

TECHNICAL REPORT

ISO/IEC TR 11589

First edition
1995-10-01

Information technology — Open Systems Interconnection — LOTOS description of the CCR service

*Technologies de l'information — Interconnexion de systèmes ouverts
(OSI) — Description LOTOS du service CCR*



Reference number
ISO/IEC TR 11589:1995(E)

Contents

1 Scope 1

2 Normative references 1

3 Definitions 1

4 Symbols and abbreviations 2

5 Conventions 2

6 Introduction to the formal description 2

6.1 Model 2

6.2 Structure 2

6.3 Service overview 5

7 Global constraints of the CCR service 6

8 Service primitives 7

8.1 Basic construction 7

8.2 CSP classification 9

8.3 CSP parameter selectors 11

8.4 CSP parameters 12

8.4.1 Atomic Action Identifier 12

8.4.2 Branch Identifier 12

8.4.3 User Data 12

8.4.4 Recovery State 12

8.5 Others 13

9	Constraints of the superior side	13
9.1	Normal sequence of the superior	13
9.2	Recovery sequence of the superior	15
10	Constraints of the subordinate side	16
10.1	Normal sequence of the subordinate	17
10.2	Recovery sequence of the subordinate	19
11	Constraints of the relation between superior and subordinate	20
11.1	Service Primitive Queue	20
11.2	CCEPRelation	22

IECNORM.COM : Click to view the full PDF of ISO/IEC TR 11589:1995

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The main task of technical committees is to prepare International Standards. In exceptional circumstances a technical committee may propose the publication of a Technical Report of one of the following types:

- type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;
- type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;
- type 3, when a technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

ISO/IEC TR 11589, which is a Technical Report of type 2, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 21, *Open systems interconnection, data management and open distributed processing*.

Introduction

This Technical Report gives a LOTOS specification of CCR Service Definition defined in ISO/IEC 9804.

LOTOS is a formal description technique(FDT) to define behavior of systems with a formal syntax and semantics.

Aim of this Technical Report is to give an unambiguous, precise, and self-contained description of ISO/IEC 9804.

However, in case of inconsistency between ISO/IEC 9804 and this Technical Report, ISO/IEC 9804 takes precedence over this Technical Report.

IECNORM.COM : Click to view the full PDF of ISO/IEC TR 11589:1995

This page intentionally left blank

IECNORM.COM : Click to view the full PDF of ISO/IEC TR 11589:1995

Information technology — Open Systems Interconnection — LOTOS description of the CCR service

1 Scope

This Technical Report focuses on one atomic action branch and covers the relevant service primitives with their types and parameters, and the behaviour as to how these service primitives can be issued and received by the CCR service provider.

Service primitives and its parameters are defined in clause 8. The sequential rules on superior side is defined in clause 9, and on subordinate side is defined in clause 10. The relation of the events that occur on the superior and subordinate sides is described in clause 11.

This Technical Report does not cover the following items in ISO/IEC 9804.

- Clause 6: concept. Because formal language can not describe this kind of semantics.
- CCR service-user rules(Annex A), Relationship of CCR to the Application Layer Structure(Annex B).
- Description to manage more than one atomic action branch.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this Technical Report. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this Technical Report are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 9804:1994, *Information technology – Open Systems Interconnection – Service definition for the Commitment, Concurrency and Recovery service element.*

ISO/IEC 9805-1:1994, *Information technology – Open Systems Interconnection – Protocol for the Commitment, Concurrency and Recovery service element: Protocol Specification.*

ISO 8807:1989, *Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour.*

3 Definitions

For the purposes of this Technical Report the definitions given in ISO/IEC 9804 apply.

There is not a term “CCR connection end point” in ISO/IEC 9804. This term is used in this Technical Report to represent the point at which the CCR user received the service from CCR provider.

4 Symbols and abbreviations

The abbreviations given in ISO/IEC 9804 apply.

The abbreviation “LOTOS” is defined in ISO 8807.

The following additional abbreviations are employed in this Technical Report.

CCEP CCR Connection End Point
CSP CCR Service Primitive

5 Conventions

An action for a combination of an event and a state not defined in the state table is left to implementer as “a local matter” by ISO/IEC 9804. Such undefined events are not specified in the LOTOS specification at all. This is not an exact representation of natural language specification, because deadlock will occur when the CCR user insists to taking an undefined event. Events not defined in this LOTOS specification are expected to be treated as a local matter.

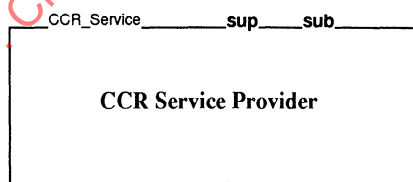
Clauses 7 through 11 of this Technical Report constitute LOTOS text. All explanatory descriptions are caught between “(” and “)” symbols and are treated as comments of the LOTOS text.

ISO/IEC 10731 (Service Conventions) is adopted, but the service primitive names and parameter names may be simplified or modified.

6 Introduction to the formal description

6.1 Model

CCR service is modeled in figure 1.



LEGEND
process name
gate name

Figure 1 - CCR service model

6.2 Structure

The relationship among the processes are shown in figure 2, and the relationship among the types are shown in figure 3. In these figures, “*” means that “the definition has already listed above”.

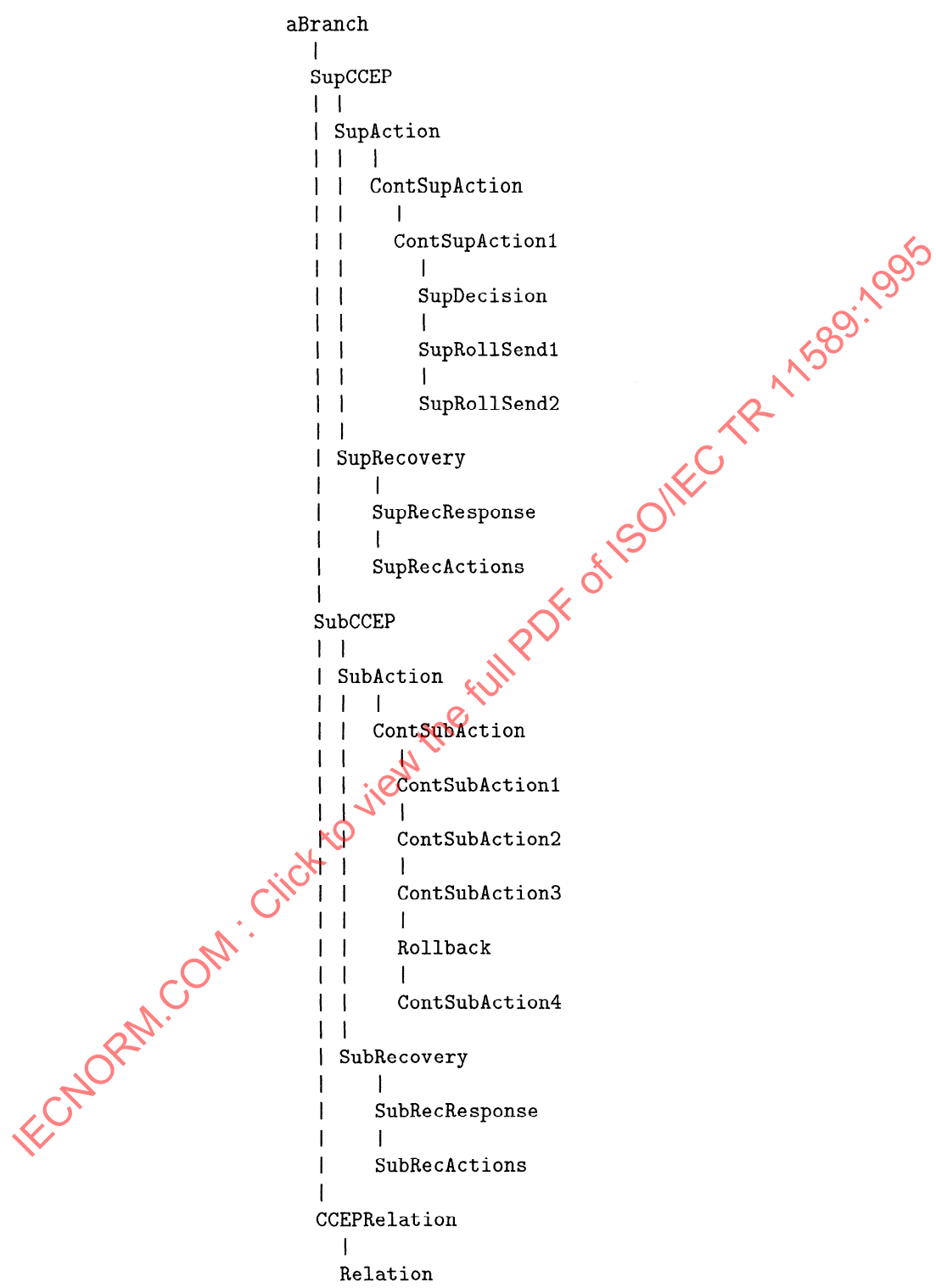


Figure 2 - Relationship among the processes



Figure 3 - Relationship among the types

6.3 Service overview

CCR service primitives and parameters are summarized in table 1.

Table 1 - Service Primitives and Parameters

Service Primitives and Parameters	Req	Ind	Rsp	Cnf
C-BEGIN				
Atomic Action Identifier	M	M(=)		
- Master's Name				
Atomic Action Identifier	M	M(=)		
- Suffix				
Branch Identifier	M	M(=)		
- Superior's Name				
Branch Identifier	M	M(=)		
- Suffix				
User Data	U	C(=)	U	C(=)
C-PREPARE				
User Data	U	C(=)		
C-READY				
User Data	U	C(=)		
C-COMMIT				
User Data	U	C(=)	U	C(=)
C-ROLLBACK				
User Data	U	C(=)	U	C(=)
C-RECOVER				
Recovery State	M	M(=)	M	M(=)
Atomic Action Identifier	M	M(=)	M	M(=)
- Master's Name				
Atomic Action Identifier	M	M(=)	M	M(=)
- Suffix				
Branch Identifier	M	M(=)	M	M(=)
- Superior's Name				
Branch Identifier	M	M(=)	M	M(=)
- Suffix				
User Data	U	C(=)	U	C(=)

The symbols in the table represent the following semantics;

blank	not applicable
C	conditional
M	mandatory
U	user option

In the table, notation (=) indicates that the parameter value is semantically equal to the value to its left in the table.

7 Global constraints of the CCR service

CCR service is represented in terms of type definitions and process definitions. The static attribute, i.e. the type of the service primitives and their parameters are described in type definitions. The behavior, i.e. the service primitives issuing/receiving procedures are described in process definitions.

Issuing and receiving of service primitives are represented as the events at the gates.

Formal data types, FBoolean, Boolean, NaturalNumber, Element, Set, OctetString, NatRepresentations, and DecDigit are imported from LOTOS standard library.

```
specification CCR_Service[sup,sub] : noexit
```

```
library
```

```
  FBoolean, Boolean, NaturalNumber, Element, Set, OctetString
endlib
```

CCR service defines the behavior of one atomic action branch. Constraints beyond one atomic action branch are described in annex A of ISO/IEC 9804, but they are not included in this Technical Report.

```
behavior
```

```
  Branch[sup,sub]
```

```
where
```

```
process Branch[sup,sub] : noexit :=
  aBranch[sup,sub] >> Branch[sup,sub]
```

```
where
```

CCR service are defined through the following three view points;

- behavior local at superior, not restricted by remote behavior,
- behavior local at subordinate, not restricted by remote behavior, and
- end-to-end relations which result from the exchange of information via the CCR service provider.

These are described in *SupCCEP*, *SubCCEP*, and *CCEPRelation* respectively.

```
process aBranch[sup,sub] : exit :=
```

```
  SupCCEP[sup]
  | [sup] |
  CCEPRelation[sup,sub]
  | [sub] |
  SubCCEP[sub]
where
```

8 Service primitives

8.1 Basic construction

ISO/IEC 9804 defines 6 service primitives, **C-BEGIN**, **C-PREPARE**, **C-READY**, **C-COMMIT**, **C-ROLLBACK**, **C-RECOVER**. These are represented by the following names respectively; **BEG**(or **Beg**), **PRP**(or **Prp**), **RDY**(or **Rdy**), **COMM**(or **Comm**), **ROLL**(or **Roll**), **REC**(or **Rec**).

Each of these service primitives has 4 primitive types, **request**, **indication**, **response**, **confirm**. These are represented as follows;

req, **ind**, **rsp**, **cnf**.

C-BEGIN service primitive may be concatenated with C-COMMIT service primitive or C-ROLLBACK service primitive. Therefore, these concatenated service primitives are included in the BasicCSP type.

Parameter attributes (mandatory, conditional, user option etc.) are not treated in this Technical Report. See 6.3.

```

type CSPName is Boolean, NaturalNumber
  sorts CSPName
  opns Beg, Prp, Rdy, Comm, Roll, Rec,
        Comm_plus_Beg, Roll_plus_Beg: -> CSPName
        _eq_, _ne_: CSPName, CSPName -> Bool
        h: CSPName -> Nat
  eqns
    forall f, f1: CSPName
      ofsort Nat
        h(Beg) = 0;
        h(Prp) = Succ(h(Beg));
        h(Rdy) = Succ(h(Prp));
        h(Comm) = Succ(h(Rdy));
        h(Roll) = Succ(h(Comm));
        h(Rec) = Succ(h(Roll));
        h(Comm_plus_Beg) = Succ(h(Rec));
        h(Roll_plus_Beg) = Succ(h(Comm_plus_Beg));
      ofsort Bool
        f eq f1 = h(f) eq h(f1);
        f ne f1 = not(f eq f1);
    endtype (* CSPName *)

```

```

type CSPTType is Boolean, NaturalNumber
  sorts CSPTType
  opns Req, Ind, Rsp, Cnf: -> CSPTType
        h: CSPTType -> Nat
        _eq_, _ne_: CSPTType, CSPTType -> Bool
        IsReq, IsInd, IsRsp, IsCnf: CSPTType -> Bool
        CorrType: CSPTType -> CSPTType
  eqns
    forall d, d1: CSPTType
      ofsort Nat
        h(Req) = 0;
        h(Ind) = Succ(h(Req));

```

```

    h(Rsp) = Succ(h(Ind));
    h(Cnf) = Succ(h(Rsp));
ofsort Bool
    d eq d1 = h(d) eq h(d1);
    d ne d1 = not(d eq d1);
    IsReq(d) = h(d) eq h(Req);
    IsInd(d) = h(d) eq h(Ind);
    IsRsp(d) = h(d) eq h(Rsp);
    IsCnf(d) = h(d) eq h(Cnf);
ofsort CSPTYPE
    CorrType(Req) = Ind;
    CorrType(Ind) = Req;
    CorrType(Rsp) = Cnf;
    CorrType(Cnf) = Rsp;
endtype (* CSPTYPE *)

type CSPConstant is
    NaturalNumber, CSPName, CSPTYPE
sorts CSPConstant
opns c: CSPName, CSPTYPE -> CSPConstant
    IsConstantReq, IsConstantInd: CSPConstant -> Bool
    _eq_, _ne_: CSPConstant, CSPConstant -> Bool
eqns
    forall c,c1: CSPConstant,
        n,n1: CSPName,
        t,t1: CSPTYPE
ofsort Bool
    IsConstantReq(c(n,t)) = IsReq(t) or IsRsp(t);
    IsConstantInd(c(n,t)) = IsInd(t) or IsCnf(t);
    c(n,t) eq c(n1,t1) = (n eq n1) and (t eq t1);
    c ne c1 = not(c eq c1);
endtype (* CSPConstant *)

```

```

type BasicCSP is
    CSPConstant, AtomicActionId, BranchId, UserData,
    Requestor_Recovery_State, Responder_Recovery_State
sorts CSP
opns BEGreq, BEGind: AAID, BID, DATA -> CSP
    BEGrsp, BEGcnf: DATA -> CSP
    PRPreq, PRPind: DATA -> CSP
    RDYreq, RDYind: DATA -> CSP
    COMMreq, COMMind, COMMrsp, COMMcnf: DATA -> CSP
    ROLLreq, ROLLind, ROLLrsp, ROLLcnf: DATA -> CSP
    RECreq, RECind:
        ReqRSTATE, AAID, BID, DATA -> CSP
    RECrsp, RECcnf:
        RspRSTATE, AAID, BID, DATA -> CSP
    COMMreq_plus_BEGreq, COMMind_plus_BEGind,
    ROLLreq_plus_BEGreq, ROLLind_plus_BEGind:
        DATA, AAID, BID, DATA -> CSP
    f1: CSPConstant, AAID, BID, DATA -> CSP
    f2: CSPConstant, DATA -> CSP

```

```

f3: CSPConstant, ReqRSTATE, AAID, BID, DATA ->CSP
f3: CSPConstant, RspRSTATE, AAID, BID, DATA ->CSP
f4: CSPConstant, DATA, AAID, BID, DATA ->CSP
eqns
forall a: AAID, b: BID, u,u1:DATA, r1:ReqRSTATE, r2:RspRSTATE,
      c: CSPConstant
ofsort CSP
  BEGreq(a,b,u) = f1(c(Beg,Req),a,b,u);
  BEGind(a,b,u) = f1(c(Beg,Ind),a,b,u);
  BEGrsp(u) = f2(c(Beg,Rsp),u);
  BEGcnf(u) = f2(c(Beg,Cnf),u);
  PRPreq(u) = f2(c(Prp,Req),u);
  PRPind(u) = f2(c(Prp,Ind),u);
  RDYreq(u) = f2(c(Rdy,Req),u);
  RDYind(u) = f2(c(Rdy,Ind),u);
  COMMreq(u) = f2(c(Comm,Req),u);
  COMMind(u) = f2(c(Comm,Ind),u);
  COMMrsp(u) = f2(c(Comm,Rsp),u);
  COMMcnf(u) = f2(c(Comm,Cnf),u);
  ROLLreq(u) = f2(c(Roll,Req),u);
  ROLLind(u) = f2(c(Roll,Ind),u);
  ROLLrsp(u) = f2(c(Roll,Rsp),u);
  ROLLcnf(u) = f2(c(Roll,Cnf),u);
  RECreq(r1,a,b,u) = f3(c(Rec,Req),r1,a,b,u);
  RECind(r1,a,b,u) = f3(c(Rec,Ind),r1,a,b,u);
  RECrsp(r2,a,b,u) = f3(c(Rec,Rsp),r2,a,b,u);
  RECcnf(r2,a,b,u) = f3(c(Rec,Cnf),r2,a,b,u);
  COMMreq_plus_BEGreq(u,a,b,u1) = f4(c(Comm_plus_Beg,Req),u,a,b,u1);
  COMMind_plus_BEGind(u,a,b,u1) = f4(c(Comm_plus_Beg,Ind),u,a,b,u1);
  ROLLreq_plus_BEGreq(u,a,b,u1) = f4(c(Roll_plus_Beg,Req),u,a,b,u1);
  ROLLind_plus_BEGind(u,a,b,u1) = f4(c(Roll_plus_Beg,Ind),u,a,b,u1);
endtype (* BasicCSP *)

```

8.2 CSP classification

type CSPBasicClassifiers is BasicCSP,CSPConstant,Boolean

```

opns
  k:CSP ->CSPConstant
  IsBEGreq,IsBEGind,IsBEGrsp,IsBEGcnf,
  IsBEG,IsBEGACK,
  IsPRPreq,IsPRPind,IsPRP,
  IsRDYreq,IsRDYind,IsRDY,
  IsCOMMreq,IsCOMMind,IsCOMMrsp,IsCOMMcnf,
  IsCOMM,IsCOMMACK,
  IsROLLreq,IsROLLind,IsROLLrsp,IsROLLcnf,
  IsROLL,IsROLLACK,
  IsRECreq,IsRECind,IsRECrsp,IsRECcnf,
  IsREC,IsRECAK,
  IsCOMMreq_plus_BEGreq,IsCOMMind_plus_BEGind,IsCOMM_plus_BEG,
  IsROLLreq_plus_BEGreq,IsROLLind_plus_BEGind,IsROLL_plus_BEG,
  IsROLLreqs,IsROLLinds,
  IsReq,IsInd: CSP ->Bool
  IndOf: CSP ->CSP

```

```

_eq_, _ne_: CSP,CSP ->Bool
_match_: CSP,CSP -> Bool
eqns
forall a,aa: AAID, b,bb: BID, u,uu,u1,uu1: DATA,
    r1,rr1: ReqRSTATE, r2,rr2:RspRSTATE,
    p,pp: CSP, c: CSPConstant, x:CSPName, y:CSPTYPE
ofsort CSPConstant
    k(f1(c,a,b,u)) = c;
    k(f2(c,u)) = c;
    k(f3(c,r1,a,b,u1)) = c;
    k(f3(c,r2,a,b,u1)) = c;
    k(f4(c,u,a,b,u1)) = c;
ofsort Bool
    IsBEGreq(p) = k(p) eq c(Beg, Req);
    IsBEGind(p) = k(p) eq c(Beg, Ind);
    IsBEGrsp(p) = k(p) eq c(Beg, Rsp);
    IsBEGcnf(p) = k(p) eq c(Beg, Cnf);
    IsPRPreq(p) = k(p) eq c(Prp, Req);
    IsPRPind(p) = k(p) eq c(Prp, Ind);
    IsRDYreq(p) = k(p) eq c(Rdy, Req);
    IsRDYind(p) = k(p) eq c(Rdy, Ind);
    IsCOMMreq(p) = k(p) eq c(Comm, Req);
    IsCOMMind(p) = k(p) eq c(Comm, Ind);
    IsCOMMrsp(p) = k(p) eq c(Comm, Rsp);
    IsCOMMcnf(p) = k(p) eq c(Comm, Cnf);
    IsROLLreq(p) = k(p) eq c(Roll, Req);
    IsROLLind(p) = k(p) eq c(Roll, Ind);
    IsROLLrsp(p) = k(p) eq c(Roll, Rsp);
    IsROLLcnf(p) = k(p) eq c(Roll, Cnf);
    IsRECre(p) = k(p) eq c(Rec, Req);
    IsRECind(p) = k(p) eq c(Rec, Ind);
    IsRECrsp(p) = k(p) eq c(Rec, Rsp);
    IsRECcnf(p) = k(p) eq c(Rec, Cnf);
    IsBEG(p) = IsBEGreq(p) or IsBEGind(p);
    IsBEGACK(p) = IsBEGrsp(p) or IsBEGcnf(p);
    IsPRP(p) = IsPRPreq(p) or IsPRPind(p);
    IsRDY(p) = IsRDYreq(p) or IsRDYind(p);
    IsCOMM(p) = IsCOMMreq(p) or IsCOMMind(p);
    IsCOMMACK(p) = IsCOMMrsp(p) or IsCOMMcnf(p);
    IsROLL(p) = IsROLLreq(p) or IsROLLind(p);
    IsROLLACK(p) = IsROLLrsp(p) or IsROLLcnf(p);
    IsREC(p) = IsRECre(p) or IsRECind(p);
    IsRECA(p) = IsRECrsp(p) or IsRECcnf(p);
    IsCOMMreq_plus_BEGreq(p) =
        k(p) eq c(Comm_plus_Beg, Req);
    IsCOMMind_plus_BEGind(p) =
        k(p) eq c(Comm_plus_Beg, Ind);
    IsCOMM_plus_BEG(p) =
        IsCOMMreq_plus_BEGreq(p) or IsCOMMind_plus_BEGind(p);
    IsROLLreq_plus_BEGreq(p) =
        k(p) eq c(Roll_plus_Beg, Req);
    IsROLLind_plus_BEGind(p) =
        k(p) eq c(Roll_plus_Beg, Ind);
    IsROLL_plus_BEG(p) =

```

```

    IsROLLreq_plus_BEGreq(p) or IsROLLind_plus_BEGind(p);
    IsReq(p) = IsConstantReq(k(p));
    IsInd(p) = IsConstantInd(k(p));
    IsROLLreqs(p) = IsROLLreq(p) or IsROLLreq_plus_BEGreq(p);
    IsROLLinds(p) = IsROLLind(p) or IsROLLind_plus_BEGind(p);
ofsort CSP
    IndOf(f1(c(x,y),a,b,u)) = f1(c(x,CorrType(y)),a,b,u);
    IndOf(f2(c(x,y),u)) = f2(c(x,CorrType(y)),u);
    IndOf(f3(c(x,y),r1,a,b,u)) = f3(c(x,CorrType(y)),r1,a,b,u);
    IndOf(f3(c(x,y),r2,a,b,u)) = f3(c(x,CorrType(y)),r2,a,b,u);
    IndOf(f4(c(x,y),u,a,b,u1)) = f4(c(x,CorrType(y)),u,a,b,u1);
ofsort Bool
    p eq pp = k(p) eq k(pp);
    p ne pp = k(p) ne k(pp);
ofsort Bool
    p ne pp => p match pp = false;
    f1(c,a,b,u) match f1(c,aa,bb,uu) =
        (a eq aa) and (b eq bb) and (u eq uu);
    f2(c,u) match f2(c,uu) = (u eq uu);
    f3(c,r1,a,b,u) match f3(c,rr1,aa,bb,uu) =
        (r1 eq rr1) and (a eq aa) and (b eq bb) and (u eq uu);
    f3(c,r2,a,b,u) match f3(c,rr2,aa,bb,uu) =
        (r2 eq rr2) and (a eq aa) and (b eq bb) and (u eq uu);
    f4(c,u,a,b,u1) match f4(c,uu,aa,bb,uu1) =
        (u eq uu) and (a eq aa) and (b eq bb) and (u1 eq uu1);
endtype (* CSPBasicClassifiers *)

```

8.3 CSP parameter selectors

```

type CSPPParameterSelectors is CSPBasicClassifiers
  opns
    AId:CSP ->AAID
    BId:CSP ->BID
    UData:CSP ->DATA
    ReqRstate:CSP ->ReqRSTATE
    RspRstate:CSP ->RspRSTATE
    CommUData,RollUData,BegUData:CSP ->DATA
  eqns
    forall c: CSPConstant, a: AAID, b: BID, u,u1: DATA,
      r1:ReqRSTATE, r2:RspRSTATE
  ofsort AAID
    AId(f1(c,a,b,u)) = a;
    AId(f3(c,r1,a,b,u)) = a;
    AId(f3(c,r2,a,b,u)) = a;
    AId(f4(c,u,a,b,u1)) = a;
  ofsort BID
    BId(f1(c,a,b,u)) = b;
    BId(f3(c,r1,a,b,u)) = b;
    BId(f3(c,r2,a,b,u)) = b;
    BId(f4(c,u,a,b,u1)) = b;
  ofsort DATA
    UData(f1(c,a,b,u)) = u;
    UData(f2(c,u)) = u;

```

```

    UData(f3(c,r1,a,b,u)) = u;
    UData(f3(c,r2,a,b,u)) = u;
    CommUData(f4(c,u,a,b,u1)) = u;
    RollUData(f4(c,u,a,b,u1)) = u;
    BegUData(f4(c,u,a,b,u1)) = u1;
ofsort ReqRSTATE
    ReqRstate(f3(c,r1,a,b,u)) = r1;
ofsort RspRSTATE
    RspRstate(f3(c,r2,a,b,u)) = r2;
endtype (* CSPPParameterSelectors *)

```

8.4 CSP parameters

8.4.1 Atomic Action Identifier

```

type AtomicActionId is OctetString renamedby
    sortnames AAID for OctetString
endtype (* AtomicActionId *)

```

8.4.2 Branch Identifier

```

type BranchId is OctetString renamedby
    sortnames BID for OctetString
endtype (* BranchId *)

```

8.4.3 User Data

```

type UserData is OctetString renamedby
    sortnames DATA for OctetString
endtype (* UserData *)

```

8.4.4 Recovery State

```

type Requestor_Recovery_State is NaturalNumber
    sorts ReqRSTATE
    opns commit,ready:->ReqRSTATE
        h:ReqRSTATE->Nat
        _eq_,_ne_:ReqRSTATE,ReqRSTATE->Bool
    eqns
        forall r,r1:ReqRSTATE
            ofsort Nat
                h(commit) = 0;
                h(ready) = Succ(h(commit));
            ofsort Bool

```

```

    r1 eq r = h(r1) eq h(r);
    r1 ne r = not(r1 eq r);
endtype (* Requestor_Recovery_State *)

type Responder_Recovery_State is NaturalNumber
  sorts RspRSTATE
  opns done, unknown, retry: -> RspRSTATE
    h: RspRSTATE -> Nat
    _eq_, _ne_: RspRSTATE, RspRSTATE -> Bool
  eqns
    forall r, r1: RspRSTATE
      ofsort Nat
        h(done) = 0;
        h(unknown) = Succ(h(done));
        h(retry) = Succ(h(unknown));
      ofsort Bool
        r1 eq r = h(r1) eq h(r);
        r1 ne r = not(r1 eq r);
endtype (* Responder_Recovery_State *)

```

8.5 Others

```

type Error is Boolean
  sorts ERROR
  opns failure : -> ERROR
endtype (* Error *)

```

9 Constraints of the superior side

The superior side normal sequences including rollback procedure are defined in process SupAction that is disabled by the error event at gate e.

Failures are divided into two categories in ISO/IEC 9804. One is the application failure and the other is the communication failure. In this Technical Report, both of them are represented by “e !failure”.

```

process SupCCEP[sup] : exit :=
  ( hide e in
    SupAction[sup]
    []
    SupRecovery[sup]
  ) [> e !failure; exit )

```

where

9.1 Normal sequence of the superior

The acknowledgement procedure of C-BEGIN service are optional, and C-PREPARE service is optional. These attributes are represented by the disable operator “[>”.

```
process SupAction[sup] : exit :=
```

```
    sup ?p:CSP[IsBEGreq(p)];
    ContSupAction[sup]
```

where

```
process ContSupAction[sup] : exit :=
```

```
    (
      ( sup ?p:CSP[IsBEGcnf(p)];
        sup ?p:CSP[IsPRPreq(p)];
        stop )
      []
      ( sup ?p:CSP[IsPRPreq(p)];
        sup ?p:CSP[IsBEGcnf(p)];
        stop ) )
    [>
    ContSupAction1[sup]
```

where

```
process ContSupAction1[sup] : exit :=
```

```
    ( sup ?p:CSP[IsRDYind(p)];
      SupDecision[sup] )
    []
    ( sup ?p:CSP[IsROLLind(p)];
      sup ?p:CSP[IsROLLrsp(p)];
      exit )
    []
    ( sup ?p:CSP[IsROLLreq(p)];
      SupRollSend1[sup] )
    []
    ( sup ?p:CSP[IsROLLreq_plus_BEGreq(p)];
      SupRollSend2[sup] )
```

where

```
process SupDecision[sup] : exit :=
```

```
    ( sup ?p:CSP[IsCOMMreq(p)];
      sup ?p:CSP[IsCOMMcnf(p)];
      exit )
    []
    ( sup ?p:CSP[IsROLLreq(p)];
      sup ?p:CSP[IsROLLcnf(p)];
      exit )
    []
    ( sup ?p:CSP[IsCOMMreq_plus_BEGreq(p)];
```

```

        sup ?p:CSP[IsCOMMcnf(p)];
        ContSupAction[sup] )
[]
( sup ?p:CSP[IsROLLreq_plus_BEGrreq(p)];
  sup ?p:CSP[IsROLLcnf(p)];
  ContSupAction[sup] )

endproc (* SupDecision *)

process SupRollSend1[sup] : exit :=

    ( sup ?p:CSP[IsROLLcnf(p)];
      exit )
[]
( sup ?p:CSP[IsROLLind(p)];
  sup ?p:CSP[IsROLLrsp(p)];
  exit )

endproc (* SupRollSend1 *)

process SupRollSend2[sup] : exit :=

    ( sup ?p:CSP[IsROLLcnf(p)];
      ContSupAction[sup] )
[]
( sup ?p:CSP[IsROLLind(p)];
  sup ?p:CSP[IsROLLrsp(p)];
  ContSupAction[sup] )

endproc (* SupRollSend2 *)

endproc (* ContSupAction1 *)

endproc (* ContSupAction *)

endproc (* SupAction *)

```

9.2 Recovery sequence of the superior

```

process SupRecovery[sup] : exit :=

    ( sup ?p:CSP[IsRECind(p) and (ReqRstate(p) eq ready)];
      SupRecResponse[sup] )
[]
( sup ?p:CSP[IsRECreq(p) and (ReqRstate(p) eq commit)];
  SupRecActions[sup] )

```

where

```

process SupRecResponse[sup] : exit :=
    ( sup ?p:CSP[IsRECreq(p) and (ReqRstate(p) eq commit)];
      SupRecActions[sup] )
[]
    ( sup ?p:CSP[IsRECrsp(p) and (RspRstate(p) eq unknown)];
      exit )
[]
    ( sup ?p:CSP[IsRECrsp(p) and (RspRstate(p) eq retry)];
      i; (* i works as a retry timer *)
      SupRecovery[sup] )

endproc (* SupRecResponse *)

process SupRecActions[sup] : exit :=
    ( sup ?p:CSP[IsRECcnf(p) and (RspRstate(p) eq done)];
      exit )
[]
    ( sup ?p:CSP[IsRECcnf(p) and (RspRstate(p) eq retry)];
      SupRecovery[sup] )

endproc (* SupRecActions *)

endproc (* SupRecovery *)

endproc (* SupCCEP *)

```

10 Constraints of the subordinate side

The subordinate side normal sequences including rollback procedure are defined in process SubAction that is disabled by the error event at gate e.

Failures are divided into two categories in ISO/IEC 9804. One is the application failure and the other is the communication failure. In this Technical Report, both of them are represented by “e !failure”.

```

process SubCCEP[sub] : exit :=
    ( hide e in
      ( SubAction[sub]
        []
        SubRecovery[sub]
      ) [> e!failure ; exit )

```

where

10.1 Normal sequence of the subordinate

```
process SubAction[sub] : exit :=
```

```
    sub ?p:CSP[IsBEGind(p)] ;
    ContSubAction[sub]
```

where

```
process ContSubAction[sub] : exit :=
```

```
    (* Sequence after C-BEGIN indication *)
```

```
    sub ?p:CSP[IsBEGrsp(p)] ;
    (
        sub ?p:CSP[IsPRPind(p)] ;
        ContSubAction1[sub]

        [] sub ?p:CSP[IsRDYreq(p)] ;
        ContSubAction2[sub]

        [] Rollback[sub] )

    [] sub ?p:CSP[IsPRPind(p)] ;
    (
        sub ?p:CSP[IsBEGrsp(p)] ;
        ContSubAction1[sub]

        [] sub ?p:CSP[IsRDYreq(p)] ;
        ContSubAction3[sub]

        [] Rollback[sub] )

    [] sub ?p:CSP[IsRDYreq(p)] ;
    ContSubAction2[sub]

    [] Rollback[sub]
```

where

```
process ContSubAction1[sub] :exit :=
```

```
    (* sequence after C-BEGIN response and C-PREPARE indication. *)
```

```
    sub?p:csp[IsRDYreq(p)] ;
    ContSubAction3[sub]
```

```
    [] Rollback[sub]
```

```
endproc (* ContSubAction1 *)
```

```
process ContSubAction2[sub] :exit :=
```

```

(* sequence after C-READY request but not C-PREPARE indication. *)

    sub ?p:csp[IsPRPind(p)] ;
    ContSubAction3[sub]

[] ContSubAction3[sub]

endproc (* ContSubAction2 *)

process ContSubAction3[sub] :exit :=

    (* sequence after C-READY request. *)

    sub ?p:csp[IsCOMMind(p)] ;
    sub ?p:csp[IsCOMMrsp(p)] ;
    exit

[] sub ?p:csp[IsROLLind(p)] ;
    sub ?p:csp[IsROLLrsp(p)] ;
    exit

[] sub ?p:csp[IsCOMMind_plus_BEGind(p)] ;
    sub ?p:csp[IsCOMMrsp(p)] ;
    ContSubAction[sub]

[] sub ?p:csp[IsROLLind_plus_BEGind(p)] ;
    ContSubAction4[sub]

endproc (* ContSubAction3 *)

process Rollback[sub] :exit :=

    (* rollback sequence. *)

    sub ?p:csp[IsROLLind(p)] ;
    sub ?p:csp[IsROLLrsp(p)] ;
    exit

[] sub ?p:csp[IsROLLreq(p)] ;
    ( sub ?p:csp[IsROLLind(p)] ;
      sub ?p:csp[IsROLLrsp(p)] ;
      exit

[] sub ?p:csp[IsROLLcnf(p)] ;
    exit

[] sub ?p:csp[IsROLLind_plus_BEGind(p)] ;
    ContSubAction4[sub] )

[] sub ?p:csp[IsROLLind_plus_BEGind(p)] ;
    ContSubAction4[sub]

```

```
endproc (* rollback *)
```

```
process ContSubAction4[sub] :exit :=
```

```
    (* waits for C-ROLLBACK response. *)
```

```
    sub ?p:CSP[IsROLLrsp(p)] ;  
    ContSubAction[sub]
```

```
endproc (* ContSubAction4 *)
```

```
endproc (* ContSubAction *)
```

```
endproc (* SubAction *)
```

10.2 Recovery sequence of the subordinate

```
process SubRecovery[sub] : exit :=
```

```
    ( sub ?p:CSP[IsRECreq(p) and (ReqRstate(p) eq ready)];  
      SubRecResponse[sub] )
```

```
    []
```

```
    ( sub ?p:CSP[IsRECind(p) and (ReqRstate(p) eq commit)];  
      SubRecActions[sub] )
```

where

```
process SubRecResponse[sub] : exit :=
```

```
    ( sub ?p:CSP[IsRECcnf(p) and (RspRstate(p) eq unknown)];  
      exit )
```

```
    []
```

```
    ( sub ?p:CSP[IsRECcnf(p) and (RspRstate(p) eq retry)];  
      exit )
```

```
    []
```

```
    ( sub ?p:CSP[IsRECind(p) and (ReqRstate(p) eq commit)];  
      SubRecActions[sub] )
```

```
endproc (* SubRecResponse *)
```

```
process SubRecActions[sub] : exit :=
```

```
    ( sub ?p:CSP[IsRECrsp(p) and (RspRstate(p) eq done)];  
      exit )
```

```
    []
```

```
    ( sub ?p:CSP[IsRECrsp(p) and (RspRstate(p) eq retry)];  
      exit )
```