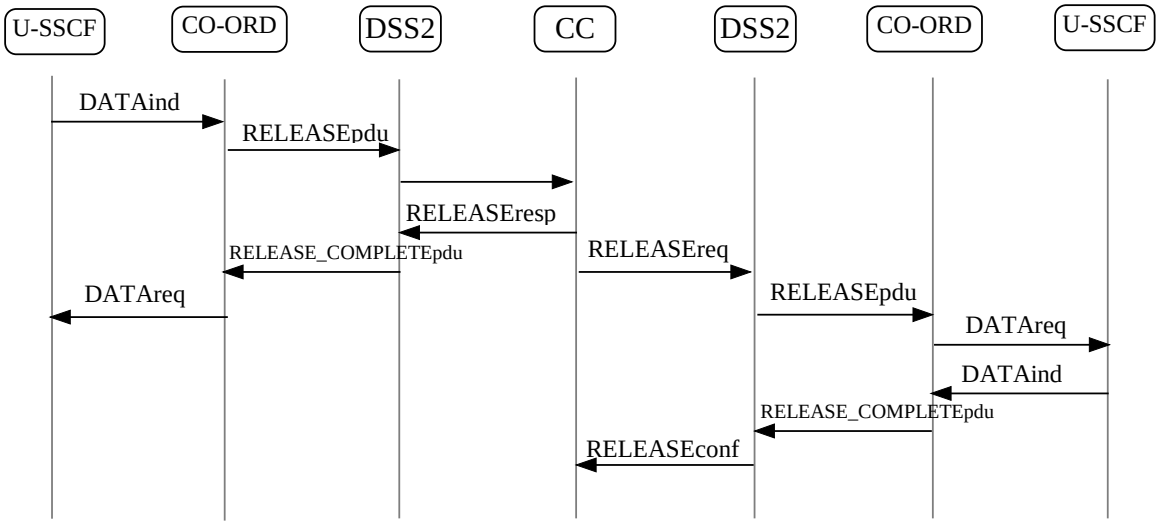

Figure 3 Normal call/connection clearing from the switch point of view

1.11 Appendix II

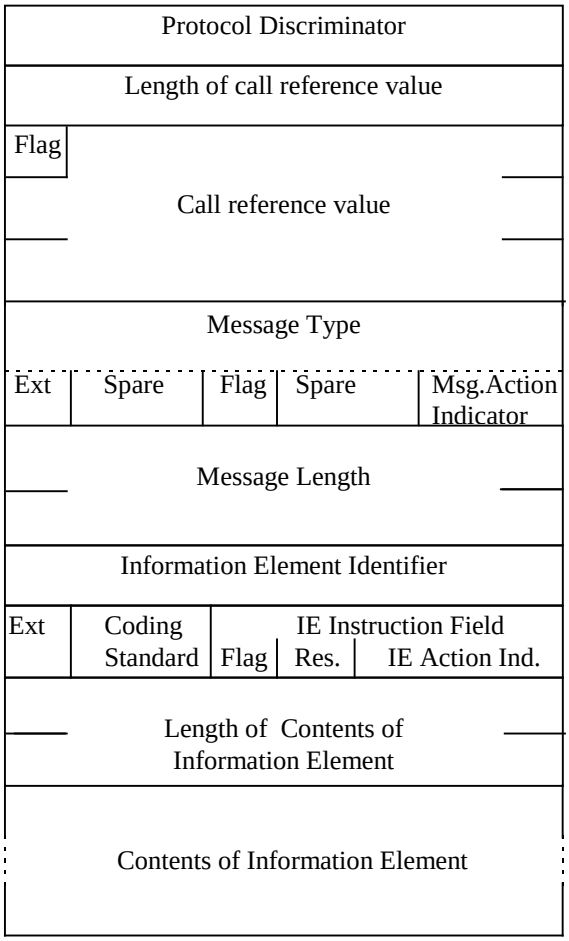

Figure 4 DSS2 message format