J.10.4 OSTM Illustration of Stack

```
with HRTS_PE;
with TFSMs;use type HRTS_PE.T_Integer;
package Stack_OSTM is
  type T_OSTDState is (EMPTY, FULL, NON_E_F, STOPPED, UNDEFINED);
  -- only state constra‘ined operations
  type T_OSTDOperation is (START, STOP, PUSH, POP);
  NB_MAX_STATES     :       constant HRTS_PE.T_Integer
                                := T_OSTDState’pos (T_OSTDState’last);
  NB_MAX_OPERATIONS  :  constant HRTS_PE.T_Integer
                                := T_OSTDOperation’pos (T_OSTDOperation’last);
  NB_MAX_TRANSITIONS :      constant HRTS_PE.T_Integer
                                := NB_MAX_STATES*NB_MAX_OPERATIONS;
  package TFsm is new G_TFsm     (
    T_Operation     =>                    T_OSTDOperation,
    T_State         =>                    T_OSTDState,
    NB_MAX_TRANSITIONS   => NB_MAX_TRANSITIONS  );
  function Stack_FSM return TFsm.Instance;
end Stack_OSTM;

package body Stack_OSTM is
  function Stack_FSM return TFsm.Instance is
  FSM           : TFsm.Instance;
  begin
  -- Creation of the state machine for object STACK
  -- note :
  --   Conditional transitions are directly processed in the operation procedure code, in order
  --   to test the conditio ne.g.  --   - trans (NON_E_F, PUSH, FULL)
    TFsm.Create (FSM, 4, 1, STOPPED);     -- initial state is STOPPED
     TFsm.Trans (FSM, EMPTY, START, EMPTY); -- no restarting an active stack
     TFsm.Trans (FSM, EMPTY, PUSH, NON_E_F);
     TFsm.Trans (FSM, EMPTY, STOP, STOPPED);
     TFsm.Trans (FSM, NON_E_F, PUSH, NON_E_F);
    TFsm.Trans (FSM, NON_E_F, POP, NON_E_F);
     TFsm.Trans FSM, NON_E_F, STOP, STOPPED);
     TFsm.Trans (FSM, FULL, POP, NON_E_F);
    TFsm.Trans (FSM, FULL, STOP, STOPPED);
    TFsm.Trans (FSM, STOPPED, START, EMPTY);
    return FSM;
  end Stack_FSM;
end Stack_OSTM;
```

Figure 98 STack_OSTM code illustration

J.10.3.3 Object_Server code Illustration in C++

```
#ifndef STACK_SERVER_H
#define STACK_SERVER_H
#include <hrts/HRTS_PE.h>
class TStack_SERVER
{ TPE::integer CurrentSize              ; // CurrentSize
 private:
     enum    {      K_STACK_SIZE = 512    };  // trick to define the array size of StackBuffer
   TPE::OK_KO   Status               ; // Status
   TString      StackBuffer[ K_STACK_SIZE ]; // StackValues
   TStack_SERVER( const TStack_SERVER & ) {}
   void operator=( const TStack_SERVER & ) {}
 public:
   TStack_SERVER ();
   ~TStack_SERVER();
   void Push( const TString & );
   void Pop (  TString & );
    void Reset();
   TPE::integer GetMaxSize    ();
   TPE::integer GetCurrentSize();
   TPE::OK_KO   GetStatus     () const;
};
#endif /*STACK_SERVER_H*/
#include "Stack_SERVER.h"
#include <hrts/EXCEPTIONS.h>
// ===========================================================================
TStack_SERVER::TStack_SERVER()
{  CurrentSize = 0;
  Status = TPE::OK;
}
// ===========================================================================
TStack_SERVER::~TStack_SERVER()
{}
// ===========================================================================
void TStack_SERVER::Push( const TString &item )
{  if ( CurrentSize < K_STACK_SIZE )  {
    StackBuffer[ CurrentSize ] = item          ;
    CurrentSize              = CurrentSize + 1;
    Status                   = TPE::OK        ;
  }  else  {
    cerr << "STACK SIZE=" << K_STACK_SIZE   << endl << "CurrentSize="   << CurrentSize    << endl;
    Status = TPE::KO;
    EXCEPTIONS_RAISE( PROJECT_PE::X_FULL , "TStack_SERVER::Push" ,    "X_FULL"    );
  }
}
// ===========================================================================
void TStack_SERVER::Pop( TString &item )
{  if ( CurrentSize > 0 )  {
    CurrentSize = CurrentSize - 1          ;
    item        = StackBuffer[ CurrentSize ];
    Status      = TPE::OK                  ;
  }  else  {
    cerr << "STACK SIZE=" << K_STACK_SIZE   << endl << "CurrentSize="  << CurrentSize    << endl;
    Status = TPE::KO;
    EXCEPTIONS_RAISE( PROJECT_PE::X_EMPTY,  "TStack_SERVER::Pop" ,"X_EMPTY"     );
  }
}
// ===========================================================================
void TStack_SERVER::Reset()
{  CurrentSize = 0;
  Status = TPE::OK;
}
// ===========================================================================
TPE::integer TStack_SERVER::GetMaxSize()
{  Status = TPE::OK;
  return K_STACK_SIZE;
}
// ===========================================================================
TPE::integer TStack_SERVER::GetCurrentSize()
{  Status = TPE::OK;
  return CurrentSize;
}
// ===========================================================================
TPE::OK_KO TStack_SERVER::GetStatus() const
{  return Status; }
```

```
void TStack_RB::Push( void )
{  TItemStack Item;
    M >> Item; //lecture de l'item dans le message que l'on vient de recevoir
  //init et inverse sender et sendee
  M.init(M.GetSendee(), M.GetSender(), M.GetOperation(), M.GetCnstrt() );
  Stack.Push( Item );//empile
  EXCEPTIONS_HANDLE();
}

// ----------------------------------------------------------------------------
void TStack_RB::Pop( void )
{  TItemStack Item;
   //init et inverse sender et sendee
  M.init(M.GetSendee(), M.GetSender(), M.GetOperation(), M.GetCnstrt() );
  Stack.Pop( Item );//depile
  EXCEPTIONS_HANDLE();
  M << Item;//insere l'item dans le message qui va etre envoye
}

// ----------------------------------------------------------------------------
void TStack_RB::Reset( void )
{  //init et inverse sender et sendee
  M.init(M.GetSendee(), M.GetSender(), M.GetOperation(), M.GetCnstrt() );
  Stack.Reset();
  EXCEPTIONS_HANDLE();
}

// ----------------------------------------------------------------------------
void TStack_RB::GetMaxSize( void )
{  //init et inverse sender et sendee
  M.init(M.GetSendee(), M.GetSender(), M.GetOperation(), M.GetCnstrt() );
  TPE::integer result = Stack.GetMaxSize();
   EXCEPTIONS_HANDLE();
  M << result;
}

// ----------------------------------------------------------------------------
void TStack_RB::GetCurrentSize( void )
{  //init et inverse sender et sendee
  M.init(M.GetSendee(), M.GetSender(), M.GetOperation(), M.GetCnstrt() );
  TPE::integer result = Stack.GetCurrentSize();
  EXCEPTIONS_HANDLE();
  M << result;
}

// ----------------------------------------------------------------------------
void TStack_RB::GetStatus( void )
{  //init et inverse sender et sendee
  M.init(M.GetSendee(), M.GetSender(), M.GetOperation(), M.GetCnstrt() );
  TPE::integer result = TPE::integer(Stack.GetStatus());
  EXCEPTIONS_HANDLE();
  M << result;
}

// ----------------------------------------------------------------------------
void TStack_RB::X_CATCH ( void )
{  switch(EXCEPTIONS_CURRENT()) {
  case TPE::X_FULL :
    EXCEPTIONS_LOG("TStack_RB::X_CATCH","X_FULL");
    M.SetX("KO");
    M << TString(tabX_VALUES[TPE::X_FULL]);
    break;
  case TPE::X_EMPTY :
    EXCEPTIONS_LOG("TStack_RB::X_CATCH","X_EMPTY");
    M.SetX("KO");
    M << TString(tabX_VALUES[TPE::X_EMPTY]);
    break;
  default :
    EXCEPTIONS_LOG("TStack_RB::X_CATCH","X_UNDEFINED");
    M.SetX("KO");
    M << TString(tabX_VALUES[TPE::X_UNDEFINED]);
    break;
  }
}
```

```
   void SerDispatcher();
};

#endif /* STACK_RB_H_INCLUDED */

#include "hrts/HRTS_PE.h"
#include "hrts/EXCEPTIONS.h"
#include "Stack_RB.h"
// ----------------------------------------------------------------------------
TStack_RB::TStack_RB( void ) : RB_Server(TPE::STACK)
{  RB_Server.connect(TString("/tmp/HRTS/ERQSTACK"));
  ACTIVE=TPE::true;
  status = TPE::OK;
}

// ----------------------------------------------------------------------------
TStack_RB::~TStack_RB( void )
{}


void TStack_RB::Stop( void )
{  ACTIVE=TPE::false;
}

void TStack_RB::Start( void )
{  while (ACTIVE) {
    RB_Server.remove(M);
    if (M.GetSendee() == "STACK_RB") {
      SerDispatcher();
    } else {
      EXCEPTIONS_LOG("STACK_RB","INCONSISTENT OBJECT NAME");
      RB_Server.SetRtnQName( M.GetRtnQName());
      M.SetX("KO");
      RB_Server.insert(M);
      status = TPE::KO;
    }
  }
  EXCEPTIONS_LOG("STACK_RB","SERVER STOPPED");
}


void TStack_RB::SerDispatcher( void )
{/  //---------- begin of Opcs code
  RB_Server.SetRtnQName( M.GetRtnQName());
  switch ( M.GetOperation() ) {
  case TPE::PUSH :
    Push ();
    break;
  case TPE::POP :
    Pop ();
    break;
  case TPE::RESET :
    Reset ();
    break;
  case TPE::GetMaxSize :
    GetMaxSize ();
    break;
  case TPE::GetCurrentSize :
    GetCurrentSize();
    break;
  case TPE::GetStatus :
    GetStatus ();
    break;
  default :
    EXCEPTIONS_LOG("STACK_RB.SerDispatcher","INCONSISTENT OPERATION NAME");
    M.SetX("X_COMMUNICATION_ERROR");
    status = TPE::KO;
  }
  RB_Server.insert(M);
}//-------------------------


TPE::OK_KO TStack_RB::GetSerStatus( void )
{  return status;
}

// -------------------------
```

```
  } else {
        // extraction and unmarshalling of return parameters
     M >> Result;  // unmarshalling;
     return Result;
  }
}

// ---------------------------------------------------------------------------
template <class T, TPE::integer StackSize> TPE::integer  TStack<T, StackSize>::GetCurrentSize( void )
{  TPE::integer  Result;
  // init Msg header with  sender, sendee, operation  and cnstrt type
  M.init( "STACK", "STACK_RB", TPE::GetCurrentSize, TPE::NON_CNSTRT  );
  // no parameterser marshalling
  // send IPC message to server and analyze return parameters

  if (Obcs.insrem(M,TString("STACK.GetCurrentSize")) == TPE::KO ) {
       EXCEPTIONS_HANDLE();
  } else {
        // extraction and unmarshalling of return parameters
     M>>Result;  // unmarshalling;
     return Result;
  }
}

// ---------------------------------------------------------------------------
template <class T, TPE::integer StackSize> TPE::OK_KO TStack<T, StackSize>::GetStatus( void)
{  TPE::integer TmpStatus;//recupere un integer
  // init Msg header with  sender, sendee, operation  and cnstrt type
  M.init( "STACK", "STACK_RB", TPE::GetStatus, TPE::NON_CNSTRT  );
  // no parameter marshalling
  // send IPC message to server and analyze return parameters

  if (Obcs.insrem(M,TString("STACK.GetStatus")) == TPE::KO ) {
       EXCEPTIONS_HANDLE();
  } else {
        // extraction and unmarshalling of return parameters
     M>>TmpStatus;
     return TPE::OK_KO(TmpStatus);
  }
}
```

J.10.3.2 / Request Broker <ObjectName>_RB structure -- SER code illustration in C++

```
#ifndef STACK_RB_H_INCLUDED
#define STACK_RB_H_INCLUDED
#include "hrts/HRTS_PE.h"
#include "Stack_SERVER.h"
#include "hrts/TMsg.h"
#include "hrts/TServerObcs.h"

typedef TString TItemStack;
/*=============================================================================*\
* TStackServer : /                                          *
\*=============================================================================*/
class TStack_RB
{  private:
    TStack_SERVER<TItemStack, K_STACK_SIZE> Stack;
    TServerObcsRB_Server;
    TMsgM;
    TPE::Tboolean ACTIVE;
    TPE::OK_KO  status;
    void Push         ( void );
    void Pop          ( void );
    void Reset        ( void );
    void GetMaxSize    ( void );
    void GetCurrentSize( void );
    void GetStatus     ( void );
    void X_CATCH       ( void );
  public:
    TStack_RB ( void );
    ~TStack_RB( void );
    void Start( void ) ;
    void Stop( void ) ;
    TPE::OK_KO GetSerStatus( void ) ;
```

J.10.3 C++code illustrations

J.10.3.1 Client Side <ObjectName> structure -- ER code Illustration in C++

```
#ifndef STACK_H_INCLUDED
#define STACK_H_INCLUDED
#include "hrts/HRTS_PE.h"
#include "hrts/TMsg.h"
#include "hrts/TClientObcs.h"

template<class T, const TPE::integer  StackSize> class TStack {
 private:
    // note that no "application data members are needed"
   TClientObcs   Obcs;  // only HRTS data members
   TMsg          M;
 public:
   TStack ( void );
   ~TStack( void );
   void Push( const T & );
   void Pop (       T & );
   void Reset( void );
   TPE::integer GetMaxSize    ( void );
   TPE::integer GetCurrentSize( void );
   TPE::OK_KO GetStatus       ( void ) ;
};
#endif /* STACK_H_INCLUDED */

#include "Stack.h"
template <class T, TPE::integer StackSize> TStack<T, StackSize>::TStack( void )
{  Obcs.connect(TString("/tmp/HRTS/ERQSTACK"));
}

template <class T, TPE::integer StackSize> TStack<T, StackSize>::~TStack( void )
{}

template <class T, TPE::integer StackSize> void TStack<T, StackSize>::Push( const T &item )
{  // init Msg header with  sender, sendee, operation  and cnstrt type
  M.init( "STACK", "STACK_RB", TPE::PUSH, TPE::NON_CNSTRT  );
  M << item; // parameter marshalling
  // send IPC message to server and analyze return parameters
  if (Obcs.insrem(M,TString("STACK.PUSH")) == TPE::KO ) {
  EXCEPTIONS_HANDLE();
  } // no return parameter
}

template <class T, TPE::integer StackSize> void TStack<T, StackSize>::Pop( T &item )
{  // init Msg header with  sender, sendee, operation  and cnstrt type
  M.init( "STACK", "STACK_RB", TPE::POP, TPE::NON_CNSTRT  );
  if (Obcs.insrem(M,TString("STACK.POP")) == TPE::KO ) {
     EXCEPTIONS_HANDLE();
  } else {
        // extraction and unmarshalling of return parameters
     M>>item;
  }
}

// ---------------------------------------------------------------------------
template <class T, TPE::integer StackSize> void TStack<T, StackSize>::Reset( void )
{  // init Msg header with  sender, sendee, operation  and cnstrt type
  M.init( "STACK", "STACK_RB", TPE::RESET, TPE::NON_CNSTRT  );
  if (Obcs.insrem(M,"STACK.RESET") == TPE::KO ) {
       EXCEPTIONS_HANDLE();
  }
}

// ---------------------------------------------------------------------------
template <class T, TPE::integer StackSize> TPE::integer  TStack<T, StackSize>::GetMaxSize( void )
{  TPE::integer Result;
  // init Msg header with  sender, sendee, operation  and cnstrt type
  M.init( "STACK", "STACK_RB", TPE::GetMaxSize, TPE::NON_CNSTRT  );
  // no parameter marshalling
  // send IPC message to server and analyze return parameters

  if (Obcs.insrem(M,TString("STACK.GetMaxSize")) == TPE::KO ) {
       EXCEPTIONS_HANDLE();
```

J.10.1.4 VN IMPLEMENTATION ILLUSTRATION

J.10.2.5.a Package <VNName> structure -- VN code Illustration

```
package <VN Name> is
   VN_ID : HRTS_PE.TVN_ID := <VnName>; -- declare VN identification
   ALLOCTABLE : array (HRTS_PE.TVN_ID'range) of HRTS.T_OBJECT.
   FIRSTIME : array (HRTS_PE.TVN_ID'range) of HRTS_PE.Boolean : = false;
end <VN Name>;
with HRTS.VNCS; with HRTS.UDQUEUES;
package body <VNName> is
   packageVNCS is new HRTS.VNCS(     HRTS.UDQUEUES.T_QUEUE; -- UD stands for User Defined
VN_ID=> <VNName>;HRTS.UDQUEUES.create;, HRTS.UDQUEUES.Insert;
HRTS.UDQUEUES.Remove);
   -- local objects
   package <ObjectName1> is --OPCS_ER code for local active objects see sections above
   package <ObjectName1_SER> is --OPCS_SER code for local active objects see sections above
   package <ObjectName1_SERVER> is --OPCS_ER code for local active objects see sections above
   • • • • •
   --remote objects
   package <remoteObjectNamex> is --OPCS_ER code for client stubs objects see sections below
   • • • • •
   package <remotelycalled ObjectNamey> is --OPCS_SER code for server stubs objects see sections below
   • • • • •
   procedure Message_In(MSG : in T_MSG) is -- ServerVNCS code illustration
   begin
      case MSG.type is
      when RTN_MSG =>
         VNCS.insert(MSG.RTNQ,RTNMSG); -- insert in object RTNQ
      when IPC_MSG=>  -- just launch local OPCS_SER code
         VNCS.insert(MSG.OBJECT,MSG); -- insert in OBCS ERQ of objectname
      when others =>
               HRTS.EXCEPTIONS.LOG ("VN_ID.Message_IN", "INCONSISTENT MSG type"
               MSG.X=COMMUNICATION_ERROR;
               Message_out(MSG); --send messageback
      end case;
   exception
   when Others =>
      HRTS.EXCEPTIONS.LOG(«VN_ID.Message_In», "Others");
end Message_In;
begin
   VNCS.connect(<ObjectName> ); -- create VN incomming ERQ
-- this creates and /or provides for other VN to connect
   loop -- for ever
      VNCS.remove(MSG); - we are consumers of messages send by remote VNs.
      if MSG.VNSENDEE == VNID then -- this to check consistency
         If not  FIRSTIME(MSG.SENDER) then  --MSG.SENDER holds the name of the returnQ to SENDEE VN
            VNCS.Connect (MSG.SENDER); --connect to this queue for RTNMSG
            FIRSTIME(MSG.SENDER):= true;
         end if;
         Message_In;
      else
         HRTS.EXCEPTIONS.LOG ("<VNname.>", "INCONSISTENT OBJECT NAME"
         MSG.X=COMMUNICATION_ERROR;
         [Build RTNMSG]
         VNCS.insert(RTNMSG); --release client process immediately
      end ;
      exit on VNCS.ShutDown; -- all VNCS activities have to be stooped if set to true.
   end loop;
end <VNName>;
```

```
   end GetMaxSize;

   function GetCurrentSize (Me : in Instance) return HRTS_PE.T_Integer is
   begin
     -- OPCS BODY CODE (extracted directly from ODS fields) --
     return Me.CurrentSize;
   end GetCurrentSize;

   function GetStatus (Me : in Instance) return HRTS_PE.OK_KO is
   begin
     -- OPCS BODY CODE (extracted directly from ODS fields) --
     if  TFsm.GetState (Me.FSM) = Stack_OSTM.STOPPED  then
       return HRTS_PE.KO;
     else
       return HRTS_PE.OK;
     end if;
   end GetStatus;

 end TStack_Server;
```

J.10.1.3 Object_Server code Illustration

```
-------------------------------------------------------------------------------
-- STACK_SERVER code sample
-------------------------------------------------------------------------------
with HRTS_PE; use type HRTS_PE.T_Integer;
with Stack_OSTM;
package TStack_Server is
  type Instance is tagged private;
  procedure Start (Me : in out Instance);
  procedure Stop (Me : in out Instance);
  procedure Push (Me : in out Instance;  MyItem  : in HRTS_PE.T_Integer);
  procedure Pop  (Me : in out Instance;    AnItem  : out HRTS_PE.T_Integer);
  function GetMaxSize (Me : in Instance) return HRTS_PE.T_Integer;
  function GetCurrentSize (Me : in Instance) return HRTS_PE.T_Integer;
  function GetStatus (Me : in Instance) return HRTS_PE.OK_KO;
  private
    StackSize : constant HRTS_PE.T_Integer := 20;
   type T_StackBuffer is array (0 .. StackSize - 1) of HRTS_PE.T_Integer;
   type Instance is tagged record
      CurrentSize          : HRTS_PE.T_Integer := 0;
      StackBuffer          : T_StackBuffer;
      FSM                  : Stack_OSTM.TFsm.Instance := Stack_OSTM.Stack_FSM;
    end record;
end TStack_Server;

with Stack_OSTM;
package body TStack_Server is
  package TFsm renames Stack_OSTM.TFsm;
  procedure Start (Me : in out Instance) is
  begin
    -- OPCS HEADER part (automatically generated) -----------
   TFsm.Fire (Me.FSM, Stack_OSTM.START);
    -- OPCS BODY CODE (extracted directly from ODS fields) --
    Me.CurrentSize := 0;
    -- OPCS FOOTER part (automatically generated) -----------
  end Start;

  procedure Stop (Me : in out Instance) is
  begin
    -- OPCS HEADER part (automatically generated) -----------
   TFsm.Fire (Me.FSM, Stack_OSTM.STOP);
    -- OPCS BODY CODE (extracted directly from ODS fields) --
    -- OPCS FOOTER part (automatically generated) -----------
  end Stop;

  procedure Push (Me : in out Instance;                MyItem  : in HRTS_PE.T_Integer) is
  begin
    -- OPCS HEADER part (automatically generated) -----------
    TFsm.Fire (Me.FSM, Stack_OSTM.PUSH);
    -- OPCS BODY CODE (extracted directly from ODS fields) --
    Me.StackBuffer (Me.CurrentSize) := MyItem;
    Me.CurrentSize := Me.CurrentSize + 1;
    if ( Me.CurrentSize = StackSize ) then
       TFsm.Set (Me.FSM, Stack_OSTM.FULL);
    end if;
    -- OPCS FOOTER part (automatically generated) -----------
  end Push;

  procedure Pop  (Me : in out Instance;                AnItem  : out HRTS_PE.T_Integer) is
  begin
    -- OPCS HEADER part (automatically generated) -----------
   TFsm.Fire (Me.FSM, Stack_OSTM.POP);
    -- OPCS BODY CODE (extracted directly from ODS fields) --
    Me.CurrentSize := Me.CurrentSize - 1;
    AnItem := Me.StackBuffer (Me.CurrentSize);
    if ( Me.CurrentSize = 0 ) then
      TFsm.Set (Me.FSM, Stack_OSTM.EMPTY);
    end if;
    -- OPCS FOOTER part (automatically generated) -----------
  end Pop;

  function GetMaxSize (Me : in Instance) return HRTS_PE.T_Integer is
  begin
    -- OPCS BODY CODE (extracted directly from ODS fields) --
    return StackSize;
```

```
            TMsg.SetX (Message, HRTS_PE.X_UNKNOWN_ERROR);
    end;
  end  RB_Dispatcher_HSER ;
procedure RB_Dispatcher_LSER (aMessage in out TMsg.TMsg)  is
begin
      case TMsg.GetOperation (aMessage) is
         when Stack_PE.START          =>       Start ;
         when Stack_PE.STOP       =>       Stop;
         when Stack_PE.PUSH       =>       Push;
         when Stack_PE.POP        =>       Pop;
         when Stack_PE.GETMAXSIZE   =>    GetMaxSize;
         when Stack_PE.GETCURRENTSIZE   => GetCurrentSize;
         when Stack_PE.GETSTATUS   => GetStatus;
         when others         =>raise HRTS_PE.E_UNKNOWN_OPERATION;
      end case;
      exception
         when HRTS_PE.E_BadExecutionRequest => EXCEPTIONS.LOG(
              'TStack_RB.Dispatcher_LSER'& HRTS_PE.T_X_VALUE'image (HRTS_PE.X_BADREQUEST)  );
              TMsg.SetX (Message, HRTS_PE.X_BadExecutionRequest);
         when HRTS_PE.E_UNKNOWN_OPERATION =>  EXCEPTIONS.LOG(
              'TStack_RB.Dispatcher_LSER'& HRTS_PE.T_X_VALUE'image (X_UNKNOWN_OPERATION)  );
              TMsg.SetX (Message, HRTS_PE.X_UNKNOWN_OPERATION);
         when others =>   EXCEPTIONS.LOG(
           'TStack_RB.Dispatcher_LSER'»&HRTS_PE.T_X_VALUE'image(HRTS_PE.X_UNKNOWN_ERROR) );
           TMsg.SetX (Message, HRTS_PE.X_UNKNOWN_ERROR);
    end;
   end  RB_Dispatcher_LSER ;

task body OBCSServer is
begin   -- at package elaboration
  loop
     accept ExecuteRequest (Message : in out TMsg.TMsg) do--TServerObcs.Remove (OBCS, Message);
        if TMsg.getSender(Message) /=Stack_PE.STACK then
             raise HRTS_PE.E_UNKNOWN_SENDEE;
        end if;
        Params:=TMsg.GetParams(Message);
        Constraint:=TMsg.GetCnstrt(Message);
        Operation:=TMsg.getOperation(Message);
        case Constraint is
        whenHRTS_PE.HSER=> RB_Dispatcher_HSER(Message); IPCFormat(Message);
        whenHRTS_PE.LSER=>  IPCFormat(Message);RB_Dispatcher_LSER(Message);
        whenHRTS_PE.RASER=>null; --TBS;
        whenHRTS_PE.RLSER=>null; --TBS;
         when others     =>   EXCEPTIONS.LOG(
              'TStack_RB.Dispatcher_ASER'& HRTS_PE.T_X_VALUE'image (X_UNKNOWN_CONSTRAINT)  );
              TMsg.SetX (Message, HRTS_PE.X_UNKNOWN_CONSTRAINT);
      end case;
     end ExecuteRequest;
end loop;
end OBCSServer;
 end TStack_RB;
```

```
package body TStack_RB is
    TheStack      : TStack_Server.Instance;-- stack declaration :
    Params : TParams.PtrInstance;
    Constraint : HRTS_PE.T_Constraint;
    Operation : HRTS_PE.T_HOODOperation;
 --OBCS          : TServerOBCS.TServerOBCS; --not needed here
 procedure Start  is  -- OPCS_SER code
 begin
   TStack_Server.Start(TheStack);
 end  Start ;
procedure Stop  is-- OPCS_SER code
 begin
   TStack_Server.Stop(TheStack);
 end Stop;
procedure Push  is-- OPCS_SER code
    Item          : HRTS_PE.T_Integer;
    StringItem    : String( 1 .. HRTS_PE.T_Integer'width);
 begin
    TParams.Read(Params,StringItem);
    item:=HRTS_PE.T_Integer'value(StringItem);
    TStack_Server.Push (TheStack, Item);
 end Push;
procedure Pop is-- OPCS_SER code
  Item          : HRTS_PE.T_Integer;
 begin
    TStack_Server.Pop (TheStack, Item);
    TParams.Write(Params,HRTS_PE.T_Integer'image(Item);
 end Pop;
procedure GetCurrentSize  is
  CurrentSize    : HRTS_PE.T_Integer;
begin
   CurrentSize := TStack_Server.GetCurrentSize (TheStack);
    TParams.Write(Params,HRTS_PE.T_Integer'image(CurrentSize);
 end GetCurrentSize;
procedure GetMaxSize  is-- OPCS_SER code
 MaxSize       : HRTS_PE.T_Integer;
 begin
    MaxSize := TStack_Server.GetMaxSize (TheStack);
    TParams.Write(Params,HRTS_PE.T_Integer'image(MaxSize);
 end  GetMaxSize;
procedure GetStatus  is
  Status        : HRTS_PE.OK_KO;
begin
    Status := TStack_Server.GetStatus (TheStack);
    TParams.Write(Params,HRTS_PE.OK_KO'image(Status);
 end  GetMaxSize;

procedure IPCMsgFormat(aMessage : in out TMsg.TMsg) is  --common code to all operations
  PreviousSender : Stack_PE.T_HOODObject := TMsg.GetSender (Message);
begin
    TMsg.SetSender (aMessage, TMsg.GetSendee (Message));
    TMsg.SetSendee (aMessage, PreviousSender);
    PtrStream      := TMsg.GetParams (Message);
    TMsg.FlushParams (Message); -- enforce writing of parameters in the stream
end  IPCMsgFormat;

procedure RB_Dispatcher_HSER (aMessage in out TMsg.TMsg)  is
begin
     case TMsg.GetOperation (aMessage) is
       when Stack_PE.START          =>        Start ;
       when Stack_PE.STOP       =>        Stop;
       when Stack_PE.PUSH       =>        Push;
       when Stack_PE.POP        =>        Pop;
       when Stack_PE.GETMAXSIZE   =>    GetMaxSize;
       when Stack_PE.GETCURRENTSIZE   => GetCurrentSize;
       when Stack_PE.GETSTATUS   => GetStatus;
       when others         =>raise HRTS_PE.E_UNKNOWN_OPERATION;
     end case;
     exception
       when HRTS_PE.E_BadExecutionRequest => EXCEPTIONS.LOG(
            'TStack_RB.Dispatcher_HSER'& HRTS_PE.T_X_VALUE'image (HRTS_PE.X_BADREQUEST)  );
            TMsg.SetX (Message, HRTS_PE.X_BadExecutionRequest);
       when HRTS_PE.E_UNKNOWN_OPERATION =>  EXCEPTIONS.LOG(
            'TStack_RB.Dispatcher_HSER'& HRTS_PE.T_X_VALUE'image (X_UNKNOWN_OPERATION)  );
            TMsg.SetX (Message, HRTS_PE.X_UNKNOWN_OPERATION);
       when others =>   EXCEPTIONS.LOG(
         'TStack_RB.Dispatcher_HSER'»&HRTS_PE.T_X_VALUE'image(HRTS_PE.X_UNKNOWN_ERROR) );
```

```
   end Push;

   procedure Pop  (Me : in out Instance;  AnItem  : out HRTS_PE.T_Integer) is
   begin -- OPCS_ER code
    TMsg.Initialize     (     Me         => Me.Message,      Sender      => Stack_PE.STACK,
                              Sendee     => Stack_PE.STACK_RB,      Operation   => Stack_PE.POP,
                              Cnstrt     => HRTS_PE.HSER,      ParamSize   => 1      );
     IPCProcess (Me);
     Params:=TMsg.GetParams(Me.Message);
     TParams.Read(Params,StringParam);
     AnItem:=HRTS_PE.T_Integer'value (StringParam);
    end Pop;

   function GetMaxSize (Me : in Instance) return HRTS_PE.T_Integer is -- OPCS_ER code
   Size          : HRTS_PE.T_Integer;
   begin
     TMsg.Initialize     (     Me         => Me.Message,      Sender      => Stack_PE.STACK,
                               Sendee     => Stack_PE.STACK_RB,      Operation   => Stack_PE.GETMAXSIZE,
                               Cnstrt     => HRTS_PE.HSER,      ParamSize   => 1      );
     IPCProcess (Me);
     Params:=TMsg.GetParams(Me.Message);
     TParams.Read(Params,StringParam);
     Size:=HRTS_PE.T_Integer'value (StringParam);
    return Size;
   end GetMaxSize;

   function GetCurrentSize (Me : in Instance) return HRTS_PE.T_Integer is -- OPCS_ER code
   Size          : HRTS_PE.T_Integer;
   begin
    TMsg.Initialize     (     Me         => Me.Message,      Sender      => Stack_PE.STACK,
                              Sendee     => Stack_PE.STACK_RB,      Operation   => Stack_PE.GETCURRENTSIZE,
                              Cnstrt     => HRTS_PE.NO_CONSTRAINT,      ParamSize   => 1      );
     IPCProcess (Me);
     Params:=TMsg.GetParams(Me.Message);
     TParams.Read(Params,StringParam);
     Size:=HRTS_PE.T_Integer'value (StringParam);
    return Size;
   end GetCurrentSize;

   function GetStatus (Me : in Instance) return HRTS_PE.OK_KO is -- OPCS_ER code
     StringParam : String(1 .. HRTS_PE.OK_KO'width);
  Status          : HRTS_PE.OK_KO;
   begin
    TMsg.Initialize     (     Me         => Me.Message,      Sender      => Stack_PE.STACK,
                              Sendee     => Stack_PE.STACK_RB,      Operation   => Stack_PE.GETSTATUS,
                              Cnstrt     => HRTS_PE.NO_CONSTRAINT,      ParamSize   => 1      );
     IPCProcess (Me);
     Params:=TMsg.GetParams(Me.Message);
     TParams.Read(Params,StringParam);
     Statuse:=HRTS_PE.T_Integer'value (StringParam);
     return Status;
   end GetStatus;
  end TStack;
```

J.10.1.2 Request Broker <ObjectName>_RB structure -- SER code illustration

```
  with TMsg;
  package TStack_RB is
      procedure Start ;
      procedure Stop ;
      procedure Push :
      procedure Pop :
      procedure GetSatus;
      procedure GetMAxSize;
      procedure GetCurrentSize;
      procedure GetSatus;
      task OBCSServer is
          entry ExecuteRequest(message : in out TMsg.Msg);
      end OBCSServer;
  end TStack_RB;

  with HRTS_PE; use type HRTS_PE.T_Integer;
  with Stack_PE;with Stack_OSTM;
  with TMsg;with TQPool;
  with TStack_Server;
```

J.10 TARGET Code Illustrations FOR PROTOCOL CONSTRAINED OPERATIONS

J.10.1 Ada code illustrations

J.10.1.1 Client Side <ObjectName> structure -- ER code Illustration

```
with HRTS_PE; use type HRTS_PE.T_Integer;
with TMsg;with TClientObcs;
package TStack is
  type TStack is private;-- NO TAGGED TYPE HERE  since TStack is a «client stub »
  type Instance is access TStack;
  procedure Start (Me : in out Instance);
  procedure Stop (Me : in out Instance);
  procedure Push (Me : in out Instance; MyItem  : in HRTS_PE.T_Integer);
  procedure Pop (Me : in out Instance; AnItem  : out HRTS_PE.T_Integer);
  function GetMaxSize (Me : in Instance) return HRTS_PE.T_Integer;
  function GetCurrentSize (Me : in Instance) return HRTS_PE.T_Integer;
  function GetStatus (Me : in Instance) return HRTS_PE.OK_KO;
  private
    type TStack is record
      OBCS              : TClientOBCS.TClientOBCS;
      Message           : TMsg.TMsg;
    end record;
end TStack;

with Stack_PE;
package body TStack is
     Params : TParams.PtrInstance;
     StringParam : String(1 .. HRTS_PE.T_Integer'width);
  procedure IPCProcess  (Me : in Instance) is
  Status        : HRTS_PE.T_X_VALUE;
  begin
     TClientObcs.Insrem (Me.OBCS, Me.Message); -- insert message then remove returned message :
     Status := TMsg.GetX (Me.Message);
     case Status is
     when HRTS_PE.X_OK =>       null;
     when HRTS_PE.X_KO =>       raise HRTS_PE.E_KO;
     when HRTS_PE.X_UNKNOWN_SENDER =>       raise HRTS_PE.E_UNKNOWN_SENDER;
     when HRTS_PE.X_UNKNOWN_SENDEE =>       raise HRTS_PE.E_UNKNOWN_SENDEE;
     when HRTS_PE.X_UNKNOWN_OPERATION =>  raise HRTS_PE.E_UNKNOWN_OPERATION;
     when HRTS_PE.X_BADREQUEST =>              raise HRTS_PE.E_BADREQUEST;
     when HRTS_PE.X_FSMERROR =>              raise HRTS_PE.E_FSMERROR;
     when HRTS_PE.X_OBCS_NOMOREQUEUES => raise HRTS_PE.E_OBCS_NOMOREQUEUES;
     when HRTS_PE.X_UNKNOWN_ERROR =>       raise HRTS_PE.E_UNKNOWN_ERROR;
    end case;
  end IPCProcess;

-- public subprograms :
  procedure Start (Me : in out Instance) is
  begin
    TMsg.Initialize     (     Me        => Me.Message,     Sender     => Stack_PE.STACK,
      Sendee     => Stack_PE.STACK_RB,     Operation   => Stack_PE.START,
      Cnstrt     => HRTS_PE.NO_CONSTRAINT,     ParamSize   => 0      );
    IPCProcess (Me);
  end Start;
  procedure Stop (Me : in out Instance) is -- OPCS_ER code
  begin
    TMsg.Initialize     (     Me        => Me.Message,     Sender     => Stack_PE.STACK,
      Sendee     => Stack_PE.STACK_RB,     Operation   => Stack_PE.STOP,
      Cnstrt     => HRTS_PE.NO_CONSTRAINT,     ParamSize   => 0      );
    IPCProcess (Me);
  end Stop;

  procedure Push (Me : in out Instance;              MyItem  : in HRTS_PE.T_Integer) is
  begin -- OPCS_ER code
    TMsg.Initialize     ( Me   => Me.Message,     Sender => Stack_PE.STACK,
      Sendee     => Stack_PE.STACK_RB,     Operation   => Stack_PE.PUSH,
      Cnstrt     => HRTS_PE.NO_CONSTRAINT, ParamSize   => 1  );
      Params:=TMsg.GetParams(Me.Message);
     Tparams.write(Params,HRTS_PE.T_Integer'image(MyItem);
     IPCProcess (Me);
```

J.9.2 HRT_ SCHEDs Sample Code

to be supplied by HRTS Working Group

- timedout/deadline : defines the upper limit of the time window **with**in which the requested CHILD operation may execute. Hence the time window is given by 0 (now) and timedout value.
- scheduling specific information : similar to that of ROOT operation. If this feature is supported, then priority inheritance can be achieved.

Thus the ODS of HRT_SCHED can be **define**d as :

J.9.1 Object HRT_ SCHEDs IS

DESCRIPTION
Scheduler to handle ROOT operation execution threads according to scheduling information as **define**d e.g by HRT-HOOD.
PROVIDED_INTERFACE
TYPES
T_ROOTOId **is** (State1, State2, • • • Staten);
-- ROOT or MASTER Operation Id, eg. address
T_ExecTime
--floatr.
T_Priority
--integer defining the priority of execution threads.
T_PrecOnstr;
precedence constraints tbd.
T_Critically;
-tbd.
T_SCHEDINFO **is record**--**type** need for ROOT OP scheduling requests.
period : tbd;
offset: tbd;
timeout: tbd;
WCET: tbd;
priority : tbd; -- scheduling sepcific information
ceiling priority : tbd;
precedence constraints : tbd;
execution time transformation : tbd;
importance : tbd;
end record;
T_ERINFO **is record**--**type** need for CHILD ER
timeout: tbd;
priority : tbd;-- scheduling sepcific information
ceiling priority : tbd;
precedence constraints : tbd;
execution time transformation : tbd;
importance : tbd;
end record;
OPERATIONS
TRA (ROOTOpId : **in** T_MOId; SchedInfo : **in** T_SchedInfo)
Transfer scheduling information associated to a given ROOT operation to the scheduler.
REQUIRED_INTERFACE
types
INTERNALS --hidden part:
--to be supplied by HRT Working Group.
END OBJECT HRT_SCHEDs.

- All ER types which have an equivalent in HOOD4 and can respond like ASATC and LSTAC.

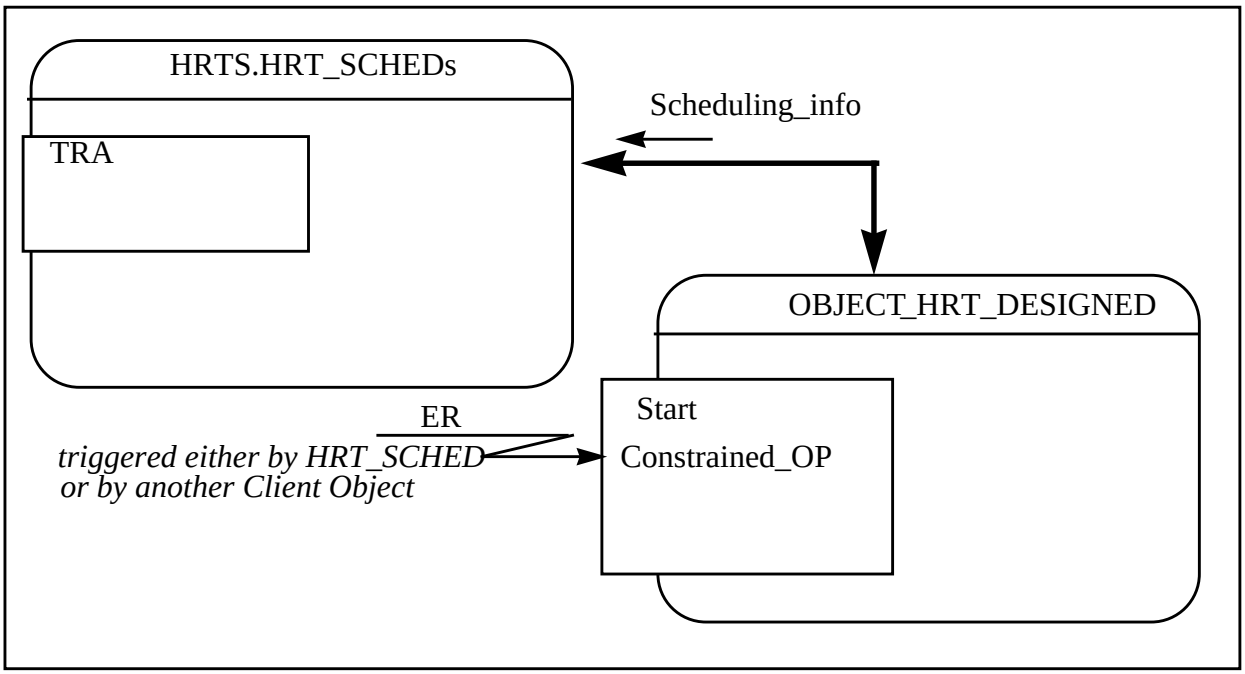


Figure 97 -Relationships between a HRT designed object and the HRT_SCHEDULER

In the following we outline the real-time attributes to be defined **with**in ROOT OPERATION and provided to the HRT_SCHEDuler :

- period : in case of sporadically scheduled operations an equivalent period (e.g minimum or average period) shall be provided to calculate an equivalent workload.
- offset : defines the lower limit of the time window **with**in which an operation may execute
- timedout/deadline : defines the upper limit of the time window **with**in which an operation may execute.
  Offset and deadline define the allowed time window **with** respect to the period.
- start time : define the time at which the ROOT operation shall be executed for the first time
- WCET : worst case execution time
- scheduling specific information : this is additional information related to the operation (to be assigned at compilation or at runtime). This information depends on the scheduling algorithm and **include**s, e.g; priority. AN ER may override some of these data if supported.

Now we have to define the real-time attributes managed at normal execution Request and threads :

J.9 OBJECT HRT_SCHEDs

Object HRT_SCHEDs is a software module for support of HRT_HOOD [HRTOSK] real-time attributes management.

HRT-HOOD introduced new kind of objects (cyclic, sporadic, protected) including new real-time information in the ODS field «Real-Time Attributes». The latter shall be summarized in *Appendix J - HOOD RUN TIME SUPPORT LIBRARY* - of present document .

In the following we suggest an approach to implement HRT-HOOD designs using a HRT_SCHED object instanciated in HOOD4.

HRT-HOOD implicitly introduced a «scheduler», scheduling cyclic operations, whereas sporadic operations are either triggered by the scheduler or by other external events. Scheduling of a thread occurs only if the top-level or ROOT operation is triggered by an execution request triggered by the scheduler. Thus one can distinguish :

- ROOT_OPERATIONS which themselves provide information on how they have to be scheduled. Such operations shall ask to the HRT_SCHED an execution request.
- CHILD_OPERATION which are just awaiting an ER, but never ask it to the sheduler. They are providing their code to every client thread accessing them for execution.
- the HRT_SCHEDuler is:
  - an execution thread that issues ER previously requested by ROOT OPERATIONS
  - harmonises/schedules ERs from several ROOT operations
  - possibly controls execution resources, i.e. may preempty an executing thread in an operation, if a register ER has the right to execute.

In order to be flexible enough, a general scheduler shall not know the objects/operation it uses. Therefore ROOT operations must provide scheduling information, in order to be scheduled. One way to implement this information exchange is by having an object **with** ROOT OPERATIONs provide their scheduling information to the scheduler at creation or initialization time. In the following we give a possible specification of an HRT-SCHEDULER, first implementation shall find out whether this object should be a generic one and be instanciated **with**in each HRT object, or if it may be unique for a full HRT design.

HRT-HOOD also introduces so-called «Asynchronous Execution request» which considers the possibility that a constrained operation could be blocked as :

- PSER : protected synchronous execution request
- ASATC : asynchronous-asynchronous transfer of control request
- LSATC : loosely synchronised asynchronous transfer of control request

J.8.2 HRTS.TIMER sample code

```
#ifndef TIMER_H_INCLUDED
#define TIMER_H_INCLUDED
#include <sys/time.h>
class TTimer {
  public:
    TTimer ( void );
    ~TTimer( void );
    void Set( long );
    TBoolean TimeOutReached( void ) const;
    long     GetTimeLeft   ( void ) const;
  private:
    struct timeval SetTime;
    long           TimeOut;
};
#endif /* TIMER_H_INCLUDED */

#include <sys/time.h>// BODY OF OBJECT : TTimer
 #include <sys/time.h>
// ========== visibility on specification file :
#include "TTimer.h"
// ========== visibility on required objects :
#include "time.h"
// ---------- provided operation bodies :
TTimer::TTimer( )
 {//---------- begin of Opcs code
     SetTime.tv_sec  = 0L;
     SetTime.tv_usec = 0L;
     TimeDelay       = 0 ;
}//--------------------------
 TTimer::~TTimer( ) {}
boolean TTimer::TimeOut( ) const
 {//---------- begin of Opcs code
     if ( GetTimeleft() == 0 ) {
         return Tproject::TRUE;
     } else {
         return Tproject::FALSE;
     }
}//-------------------------

void TTimer::Set(/* IN */ const long Delay )
 {//---------- local Opcs variables
     struct timezone tzp;
//---------- begin of Opcs code
// On recupere l'heure courante ( en secondes et micro secondes )
  (void)gettimeofday( &SetTime, &tzp );
 // On sauvegarde la duree demandee avant le time out
     TimeDelay = ( Delay < 0 ) ? 0 : Delay;
}//---------------------------------

long TTimer::GetTimeleft( ) const
 {//---------- local Opcs variables
     struct timeval  tp    ;
      struct timezone tzp   ;
      long           Result;
//---------- begin of Opcs code
 // On recupere l'heure courante ( en secondes et micro secondes )
     (void)gettimeofday( &tp, &tzp );
// On calcule le temps restant avant le time out ( en prenant en compte
// les eventuels overflows ).
     if (( tp.tv_sec - SetTime.tv_sec ) > ( LONG_MAX / 1000 + 1 )) {
         Result = 0;
     } else {
         Result = TimeDelay                          -
           (( tp.tv_sec  - SetTime.tv_sec  ) * 1000 +
            ( tp.tv_usec - SetTime.tv_usec ) / 1000 );
          return ( Result < 0 ) ? 0 : Result;
     }
}//--------------------------------
// END OF OBJECT : TTimer
```

J.8 Object TIMER

TTimer is an implementation of timer abstract type as a standardised interface between HOOD designs and specific target OS process synchronisation primitives. **Class** TTIMER standardised facilities to declare, initiate, set and testa timer value.

J.8.1 Object TIMER IS ACTIVE

DESCRIPTION
Implementation of an abstract data **type** for TIMERs management.
IMPLEMENTATION_CONSTRAINTS
Implementation shall use the facilities provided by a RTL library or directly host OS
PROVIDED_INTERFACE
TYPES
T_TIMER **is private**;
OPERATIONS
SET(name : **in** TString; asemaphore : **out** T_Semaphore);
Creates and initializes a semaphore associated to the string name.If the underlying Host OS services **return**s an error, the exception X_HostError **is** raised.
TIMED_OUT(name : **in** TString) **return boolean;**
release the semaphore. If no match **with** the name **is** found, exception X_BadSemaphoreName **is** raised.
EXCEPTIONS
X_BadSemaphoreName raised by V or P
X_HOSError raised by P, create , delete
REQUIRED_INTERFACE
OBJECT HRTS_PE
types
T_String;
--HRTS string **type** possibly hiding an implmenation **define**d one.
OBJECT RTL -- traget host OS services
Operations
OS sempahore operations.
INTERNALS -- hidden part of the object:
--to be supplied by HRTS Working Group.
END Object Timers.

J.7.2 HRTS.SEMAPHORE sample code

```
// SPEC OF OBJECT : TSemaphore

#ifndef INCLUDE_spec_TSemaphore
#define INCLUDE_spec_TSemaphore
// ========== visibility on required objects :
#include "sys/types.h"
// ---------- provided types and constants :
class TSemaphore
{private:
    int SemId;
public:
    int V();
    int P();
    TSemaphore();
    TSemaphore(const key_t ident, const int level);
    ~TSemaphore();
};
// END OF OBJECT : TSemaphore
#endif

// BODY OF OBJECT : TSemaphore
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
// ========== visibility on specification file :
#include "TSemaphore.h"
// ---------- provided operation bodies :
int TSemaphore::V()
{//---------- local Opcs variables
    static struct sembuf op[] = {{ 0, 1, 0 }};
//---------- begin of Opcs code
    return semop( SemId, op, 1 );
}//----------------------------

int TSemaphore::P()
{//---------- local Opcs variables
    static struct sembuf op[] = {{ 0, -1, 0 }};
//---------- begin of Opcs code
     return semop( SemId, op, 1 );
}//----------------------------

TSemaphore::TSemaphore(const key_t ident, const int level)
{//---------- local Opcs variables
    union semun arg;
//---------- begin of Opcs code
    SemId = semget( ident, 1, IPC_CREAT | 0666 );// Creation du semaphore
    arg.val = level;                // Initialisation de la valeur du semaphore
     (void)semctl( SemId, 0, SETVAL, arg );
}//------------------------------


TSemaphore::~TSemaphore0()
{//---------- local Opcs variables
    union semun arg;
//---------- begin of Opcs code
     semctl( SemId, 0, IPC_RMID, arg );
}//--------------------------
// END OF OBJECT : TSemaphore
```

J.7 Object Semaphores

Semaphores is implementation of semaphore type as a standardised interface between HOOD designs and specific target OS process synchronisation primitives. **Class** TSem provides standardised facilities to declare, initiate, request and release a semaphore.

J.7.1 Object SEMAPHOREs IS ACTIVE

DESCRIPTION
Implementation of an abstract data **type** for mutual exclusion management.
IMPLEMENTATION_CONSTRAINTS
Implementation should take care to hanlde automatic creation of semaphore **in** order to provide a clean interface at client level. Alternatively implementation should minimize the overhead associated to name handling and «automatic» creation of sempahores.
PROVIDED_INTERFACE
TYPES
T_SEMAPHORE **is private**;
OPERATIONS
Create(name : **in** TString; asemaphore : **out** T_Semaphore);
Creates and initializes a semaphore associated to the string name.If the underlying Host OS services **return**s an error, the exception X_HOSError **is** raised.
V(name : **in** TString);
release the semaphore. If no match **with** the name **is** found, exception X_BadSemaphoreName **is** raised.
P(name : **in** TString);
seize the semaphore. If it **is** the first time the semaphore **is** addressed, it **is** created. If the underlying Host OS services **return**s an error, the exception X_HOSError **is** raised.
Delete(asemaphore : **in** T_Semaphore);
deletes the semaphore associated to the name.If the underlying Host OS services **return**s an error, the exception X_HOSError **is** raised.
EXCEPTIONS
X_BadSemaphoreName raised by V or P
X_HOSError raised by P, create , delete
REQUIRED_INTERFACE
OBJECT HRTS_PE
types
T_String;
--HRTS string **type** possibly hiding an implmenation **define**d one.
OBJECT RTL -- traget host OS services
Operations
OS sempahore operations.
INTERNALS -- hidden part of the object:
--to be supplied by HRTS Working Group.
END OBJECT Semaphores.

J.6.3 HRTS.HRTS_PE sample code in Ada

```
// SPEC OF OBJECT : HRTS_PE
#ifndef INCLUDE_spec_HRTS_PE
#define INCLUDE_spec_HRTS_PE
// ---------- provided types and constants :
#include <rw/cstring.h>
#include <fstream.h>

class TPE
{public:
 TPE();
 ~TPE();
// Redefinition of basic types in order to have more portable
// application code
 typedef int integer;
 typedef float real;


 enum Tboolean
 {   hrtsfalse,   hrtstrue
 };
 enum T_ON_OFF
 {   ON,   OFF
 };
 enum T_TRACE{
  TON,
  TOFF,
  GRAPHIC
 };

 enum OK_KO
 {   KO=-1,   OK
 };  // standard for checkin failure upon function return
//
//----CONSTRAINTS VALUES
 enum T_HOODCSTRNT
 {// Protocol constraints
  HSER,
  LSER,
  ASER,
  RASER,
  RLSER,
  // TIMEOut  constraints
  TOER_HSER,
  TOER_LSER,
  TOER_RASER,
  TOER_RLSER,
  // Concurrency  constraints
  MUTEX,
  ROER,
  RWER,
  // State  constraints
  STATE_CNSTRT,
  // Non Constraints
  NON_CNSTRT,
  CsUNDEFINED
 };

};// end class
typedef TPE *pTPE;
typedef TPE & rTPE;

// standard definition of class for string support
typedef RWCString TString;

//useful constants
const TString  K_OBCS_FIFO_NAME ("/tmp/HRTS/fifoqueues");
const TPE::integer MAX_PENDING_ER=10;
// END OF OBJECT : HRTS_PE
#endif
```

J.6.2 HRTS.HRTS_PE sample code in Ada

```
 ------------------------------------------------------------------------------
 -- Hood Run Time System----       HRTS definitions which are  Application INDEPENDANT
 ------------------------------------------------------------------------------
package HRTS_PE is
  type T_Integer is new integer range -32765 .. 32766;
  type OK_KO is (OK, KO);
-- Definitions for client server mode :
  -- exceptions to be translated from status returned from server
    E_KO,
    E_UNKNOWN_SENDER,
    E_UNKNOWN_SENDEE,
    E_UNKNOWN_OPERATION,
    E_UNKNOWN_CONSTRAINT,
    E_BAD_EXECUTION_REQUEST,
    E_FSM_ERROR,
    E_OBCS_NO_MORE_QUEUES,
    E_UNKNOWN_ERROR       : exception;
    type T_Status is (-- associated status to translate into exceptions
    X_OK,
    X_KO,
    X_UNKNOWN_SENDER,
    X_UNKNOWN_SENDEE,
    X_UNKNOWN_OPERATION,
    X_UNKNOWN_CONSTRAINT,
    X_BAD_EXECUTION_REQUEST,
    X_FSM_ERROR,
    X_OBCS_NO_MORE_QUEUES,
    X_UNKNOWN_ERROR
    );
  -- other exceptions :
    E_MESSAGE_FORMAT_ERROR : exception;

  type T_Trace is (TON, TOFF, GRAPHIC);

  type T_Constraint is (  -- HOOD constraints definitions
    HSER,
    LSER,
    ASER,
    RASER,
    RLSER,
    TOER_HSER,
    TOER_LSER,
    MUTEX,
    ROER,
    RWER,
    NO_CONSTRAINT,
    UNDEFINED
    );
  -- queue pool management :
  MAX_PENDING_ER : constant T_Integer := 5;
  type T_QueueIndex is new T_integer range 0 .. MAX_PENDING_ER;
  INullQueue   : constant T_QueueIndex := 0;
end HRTS_PE;
```

J.6 OBJECT HRTS_PE

Object HRTS Project environment is a software module that defines and interfaces HRTS target dependent definition :

- object names of a project
- operation and state names for state constrained operation
- VN identifier names
- exception names
- Basic types and **class**es redefinition in order to handle the impedance mismatch of the different «predefined environment» of the targets.

J.6.1 Object HRTS_PE IS

DESCRIPTION
General placeholder for common **type**s, constant, and operations for the HOOD RUN TIME Library. Especially the HRTS_PE object contains a number of basic tools such as a cmdline and option analyzer.
PROVIDED_INTERFACE
TYPES
T_STATES **is** (ETAT1, ETAT2, • • • ETATn);
-- enumerated **type** giving all states identifier used **in** the OSTD descriptionss
T_OPERATION **is** (OPERATION1, OPERATION2, • • • •OPERATIONn);
--enumerated **type** giving all constrained operation identifier.
T_OBJECT **is** (OBJECT1, OBJECT2, • • • •OBJECTn);
--enumerated **type** giving all active object and **class**es identifier.
OPERATIONS
GETC(tbd)
operation to get a character from the UNIX shell cmdline. This operation requires operation GETCMDLINE that gets the full unix cmdline that launched the current ma**in** program.
GETOPt(tbd)
operation to the next option value from the UNIX shell cmdline. assuming a standard program call **with** options separated by yhe «-» character.
READIN(tbd)
operation to get an integer value from the UNIX standard input to be processed **in case** of the genric perform operation for doing unit testing of an object.
--TBD others as identified.
REQUIRED_INTERFACE
types
INTERNALS -- hidden part of the object:
--to be supplied by HRTS Working Group.
END OBJECT HRTS_PE.

J.5 Object VNCS

VNCS is a standardised implementation of remote interprocess communication through distributed fifo queues. **Class** TVNCS provides facilities to declare, initiate a client-server sending message queue and an associate receiving queue dedicated to handle HOOD operation request and **return** parameters.

Object GENERIC VNCS **IS**
FORMAL_PARAMETERS
TYPES
T_QUEUE
Fifo queue to be provided by the environment
OPERATIONs
IPCInsert (aMsg : **in** T_MSG; ina : **in** T_QUEUE)
Insert a message **in** a T_QUEUE queue
IPCRemove (aMSg: **in out** T_MSG; froma : **in out** T_QUEUE);
Removes a message from a T_QUEUE queue
DESCRIPTION
Implementation of a facility to setup IPC messages, s**end** them to external process using queuing software supplied by the environment, and processing incoming messages as well as **return** messages.
PROVIDED_INTERFACE
TYPES
TVNCS **is private**;
-- **class** allow client code to instantiate several such VNCSs facilities
OPERATIONS
INSERT(aMSG : **in out** T_MSG);
Allocates a RTNQ for this MSG and insert operation execution request MSG data **in** a sending VNQUEUE named VNERQ. When this operation **is** called by an OPCS_ER client code, for the first time it initializes a pool of RTNQ and creates the VNERQ.
REMOVE(QUEUE: **in** T_QUEUE; aRTNMSG : **out** T_MSG);
Removes a RTNMSG from the **return** queue allocated at INSERT time..
FREE(MSG : **in** T_MSG);
deallocates the RTNQ and the message structure MSG when all parameters have been processed..
INSERT(QUEUE: **in** T_QUEUE; aRTNMSG : **in** T_MSG);
Inserts a RTNMSG **in** an RTNQ identified by a incomming message.
Message_IN(aMSG : **in out** T_MSG)
Remove a MSG data from an IPC QUEUE.
Triggered by ServerOBCS, either through serverOBCS infinite loop or by a call back dispatch by a IPC event handler.

REQUIRED_INTERFACE
OBJECT HRTS_PE
types
Boolean, T_OPERATION, T_OBJECT;
INTERNALS -- hidden part of the object:
--to be supplied by HRTS Working Group.
END OBJECT VNCS.

J.5.1 VNCS sample code

to be supplied (implemetation currently (22/09/95) under tests)


```
   end SetParams;
   procedure Initialize    (    Me              : in out TMsg;    Sender          : in Stack_PE.T_HOODObject;
     Sendee          : in Stack_PE.T_HOODObject;    Operation       : in Stack_PE.T_HOODOperation;
     Cnstrt          : in HRTS_PE.T_Constraint;    ParamSize       : in HRTS_PE.T_Integer    ) is
   begin
     Me                    := new TMsgDesc;
     Me.X                   := HRTS_PE.X_OK;
     Me.Sender              := Sender;
     Me.Sendee              := Sendee;
     Me.RtnQId              := HRTS_PE.INullQueue;
     Me.Operation           := Operation;
     Me.Constraint          := Cnstrt;
     Me.ParamSize           := ParamSize;
     Me.ParamStream         := new TParams.Instance;
    end Initialize;
   procedure Copy    (    Me              : in TMsg;    Into            : out TMsg    ) is
   begin
     Into := Me;
   end Copy;
   procedure FlushParams    (    Me       : in out TMsg    ) is
   begin    -- to write the parameter stream in the stream file :
     TParams.Flush (Me.ParamStream);
   end FlushParams;

   procedure Free    (    Me       : in out TMsg    ) is
   begin
     null;
   end Free;
   procedure Trace    (    Me              : in TMsg    ) is--       FOR TEST PURPOSE
   begin
    EXCEPTIONS_LOG (“  X “ & HRTS_PE.T_Status’image(Me.X));
   EXCEPTIONS_LOG (“  Sender “ & Stack_PE.T_HOODObject’image (Me.Sender));
   EXCEPTIONS_LOG (“  Sendee “ & Stack_PE.T_HOODObject’image (Me.Sendee));
   EXCEPTIONS_LOG (“  RtnQ.Id “ & HRTS_PE.T_QueueIndex’image (Me.RtnQId));
    EXCEPTIONS_LOG (“  Operation “ & Stack_PE.T_HOODOperation’image (Me.Operation));
   EXCEPTIONS_LOG (“  Constraint “ & HRTS_PE.T_Constraint’image (Me.Constraint));
  EXCEPTIONS_LOG (“  ParamSize “ & HRTS_PE.T_Integer’image (Me.ParamSize));
     TParams.Trace (Me.ParamStream);
   end Trace;

 end TMsg;
```

```
    X               : HRTS_PE.T_Status := HRTS_PE.X_OK;
    Sender, Sendee  : Stack_PE.T_HOODObject;
    RtnQId          : HRTS_PE.T_QueueIndex := HRTS_PE.INullQueue;
    Operation       : Stack_PE.T_HOODOperation;
    Constraint      : HRTS_PE.T_Constraint;
    ParamSize       : HRTS_PE.T_Integer;
    ParamStream     : TParams.PtrInstance;
  end record;
  type TMsg is access TMsgDesc;
end TMsg;

with EXCEPTIONS_Log;
package body TMsg is
  function GetX (Me          : in TMsg) return HRTS_PE.T_Status is
  begin
    return Me.X;
  end GetX;
  function GetSender (Me     : in TMsg) return Stack_PE.T_HOODObject is
  begin
    return Me.Sender;
  end GetSender;
  function GetSendee (Me     : in TMsg) return Stack_PE.T_HOODObject is
  begin
    return Me.Sendee;
  end GetSendee;
  function GetRtnQId (Me     : in TMsg) return HRTS_PE.T_QueueIndex is
  begin
    return Me.RtnQId;
  end GetRtnQId;
  function GetOperation (Me  : in TMsg) return Stack_PE.T_HOODOperation is
  begin
    return Me.Operation;
  end GetOperation;
  function GetCnstrt (Me     : in TMsg) return HRTS_PE.T_Constraint is
  begin
    return Me.Constraint;
  end GetCnstrt;
  function GetParamSize (Me  : in TMsg) return HRTS_PE.T_Integer is
  begin
    return Me.ParamSize;
  end GetParamSize;
  function GetParams (Me     : in TMsg)
    return TParams.PtrInstance is
  begin
    return Me.ParamStream;
  end GetParams;
  procedure SetX (Me         : in out TMsg;          XValue    : in HRTS_PE.T_Status) is
  begin
    Me.X := XValue;
  end SetX;
  procedure SetSender (Me    : in out TMsg;          Sender    : in Stack_PE.T_HOODObject) is
  begin
    Me.Sender := Sender;
  end SetSender;
  procedure SetSendee (Me    : in out TMsg;          Sendee    : in Stack_PE.T_HOODObject) is
  begin
    Me.sendee := Sendee;
  end SetSendee;
  procedure SetRtnQId (Me    : in out TMsg;          RtnQId    : in HRTS_PE.T_QueueIndex) is
  begin
    Me.RtnQId := RtnQId;
  end SetRtnQId;
  procedure SetOperation (Me : in out TMsg;          Operation : in Stack_PE.T_HOODOperation) is
  begin
    Me.Operation := Operation;
  end SetOperation;
  procedure SetCnstrt (Me    : in out TMsg;          Cnstrt    : in HRTS_PE.T_Constraint) is
  begin
    Me.Constraint := Cnstrt;
  end SetCnstrt;
  procedure SetParamSize (Me : in out TMsg;          ParamSize : in HRTS_PE.T_Integer) is
  begin
    Me.ParamSize := ParamSize;
  end SetParamSize;
  procedure SetParams (Me    : in out TMsg;          Params    : in TParams.PtrInstance) is
  begin
    Me.ParamStream := Params;
```

J.4.3 HRTS.TMsg Ada sample code

```
--------------------------------------------------------------------------
--- package TParams
--  Specifications :
--       This package emulates the streaming (not available on the Ada implementation we used): it is used to write and read
--            parameters of any type in the message.
---  Implementation features :
--       The stream file is replaced by a buffer string.
--------------------------------------------------------------------------
package TParams is
  type Instance is tagged private;
  type PtrInstance is access all Instance;
  procedure Read (Me : access Instance; Param : in out string);
  procedure Write (Me : access Instance; Param : in string);
  procedure Flush (Me : access Instance);
  procedure Trace (Me : access Instance);
private
  type Instance is tagged record
    Stream            : string (1 .. 100) := (others => ' ');
    Index             : integer := 1;
  end record;
end TParams;

with EXCEPTIONS_Log;
package body TParams is
  procedure Read (Me : access Instance; Param : in out string) is
  begin
    Param := Me.Stream (Me.Index .. Me.Index + Param'length - 1);
    Me.Index := Me.Index + Param'length;
  end Read;
  procedure Write (Me : access Instance; Param : in string) is
  begin
    Me.Stream (Me.Index .. Me.Index + Param'length -1) := Param;
    Me.Index := Me.Index + Param'length;
  end Write;
  procedure Flush (Me : access Instance) is
  begin
    Me.Index := 1;
  end Flush;
  procedure Trace (Me : access Instance) is
  begin
    EXCEPTIONS_LOG("Params <" & Me.Stream & "> " & integer'image(Me.Index));
  end Trace;
end TParams;

with HRTS_PE;
with TParams;--  Definitions for sending and receiving messages :
package TMsg is  -- not "tagged" since it is an access type
  type TMsg is private;
  function GetX (Me          : in TMsg) return HRTS_PE.T_Status;
  function GetSender (Me     : in TMsg) return Stack_PE.T_HOODObject;
  function GetSendee (Me     : in TMsg) return Stack_PE.T_HOODObject;
  function GetRtnQId (Me     : in TMsg) return HRTS_PE.T_QueueIndex;
  function GetOperation (Me  : in TMsg) return Stack_PE.T_HOODOperation;
  function GetCnstrt (Me     : in TMsg) return HRTS_PE.T_Constraint;
  function GetParamSize (Me  : in TMsg) return HRTS_PE.T_Integer;
  function GetParams (Me     : in TMsg) return TParams.PtrInstance;
  procedure SetX (Me         : in out TMsg;          XValue    : in HRTS_PE.T_Status);
  procedure SetSender (Me    : in out TMsg;          Sender    : in Stack_PE.T_HOODObject);
  procedure SetSendee (Me    : in out TMsg;          Sendee    : in Stack_PE.T_HOODObject);
  procedure SetRtnQId  (Me   : in out TMsg;          RtnQId    : in HRTS_PE.T_QueueIndex);
  procedure SetOperation (Me : in out TMsg;          Operation : in Stack_PE.T_HOODOperation);
  procedure SetCnstrt (Me    : in out TMsg;          Cnstrt    : in HRTS_PE.T_Constraint);
  procedure SetParamSize (Me : in out TMsg;          ParamSize : in HRTS_PE.T_Integer);
  procedure SetParams (Me    : in out TMsg;          Params    : in TParams.PtrInstance);
  procedure Initialize  (
    Me          : in out TMsg;  Sender    : in Stack_PE.T_HOODObject;  Sendee    : in Stack_PE.T_HOODObject;
    Operation   : in Stack_PE.T_HOODOperation;  Cnstrt    : in HRTS_PE.T_Constraint;
    ParamSize   : in HRTS_PE.T_Integer  );
  procedure Copy  (  Me         : in TMsg;  Into      : out TMsg  );
  procedure FlushParams  (  Me  : in out TMsg  );
  procedure Free  (  Me         : in out TMsg  );
  procedure Trace  (  Me        : in TMsg  );
private
  type TMsgDesc is record
```

J.4.2 HRTS.TMsg C++ sample code

```
// SPEC OF OBJECT : TMsg
#ifndef INCLUDE_spec_TMsg
#define INCLUDE_spec_TMsg
// ========== visibility on required objects :
#include "hrts/TIOStream.h"
// ---------- provided types and constants :
class TMsg;
typedef TMsg & rMsg;
typedef TMsg * pMsg;
class TRtnQ;
class TErQ;
class TMsg: public virtual TIOStream
{private:
  TString          X        ;//exception string value;
  TString          Sender   ;//sender of current IPCMSG
  TString          Sendee   ;//sendee of current IPCMSG
  TString          RtnQName ;//rtnQ for curret request
  TPE::T_OPERATION   Operation  ;//operation of request
  TPE::T_CSTRNT      Constraint ;//contraint attached to the operation
public:
  TMsg();
  ~TMsg();
  const TString GetX() const {return X;}
  const TString GetSender() const {return Sender;}
  const TString GetSendee() const {return Sendee;}
  const TString GetRtnQName() const {return RtnQName;}
  const TPE::T_OPERATION GetOperation() const {return Operation;}
  const TPE::T_CSTRNT GetCnstrt() const {return Constraint;}
    void SetX(const TString &XStringValue) {X = XStringValue;}
  void SetSender(const TString &Name) {Sender = Name;}
  void SetSendee(const TString &Name) {Sendee = Name;}
  void SetRtnQName(const TString &Name) {RtnQName = Name;}
  void SetOperation(const TPE::T_OPERATION &item) {Operation = item;}
  void SetCnstrt(const TPE::T_CSTRNT &item) {Constraint = item;}
  void init(const TString &SenderName,
        const TString &SendeeName,
        TPE::T_OPERATION Operation,
        TPE::T_CSTRNT Constraint);
    friend class TRtnQ;
  friend class TErQ;
};
// END OF OBJECT : TMsg
#endif

#include "hrts/TMsg.h"
#include "PROJECT_PE.h"
// ========================================================================
TMsg::TMsg() :  X        ( "OK"                ),  Sender    ( ""                  ),  Sendee    ( ""                  ),
  RtnQName  ( ""                  ),  Operation ( PROJECT_PE::OpUNDEFINED ),  Constraint( TPE::CsUNDEFINED       )
{}

TMsg::~TMsg() {}

void TMsg::init( const TString  &SenderName ,              const TString  &SendeeName ,
            TPE::integer   operation   ,              TPE::integer   constraint  )
{  Operation  = operation        ;
  Sender     = SenderName       ;
  Sendee     = SendeeName       ;
  Constraint = constraint       ;
  X          = TString   ( "OK" );
  RtnQName   = TString   ( ""   );
}

void TMsg::print( const TString &source )
{  cerr << source.data() << " X="          << X.data()       << endl
      << source.data() << " Sender="     << Sender.data()   << endl
      << source.data() << " Sendee="     << Sendee.data()   << endl
      << source.data() << " RtnQName="   << RtnQName.data() << endl
      << source.data() << " Operation="  << Operation       << endl
      << source.data() << " Constraint=" << Constraint      << endl;
}
// END OF OBJECT : TMsg
```

J.4 Object TMSg

TMsg is a standardised implementation of interprocess message data structure.

J.4.1 Class Object TMsg IS

DESCRIPTION
Implementation of a IPC Message data structure,
PROVIDED_INTERFACE
TYPES
TMsg ; -- **class** defining the IPCMSG structure shared by client and server threads
OPERATIONS
GetX (Me : **in** TMsg) **return** HRTS_PE.T_Status;
get the Exception_Value from the exception field of Message
GetSender (Me : **in** TMsg) **return** Stack_PE.T_HOODObject;
get the sender value from the Sender field of Message
GetSendee (Me : **in** TMsg) **return** Stack_PE.T_HOODObject;
get the sendee value from the Sendee field of Message
GetRtnQId (Me : **in** TMsg) **return** HRTS_PE.T_QueueIndex;
get the return Queue Id value from theRTNQ field of Message
GetOperation (Me : **in** TMsg) **return** Stack_PE.T_HOODOperation;
get the operation value from the operation field of Message
GetCnstrt (Me : **in** TMsg) **return** HRTS_PE.T_Constraint;
get the constraint value from the constraint field of Message
GetParamSize (Me : **in** TMsg) **return** HRTS_PE.T_Integer;
get the number of parameter from the paramSize field of Message
GetParams (Me : **in** TMsg) **return** TParams.PtrInstance;
get the parameter value from the params field of Message
SetX (Me : **in out** TMsg; XValue : **in** HRTS_PE.T_Status);
set the Exception_Value in the exception field of Message
SetSender (Me : **in out** TMsg; Sender : **in** Stack_PE.T_HOODObject);
set the sender value in the Sender field of Message
SetSendee (Me : **in out** TMsg; Sendee : **in** Stack_PE.T_HOODObject);
set the sendee value in the Sendee field of Message
SetRtnQId (Me : **in out** TMsg; RtnQId : **in** HRTS_PE.T_QueueIndex);
set the return Queue Id value in theRTNQ field of Message
SetOperation (Me : **in out** TMsg; Operation : **in** Stack_PE.T_HOODOperation);
set the operation value in the operation field of Message
SetCnstrt (Me : **in out** TMsg; Cnstrt : **in** HRTS_PE.T_Constraint);
set the constraint value in the constraint field of Message
SetParamSize (Me : **in out** TMsg; ParamSize : **in** HRTS_PE.T_Integer);
set the number of parameter from the paramSize field of Message
SetParams (Me : **in out** TMsg; Params : **in** TParams.PtrInstance);
set the parameter value in the params field of Message
Initialize (
Me : **in out** TMsg; Sender : **in** Stack_PE.T_HOODObject; Sendee : **in** Stack_PE.T_HOODObject;
Operation : **in** Stack_PE.T_HOODOperation; Cnstrt : **in** HRTS_PE.T_Constraint;
ParamSize : **in** HRTS_PE.T_Integer);
initialize a message structure
Copy (Me : **in** TMsg; Into : **out** TMsg);
copy one TMSg into Another
FlushParams (Me : **in out** TMsg);
get the parameter value from the params field of Message
Free (Me : **in out** TMsg);
gfree an allocated message
Trace (Me : **in** TMsg);
trace down the contenets of a message
EXCEPTIONS NONE
REQUIRED_INTERFACE
OBJECT HRTS_PE **types**
Booelan, T_OPERATION, T_OBJECT, T_X_VALUE, T_CNSTRT;
INTERNALS -- hidden part of the object:
--to be supplied by HRTS Working Group.
END OBJECTTMsg.


```
typedef TServerObcs * pServerObcs;
typedef TServerObcs & rServerObcs;
// END OF OBJECT : TServerObcs
#endif
// BODY OF OBJECT : TServerObcs
// ========== visibility on specification file :
#include "hrts/TServerObcs.h"
#include "hrts/EXCEPTIONS.h"
// NESTED HOOD OBJECT : TServerObcs
// ---------- internal types and constants :
// ---------- internal data :
// NESTED HOOD OBJECT : TServerObcs
// ---------- internal data :
// ---------- provided operation bodies :

TServerObcs::TServerObcs(TPE::T_OBJECT index) :rtnName("")
{//---------- begin of Opcs code
   ACTIVE = TPE::false;
   erqName="/tmp/HRTS/";
   erqName=erqName+tabOBJECTS[index];
}//-------------------------

TServerObcs::~TServerObcs()
{}//-------------------------

void TServerObcs::connect(TString &fifoname)
{//---------- begin of Opcs code
  if ( erq.open (fifoname.data() ) == TPE::KO ) {
     EXCEPTIONS_RAISE(TPE::X_OPEN_ERQ,"TServerObcs.connect", "pb ob open erq");
  } else {
     erqName=fifoname;
  }

}//-------------------------

void TServerObcs::start()
{//---------- begin of Opcs code
  ACTIVE = TPE::false;
  connect(erqName);

}//-------------------------

void TServerObcs::stop()
{//---------- begin of Opcs code
      ACTIVE = TPE::false;
}//-------------------------

void TServerObcs::insert(rMsg M)
{//---------- begin of Opcs code
      rtnq.insert(M);
}//-------------------------

void TServerObcs::remove(rMsg M)
{//---------- begin of Opcs code
      erq.remove(M);
}//-------------------------

void TServerObcs::SetRtnQName(const TString &fifoname)
{//---------- begin of Opcs code
   rtnName=fifoname;
   rtnq.open(rtnName);  //enough for the momemt (may be we should used connect)
}//-------------------------

// ---------- internal operation bodies :

// END OF OBJECT : TServerObcs
```

```
        EXCEPTIONS_RAISE(TPE::X_OPEN_ERQ,"TclientObcs.connect", "pb on open erq");
    }
    for (i=0; i <MAX_PENDING_ER; i++) {
        rtnqTable[i]->Status = Tfree;
        rtnName = erqName +dec(i);
        if (rtnqTable[i]->Q.open (rtnName.data() ) == TPE::KO ) {
            EXCEPTIONS_RAISE(TPE::X_OPEN_ERQ,"TclientObcs.connect", "pb ob open rtnq");
        }// end if
    }// end for
}//-------------------------

TRtnQ TClientObcs::alloc()
{//---------- local Opcs variables
TPE::integer i;
//---------- begin of Opcs code
for (i=0; i <MAX_PENDING_ER; i++) {
    if (rtnqTable[i]->Status == Tfree) {
        rtnqTable[i]->Status=Tallocated;
        return rtnqTable[i]->Q;
    }
}
EXCEPTIONS_RAISE( TPE::X_OBCS_NOMOREQUEUES,"TClientObcs.alloc","nomore RtnQueues");
}//-------------------------


// ---------------------------------------------------------------------------
TPE::OK_KO TClientObcs::insrem(const rMsg M,TString& OpName)
{  insert(M);
  remove(M);
   if (M.GetX() == "KO"){
       RemoteStatus = TPE::KO;
       EXCEPTIONS_RAISE(TPE::X_KO,OpName,"retour SER KO");
  } else {
       RemoteStatus =TPE::OK;
  }
  return RemoteStatus;
}

TPE::Tboolean ShutDown()
{///---------- begin of Opcs code
    return OBCS_ACTIVE;
}//-------------------------

// END OF OBJECT : OBCSs
```

J.3.3.5 Class TServerObcs in C++

```
/// SPEC OF OBJECT : TServerObcs
#ifndef spec_TServerObcs
#define spec_TServerObcs
// ---------- provided types and constants :
#include "hrts/TErQ.h"
#include "hrts/TRtnQ.h"
// NESTED HOOD OBJECT : TClientObcs
// ---------- provided types and constants :
#include "hrts/HRTS_PE.h"
class TServerObcs
{
private:
    TString erqName;
    TString rtnName;
    TRtnQ   erq;
    TErQ    rtnq;
    TPE::Tboolean ACTIVE;
public:
    TServerObcs();
    TServerObcs(TPE::T_OBJECT index=TPE::STACK);
    ~TServerObcs();
    void SetRtnQName(const TString &fifoname);
    void connect(TString &fifoname);
    void insert(rMsg M);
    void remove(rMsg M);
    void start();
    void stop();
};
```

```
static TClientObcs ClientObcs ;

// ---------- provided operation bodies :
void INSERT(const rRtnQ ipc, rMsg msg)
{///---------- begin of Opcs code
    ClientObcs.insert(msg);
}//-------------------------

void REMOVE(const rRtnQ rtnq, rMsg msg)
{///---------- begin of Opcs code
    ClientObcs.remove(msg);
}//-------------------------

void FREE(const rRtnQ aQ)
{//---------- begin of Opcs code
    ClientObcs.free(aQ);
}//-------------------------

// NESTED HOOD OBJECT : TClientObcs
// ---------- internal types and constants :
// ---------- internal data :
static TPE::Tboolean OBCS_ACTIVE = TPE::false;

// ---------- provided operation bodies :
TClientObcs::TClientObcs(TPE::T_OBJECT index) :rtnName("")
{///---------- begin of Opcs code
      RemoteStatus = TPE::OK;
    for (TPE::integer i=0; i <MAX_PENDING_ER; i++) {
        rtnqTable[i] = new T_Qelem;
        rtnqTable[i]->Status = Tfree;
    }
    erqName="/tmp/HRTS/";
    erqName+=tabOBJECTS[index];
}//-------------------------

TClientObcs::~TClientObcs()
{///---------- begin of Opcs code
    for (TPE::integer i=0; i <MAX_PENDING_ER; i++) {
        delete rtnqTable[i];
    }
}//-------------------------


void TClientObcs::insert(rMsg msg)
{//---------- begin of Opcs code
    msg.SetRtnQName(rtnq.getName());
    erq.insert(msg);

}//-------------------------

void TClientObcs::remove(rMsg msg)
{///---------- begin of Opcs code
    rtnq.remove(msg);
    free (rtnq);
}//-------------------------


void TClientObcs::free(const rRtnQ aQ)
{//---------- local Opcs variables
TPE::integer i;
//---------- begin of Opcs code
for (i=0; i <MAX_PENDING_ER; i++) {
    if (rtnqTable[i]->Q == aQ) {
        rtnqTable[i]->Status=Tfree;
        return;
    }
}
EXCEPTIONS_RAISE( TPE::X_OBCS_INCONSISTENTPOOLSTATUS,"TclientObcs.free","Q not found");
}//-------------------------


void TClientObcs::connect(TString& fifoname)
{//---------- local Opcs variables
TPE::integer i;
//---------- begin of Opcs code
    erqName=fifoname;
    if ( erq.open (erqName.data() ) == TPE::KO ) {
```

```
TString TErQ::getName()
{//---------- local Opcs variables
    const char* name;
//---------- begin of Opcs code
    if (ACE_FIFO_Send::get_local_addr(name)==TPE::KO) {
        EXCEPTIONS_RAISE(TPE::X_INCONSISTENCY,"TErQ.getName", "malfunction in getname");
        return TString("");
    }else {
        return TString(name);
    }
}//-------------------------

// END OF OBJECT : TErQ
```

J.3.3.4 Class TClientObcs in C++

```
// SPEC OF OBJECT : TClientObcs
#ifndef INCLUDE_spec_TClientObcs
#define INCLUDE_spec_TClientObcs
// ---------- provided types and constants :
#include "hrts/TErQ.h"
#include "hrts/TRtnQ.h"
// NESTED HOOD OBJECT : TClientObcs
// ---------- provided types and constants :
typedef enum {Tfree, Tallocated} T_AllocStatus;
struct T_Qelem
{
    TRtnQ Q;
    T_AllocStatus Status;
};

class TClientObcs
{
private:
    TString erqName;
    TString  rtnName;
    TErQ erq;
    TRtnQ rtnq;
    T_Qelem * rtnqTable [ MAX_PENDING_ER ] ;
    TPE::OK_KO RemoteStatus;
public:
    TClientObcs(TPE::T_OBJECT index=TPE::STACK);
    ~TClientObcs();
    void insert(rMsg msg);
    void remove(rMsg msg);
    void free(const rRtnQ aQ);
    void connect(TString& fifoname);
    TRtnQ alloc();
    TPE::OK_KO status();
    TPE::OK_KO insrem(const rMsg M,TString& OpName);
};
typedef TClientObcs *pClientObcs;
typedef TClientObcs & rClientrObcs;
// ---------- provided not member operations :
TPE::Tboolean ShutDown();
// ---------- provided not member operations :
void INSERT(const rRtnQ ipc, rMsg msg);
void REMOVE(const rRtnQ rtnq, rMsg msg);
void FREE(const rRtnQ aQ);
// ---------- provided exceptions :
// NESTED HOOD OBJECT : TServerObcs
// ---------- provided types and constants :
// END OF OBJECT : TClientObcs
#endif

// BODY OF OBJECT : TClientObcs
// ========== visibility on specification file :
#include "hrts/TClientObcs.h"
#include "hrts/EXCEPTIONS.h"
#include <stream.h>
// NESTED HOOD OBJECT : Obcs
// ---------- internal types and constants :
// ---------- internal data :
```

```
#ifndef INCLUDE_spec_TErQ
#define INCLUDE_spec_TErQ
#include "hrts/HRTS_PE.h"
// NESTED HOOD OBJECT : TErQ
// ---------- provided types and constants :
#include <ace/FIFO_Send.h>
#include "hrts/TMsg.h"

class TErQ;
typedef TErQ& rErQ;
typedef TErQ* pErQ;
class TErQ: public ACE_FIFO_Send
{public:
    TErQ();
    ~TErQ();
    TPE::OK_KO insert(const rMsg IPCMsg);
    TPE::OK_KO open(const TString & fifoname);
    TString getName();
};
#endif //TErQ_H

// BODY OF OBJECT : TErQ
// ========== visibility on specification file :
#include "hrts/TErQ.h"
#include "hrts/EXCEPTIONS.h"
#include <rw/pstream.h>

// NESTED HOOD OBJECT : TErQ
// ---------- internal data :
// ---------- provided operation bodies :
TErQ::TErQ() :ACE_FIFO_Send()
{///---------- begin of Opcs code
    //we' ll see later if we need to do something here
}//-------------------------

TErQ::~TErQ()
{//---------- begin of Opcs code
    //we' ll see later if we need to do something here
}//-------------------------

TPE::OK_KO TErQ::insert(const rMsg IPCMsg)
{//---------- local Opcs variables
    TPE::integer fd=ACE_FIFO_Send::get_handle();
    const char* const buf = IPCMsg.GetBuffer();
//---------- begin of Opcs code
if (fd > 0) {
    ofstream ofstr(fd);
    RWpostream ipc(ofstr);
    ipc << IPCMsg.X.data();
    ipc << IPCMsg.Sender.data();
    ipc << IPCMsg.Sendee.data();
    ipc << IPCMsg.RtnQName.data();
    ipc << int (IPCMsg.Operation);//passe en int car pb avec enum
    ipc << int (IPCMsg.Constraint);//evite de redefinir "<<"
    if (buf) {
        ipc << buf;
    } else {
        ipc << "";
    }
    if (!ipc.good()) {
        EXCEPTIONS_LOG("TErQ::insert","probleme de transmission");
            return TPE::KO;
     } else {
        return TPE::OK;
    }
}
}//-------------------------

TPE::OK_KO TErQ::open(const TString & fifoname)
{//---------- begin of Opcs code
    if (ACE_FIFO_Send::open( fifoname.data()) == TPE::KO) {
        EXCEPTIONS_RAISE(TPE::X_OPEN,"TErQ.open","pb on open FIFO");
        return TPE::KO;
    }else {
        return TPE::OK;
      }
}//-------------------------
```

```
TPE::OK_KO TRtnQ::remove(const rMsg IPCMsg)
{//---------- local Opcs variables
    TPE::integer tmpInteger;
    char* chaine = new char[10000];
    TPE::integer fd=ACE_FIFO_Recv::get_handle();
//---------- begin of Opcs code
    if (fd > 0) {
        ifstream ifstr(fd);
      RWpistream ipc(ifstr);

        ipc.getString(chaine,10000);
        IPCMsg.X = TString(chaine);
        ipc.getString(chaine,10000);
            IPCMsg.Sender = TString(chaine);
        ipc.getString(chaine,10000);
            IPCMsg.Sendee = TString(chaine);
        ipc.getString(chaine,10000);
            IPCMsg.RtnQName = TString(chaine);
        ipc >> tmpInteger;
        IPCMsg.Operation = (TPE::T_OPERATION)tmpInteger;
        ipc >> tmpInteger;
        IPCMsg.Constraint = (TPE::T_CSTRNT)tmpInteger;
        ipc.getString(chaine, 10000);
        IPCMsg.SetBuffer(chaine);
        if (ipc.good()) {
             return TPE::OK;
        } else {
             EXCEPTIONS_LOG("TRtnQ::remove","reception pb");
            return TPE::KO;
        }
    } else {
            EXCEPTIONS_LOG("TRtnQ::remove","reception impossible");
            return TPE::KO;
     } //end if
}//-------------------------


TPE::OK_KO TRtnQ::open(const TString& fifoname)
{//---------- begin of Opcs code
    if (ACE_FIFO_Recv::open( fifoname.data()) == TPE::KO) {
        EXCEPTIONS_RAISE(TPE::X_OPEN,"TRtnq.open","pb on open FIFO");
        return TPE::KO;
    } else {
        return TPE::OK;
    }
}//-------------------------

const TString TRtnQ::getName()
{//---------- local Opcs variables
      const char* name;
//---------- begin of Opcs code
    if (ACE_FIFO_Recv::get_local_addr(name)==TPE::KO) {
        EXCEPTIONS_RAISE(TPE::X_INCONSISTENCY,"TRtnq.getName", "malfunction in getname");
    }
    return TString(name);
}//-------------------------

TPE::Tboolean operator ==(const rRtnQ r1, const rRtnQ r2)
{//---------- local Opcs variables
//   None
//---------- begin of Opcs code
    if (r1.get_handle() == r2.get_handle()) {
        return TPE::true;
    } else {
        return TPE::false;
    }
}//-------------------------

// END OF OBJECT : QUEUESs
```

J.3.3.3 Class TErQ -Requets Queue for IPC

```
// SPEC OF OBJECT : TErQ
```

```
rIOStream TIOStream::operator >>(TString& item)
{
  if (type != in) {
    EXCEPTIONS_LOG("TIOStream::operator >>(TString&)","pas ouvert en entre");
  } else {
    char chaine[10000];

    pistr->getString(chaine, 10000); //lecture de seulement 10000 carac!!!
    if (pistr->fail()) {
      EXCEPTIONS_LOG("TIOStream::operator >>(TString&)","Impossible de lire une chaine de caractere");
      pistr->clear();
    }
    item = TString(chaine);
  }
  return *this;
}

rIOStream TIOStream::operator >>(char& item)
{  if (type != in) {
    EXCEPTIONS_LOG("TIOStream::operator >>(char&)","pas ouvert en entre");
  } else {
    *pistr >> item;
    if (pistr->fail()) {
      EXCEPTIONS_LOG("TIOStream::operator >>(char&)","l'operation a echouee");
      pistr->clear();
    }
  }
  return *this;
}
// END OF OBJECT : TIOStream
```

J.3.3.2 Class TRtnQ - C++ return queue for IPC communication

```
/// SPEC OF OBJECT : TRtnQ
#ifndef INCLUDE_spec_TRtnQ
#define INCLUDE_spec_TRtnQ
#include "hrts/HRTS_PE.h"
#include "hrts/TMsg.h"

// NESTED HOOD OBJECT : TRtnQ
// ---------- provided types and constants :
#include <ace/FIFO_Recv.h>
// a trick since we are working on a HOOD4 prototype tool
class TRtnQ;
typedef TRtnQ *pRtnQ;
typedef TRtnQ & rRtnQ;
class TRtnQ: public ACE_FIFO_Recv
{public:
     TRtnQ();
     ~TRtnQ();
     TPE::OK_KO remove(const rMsg IPCMsg);
     TPE::OK_KO open(const TString & fifoname);
     const TString getName();
};
     TPE::Tboolean operator ==(const rRtnQ  r1, const rRtnQ  r2);
#endif
// BODY OF OBJECT : TRtnQ
// ========== visibility on specification file :
#include "hrts/TRtnQ.h"
#include "hrts/EXCEPTIONS.h"
#include <rw/pstream.h>

// NESTED HOOD OBJECT : TRtnQ
// ---------- internal data :
// ---------- provided operation bodies :
TRtnQ::TRtnQ() :ACE_FIFO_Recv()
{//---------- begin of Opcs code
     //we' ll see later if we need to do something here
}//-------------------------

TRtnQ::~TRtnQ()
{//---------- begin of Opcs code
     //we' ll see later if we need to do something here
}//-------------------------
```

```
    EXCEPTIONS_LOG("TIOStream::operator <<(const TPE::integer&)","l'operation a echouee");
    postr->clear();
   }
   return *this;
}

rIOStream TIOStream::operator <<(const TPE::real& item)
{  if (type != out) {
    if (type) {
     TIOStream::Libere();
    }
    TIOStream::OpenOut();
   }
   *postr << item;
   if (postr->fail()) {
    EXCEPTIONS_LOG("TIOStream::operator <<(const TPE::integer&)","l'operation a echouee");
    postr->clear();
   }
   return *this;
}

rIOStream TIOStream::operator <<(const TString& item)
{  if (type != out) {
    if (type) {
     TIOStream::Libere();
    }
    TIOStream::OpenOut();
   }
   *postr << item.data();
   if (postr->fail()) {
    EXCEPTIONS_LOG("TIOStream::operator <<(const TString&)","l'operation a echouee");
    postr->clear();
   }
   return *this;
}

rIOStream TIOStream::operator <<(const char& item)
{  if (type != out) {
    if (type) {
     TIOStream::Libere();
    }
    TIOStream::OpenOut();
   }
   *postr << item;
   if (postr->fail()) {
    EXCEPTIONS_LOG("TIOStream::operator <<(const char&)","l'operation a echouee");
    postr->clear();
   }
   return *this;
}

rIOStream TIOStream::operator >>(TPE::integer& item)
{  if (type != in) {
    EXCEPTIONS_LOG("TIOStream::operator >>(TPE::integer&)","pas ouvert en entre");
   } else {
    *pistr >> item;
    if (pistr->fail()) {
     EXCEPTIONS_LOG("TIOStream::operator >>(TPE::integer&)","l'operation a echouee");
     pistr->clear();
    }
   }
   return *this;
}

rIOStream TIOStream::operator >>(TPE::real& item)
{  if (type != in) {
    EXCEPTIONS_LOG("TIOStream::operator >>(TPE::real&)","pas ouvert en entre");
   } else {
    *pistr >> item;
    if (pistr->fail()) {
     EXCEPTIONS_LOG("TIOStream::operator >>(TPE::real&)","l'operation a echouee");
     pistr->clear();
    }
   }
   return *this;
}
```

```
  case none :    return;
  case in :
    delete istrstr;
    delete pistr;
    break;
  case out :
    delete ostrstr;
    delete postr;
    break;
  default :
    EXCEPTIONS_LOG("TIOStream::Libere","type indefini");
    break;
  }
  type = none;
}

void TIOStream::OpenIn(char* buf)
{  if (!buf) {
    EXCEPTIONS_LOG("TIOStream::OpenIn","buffer nul");
    return;
  }
  if (type) {
    EXCEPTIONS_LOG("TIOStream::OpenIn","deja ouvert");
  }
  istrstr = new istrstream(buf);
  pistr = new RWpistream(*istrstr);
  type = in;
}

void TIOStream::OpenOut()
{  if (type) {
    EXCEPTIONS_LOG("TIOStream::OpenOut","deja ouvert");
  }
  ostrstr = new ostrstream();
  postr = new RWpostream(*ostrstr);
  type = out;
}

const TString& TIOStream::GetBuffer()
{  if (type != out) {          //rien n'a ete mis dans le buffer
    if (type == in) {          //doit liberer le buffer en entree
      TIOStream::Libere();
    }
    return (*(new TString("")));
  }
  char * tmp = ostrstr->str();
  if (!tmp) {
    EXCEPTIONS_LOG("TIOStream::GetBuffer","buffer nul");
    TIOStream::Libere();
    return (*(new TString("")));
  } else {
    TIOStream::Libere();
    return (*(new TString(tmp)));
  }
}

void TIOStream::SetBuffer(const TString& buf)
{  if (!buf) {
    EXCEPTIONS_LOG("TIOStream::SetBuffer","le buffer est nul");
    return;
  } else {
    if (type != none) {
      TIOStream::Libere();
    }
    TIOStream::OpenIn(strdup(buf.data()));
  }
}

rIOStream TIOStream::operator <<(const TPE::integer& item)
{  if (type != out) {
    if (type) {
      TIOStream::Libere();
    }
    TIOStream::OpenOut();
  }
  *postr << item;
  if (postr->fail()) {
```

J.3.3 HRTS.OBCS C++ sample code

J.3.3.1 Class TIOStream in C++

```
// SPEC OF OBJECT : TIOStream
#ifndef INCLUDE_spec_TIOStream
#define INCLUDE_spec_TIOStream
// ========== visibility on required objects :
#include "hrts/HRTS_PE.h"
#include <strstream.h>
#include <rw/pstream.h>
// ---------- provided types and constants :
class TIOStream;
typedef TIOStream & rIOStream;
typedef TIOStream * pIOStream;
class TIOStream
{private:
  enum IOType
  {    none=0,    in=1,    out=2  };
    IOType type;
  istream* istrstr;
  ostrstream* ostrstr;
  RWpistream* pistr;
  RWpostream* postr;
   void Libere();
  void OpenIn(char* buf);
  void OpenOut();
public:
  TIOStream();
  ~TIOStream();
  const TString& GetBuffer();
  void SetBuffer(const TString& buf);
   virtual rIOStream operator <<(const TPE::integer&);
  virtual rIOStream operator <<(const TPE::real&);
  virtual rIOStream operator <<(const TString&);
  virtual rIOStream operator <<(const char&);
  virtual rIOStream operator >>(TPE::integer&);
  virtual rIOStream operator >>(TPE::real&);
  virtual rIOStream operator >>(TString&);
  virtual rIOStream operator >>(char&);
};
// END OF OBJECT : TIOStream
#endif

// BODY OF OBJECT : TIOStream
// ========== visibility on specification file :
#include "hrts/TIOStream.h"
// ========== visibility on required objects :
#include "hrts/EXCEPTIONS.h"
// ---------- internal types and constants :
// ---------- internal data :
// ---------- provided operation bodies :

TIOStream::TIOStream()
{  type = none;
  istrstr = (istrstream*) NULL;
  ostrstr = (ostrstream*) NULL;
  pistr = (RWpistream*) NULL;
  postr = (RWpostream*) NULL;
}

TIOStream::~TIOStream()
{  if (type == in) {
    delete istrstr;
    delete pistr;
  }
  if (type == out) {
    delete ostrstr;
    delete postr;
  }
  type = none;
}

void TIOStream::Libere()
{  switch (type) {
```

J.3.2.3 package TServerObcs in Ada

```
 --  Specifications :
 --          Client-server interprocess communication services
 --          described in HRM4 "L.3 Object OBCS"
 ------------------------------------------------------------------------------
with TMsg;
with TQPool;
package TServerObcs is
  type Instance is tagged private;
  procedure Insert (Me : in out Instance; aMessage : in out TMsg.TMsg);
  procedure Remove (Me : in out Instance; aMessage : out TMsg.TMsg);
private
  type Instance is tagged record
    QueuePool         : TQPool.Instance := TQPool.ServerQPool;
  end record;
end TServerObcs;
with HRTS_PE;
with EXCEPTIONS_Log;

package body TServerObcs is
  procedure Insert (Me : in out Instance; aMessage : in out TMsg.TMsg) is
  begin
       TQPool.SetCurrentRtnQ (Me.QueuePool, TMsg.GetRtnQId (aMessage));-- set cRTNQ as indicated in the message :
      TQPool.GetRtnQ (Me.QueuePool).all.Put (aMessage);  -- put message in current return queue :
  end Insert;
  procedure Remove (Me : in out Instance; aMessage : out TMsg.TMsg) is
  begin
      TQPool.GetErQ (Me.QueuePool).all.Get (aMessage);-- get ERQ and get message from it :
  end Remove;
end TServerObcs;
```

```
    end loop;
    end if;
    raise HRTS_PE.E_OBCS_NO_MORE_QUEUES;
  end Alloc;
  procedure Free (Me : in out Instance; thisQueue      : HRTS_PE.T_QueueIndex) is
  begin
    EXCEPTIONS_LOG ("Instance.Free : return queue number "   & HRTS_PE.T_QueueIndex'image (thisQueue));
   if Me.Number = -1 then
      EXCEPTIONS_LOG ("Instance.Free : *** inconsistent pool");
   else
     Me.rtnqTable(thisQueue).Status := Tfree;
   end if;
  end Free;

  procedure SetCurrentRtnQ (Me : in out Instance; thisQueue : HRTS_PE.T_QueueIndex) is
  begin
  EXCEPTIONS_LOG ("Instance.SetCurrentRtnQ : return queue number "    & HRTS_PE.T_QueueIndex'image (thisQueue));
   if (Me.Number = -1 or Me.rtnqTable(thisQueue).Q = null) then
       EXCEPTIONS_LOG ("Instance.SetCurrentRtnQ : *** inconsistent pool");
   elsif Me.rtnqTable(thisQueue).Status = Tallocated then
      EXCEPTIONS_LOG ("Instance.SetCurrentRtnQ : *** return queue error");
   else
     Me.rtnq := Me.rtnqTable(thisQueue).Q;
   end if;
  end SetCurrentRtnQ;
end TQPool;
```

J.3.2.2 Class TClientObcs in Ada

```
--          Client-server interprocess communication services
--          described in HRM4 "L.3 Object OBCS"
-------------------------------------------------------------------------
with TMsg;
with TQPool;

package TClientOBCS is
  type Instance is tagged private;
  procedure Insert (Me : in out Instance; aMessage : in out TMsg.TMsg);
  procedure Remove (Me : in out Instance; aMessage : out TMsg.TMsg);
    procedure Insrem (Me : in out Instance; aMessage : in out TMsg.TMsg);-- insert + remove :
private
  type Instance is tagged record
    QueuePool         : TQPool.Instance := TQPool.ClientQPool;
  end record;
end TClientOBCS;

with HRTS_PE;
with EXCEPTIONS_LOG
package body TClientObcs is
  procedure Insert (Me : in out Instance; aMessage : in out TMsg.TMsg) is
  RtnQId            : HRTS_PE.T_QueueIndex;
  begin
     TMsg.FlushParams (aMessage);-- enforce the writing of parameters in the stream file
    TQPool.Alloc (Me.QueuePool, RtnQId);-- allocate an available return queue
    TMsg.SetRtnQId (aMessage, RtnQId);-- write the return queue identifier into the message
     TQPool.GetErQ (Me.QueuePool).all.Put (aMessage); -- get request queue from queue pool and put message in it
       exception
      when HRTS_PE.E_OBCS_NO_MORE_QUEUES =>
        EXCEPTIOONS_LOG       (         "*** TClientObcs.Insert "         & "E_OBCS_NO_MORE_QUEUES"         );
  end Insert;
  procedure Remove (Me : in out Instance; aMessage : out TMsg.TMsg) is
  begin
      TQPool.GetRtnQ (Me.QueuePool).all.Get (aMessage); -- get current RTNQ l and get message from it
       TQPool.Free (Me.QueuePool, TMsg.GetRtnQId (aMessage));-- release current return queue
  end Remove;
   procedure Insrem (Me : in out Instance; aMessage : in out TMsg.TMsg) is
  begin -- insert a message into request queue, then remove a message from return queue
    Insert (Me, aMessage);
    Remove (Me, aMessage);
  end Insrem;
end TClientObcs;
```

```
    EXCEPTIONS_Log          (       "T_Queue.Get : received message is "       );
    TMsg.Trace (Message);
    EXCEPTIONS_Log     ("end");
    Full := False;
  end Get;
end T_Queue;-- end of protected type definition
type PT_Instance is access Instance;
type T_State is (IDLE, SERVED, ACTIVE);
type T_Session is record
  QPool          : PT_Instance := null;
  Indicator      : T_State := IDLE;
end record;
NB_MAX_SESSIONS   : constant HRTS_PE.T_Integer := 5;
S_Id              : HRTS_PE.T_Integer := 1;
type PT_Session is access T_Session;
type APT_Session is array (1 .. NB_MAX_SESSIONS) of PT_Session;
Sessions : APT_Session;

X_CONNEXION_ERROR : exception;

function ServerQPool return Instance is
begin
  Sessions(S_Id) := new T_Session;
  Sessions(S_Id).Indicator := SERVED;
  Sessions(S_Id).QPool := new Instance;
  Sessions(S_Id).QPool.Number := 1;
  Sessions(S_Id).QPool.all.erq := new T_Queue;
  Sessions(S_Id).QPool.all.rtnq := new T_Queue;
  for J in Sessions(S_Id).QPool.all.rtnqTable'range loop
    Sessions(S_Id).QPool.all.rtnqTable(J).Q := new T_Queue;
  end loop;
  EXCEPTIONS_LOG (">>>>>>>>> server connected");
  return Sessions(S_Id).QPool.all;
end ServerQPool;

function ClientQPool return Instance is
begin
  case Sessions(S_Id).Indicator is
    when IDLE       =>      EXCEPTIONS_LOG("Instance.ClientQPool : *** attempting to connect client before server process =>
    connexion failed");       raise X_CONNEXION_ERROR;
     when ACTIVE       =>        EXCEPTIONS_LOG ("Instance.ClientQPool : *** another client is connected yet to the server =>
    connexion failed");       raise X_CONNEXION_ERROR;
    when SERVED      =>       EXCEPTIONS_LOG (">>>>>>>>> client connected");
      S_Id := S_Id + 1;
      return Sessions(S_Id-1).QPool.all;
  end case;
end ClientQPool;
function GetErQ (Me : in Instance) return PT_Queue is
begin
  if Me.Number = -1 then
   EXCEPTIONS_LOG ("Instance.GetErQ : *** inconsistent pool");
  else
    return (Me.erq);
  end if;
end GetErq;
function GetRtnQ (Me : in Instance) return PT_Queue is
begin
  if (Me.Number = -1 or Me.rtnq = null) then
     EXCEPTIONS_LOG ("Instance.GetRtnQ : *** inconsistent pool");
    return (null);
  else
    return (Me.rtnq);
  end if;
end GetRtnQ;
  procedure Alloc (Me : in out Instance; aQueue      : out HRTS_PE.T_QueueIndex)is
begin-- allocate a return queue
  if Me.Number = -1 then
   EXCEPTIONS_LOG("Instance.Alloc : *** inconsistent pool");
  else
  for I in Me.rtnqTable'range loop
    if (Me.rtnqTable(I).Status = Tfree) then
      Me.rtnqTable(I).Status := Tallocated;
      aQueue := I;
      Me.rtnq := Me.rtnqTable(I).Q;
        EXCEPTIONS_LOG ("Instance.Alloc : return queue number "  & HRTS_PE.T_QueueIndex'image (aQueue));
      return;
    end if;
```

INTERNALS -- hidden part of the object:
--to be supplied by HRTS Working Group.
END OBJECT OBCS.

J.3.2 HRTS.OBCS Ada sample code

J.3.2.1 package TQPool

```
--       This package provides services for managing a queue pool.  The queue pool datas and services are implemented as a class
     whose name is Instance.     A queue pool consists in :
--            - 1 execution request queue : "erq"
--            - N available return queues : "rtnqTable"
--            - 1 current return queue, chosen among the return queue table
---            Execution and current return queues are accessed via an access type.
--            Queues in return queue table are accessed via a table index.
------------------------------------------------------------------------------
with HRTS_PE;
with TMsg;
package TQPool is
  use type HRTS_PE.T_QueueIndex;
  protected type T_Queue is
    entry Put (SentMsg     : in TMsg.TMsg);
    entry Get (ReceivedMsg : out TMsg.TMsg);
  private
    Message               : TMsg.TMsg;
    Full                  : Boolean := False;
  end T_Queue;
  type Instance is tagged private;
  type PT_Queue is access T_Queue;-- access type is used since type T_Queue is limited private :

      function ClientQPool return Instance;- queue pool constructor for client side, performing the connect :
     function ServerQPool return Instance;-- queue pool constructor for server side, performing the connect :
     function GetErQ (Me : in Instance) return PT_Queue;-- get execution request queue :
      function GetRtnQ (Me : in Instance) return PT_Queue; -- get active return queue :
       procedure Alloc (Me : in out Instance; aQueue      : out HRTS_PE.T_QueueIndex);-- alloc a return queue in the return queue
     table :
     procedure Free (Me : in out Instance; thisQueue      : in HRTS_PE.T_QueueIndex);-- free a return queue in the return queue
     table :
      procedure SetCurrentRtnQ (Me : in out Instance;thisQueue : in HRTS_PE.T_QueueIndex);-- set the current return queue :
private
  type TAllocStatus is (Tfree, Tallocated);
  type T_Qelem is
  record
    Q           : PT_Queue;
    Status      : TAllocStatus := Tfree;
  end record;
 type T_rtnqTable is    array ((HRTS_PE.T_QueueIndex'first+1) .. HRTS_PE.T_QueueIndex'last) of T_Qelem;
  type Instance is tagged record
    Number      : HRTS_PE.T_Integer := -1;
    erq         : PT_Queue;      -- execution request queue
    rtnq        : PT_Queue;      -- current return queue
    rtnqTable   : T_rtnqTable;   -- return queue table
  end record;
end TQPool;

with EXCEPTIONS_Log;
with HRTS_PE; use type HRTS_PE.T_Integer; use type HRTS_PE.T_QueueIndex;
package body TQPool is
  protected body T_Queue is
    entry Put (SentMsg     : in TMsg.TMsg) when not Full is
    begin
      Full := True;
      TMsg.Copy (SentMsg, Message);
      EXCEPTIONS_Log     (       "T_Queue.Put : sent message is "       );
      TMsg.Trace (Message);
      EXCEPTIONS_Log     ("end");
    end Put;
    entry Get (ReceivedMsg     : out TMsg.TMsg) when Full is
    begin
      TMsg.Copy (Message, ReceivedMsg);
```

J.3.1 Object GENERIC OBCS IS

FORMAL_PARAMETERS

TYPES

T_QUEUE

Fifo queue to be provided by the environment

OPERATIONs

IPCInsert (aMsg : **in** T_MSG; ina : **in** T_QUEUE)

Insert a message **in** a T_QUEUE queue

IPCRemove (aMSg: **in out** T_MSG; froma : **in out** T_QUEUE);

Removes a message from a T_QUEUE queue

DESCRIPTION

Implementation of a facility to setup IPC messages, s**end** them to external process using queuing software supplied by the environment, and processing incoming messages as well as **return** messages.

PROVIDED_INTERFACE

TYPES

TOBCS ;

-- **class** allow client code to instantiate either a ClientObcs or a ServerObcs

OPERATIONS

CONNECT(object : **in** HRTS_PE.T_OBJECT);

Creates dynamically both an ERQ queue and a pool of HRTS.MAX_Pending_ER RTN queues **with** an acces key of **type** T_Object. If the queues are already exist, nothing **is** done (it **is** supposed that some**body** else did the connect before.)Sets the OBCS Active boolean value to true **in** current space (either Client or server). If something goes wrong during queue creation, the exception XOBCS_CommuncationError shall be raised, and the OBCS Active boolean value **is** set to false **in** current space

INSERT(aMSG : **in out** T_MSG);

Allocates a RTNQ for this MSG and insert operation execution request MSG data **in** a sending IPC QUEUE named ERQ. When this operation **is** called by an OPCS_ER client code, for the first time it initializes a pool of RTNQ and creates the ERQ.

REMOVE(QUEUE: **in** T_QUEUE; aRTNMSG : **out** T_MSG);

Removes a RTNMSG from the **return** queus allocated at INSERT time..

FREE(MSG : **in** T_MSG);

deallocates the RTNQ and the message structure MSG when all parameters have been processed.

GtRprt(QUEUE: **in** T_QUEUE; aRTNMSG : **out** T_MSG);

Removes a Report RTNMSG from the **return** queues allocated at INSERT time

INSERT(QUEUE: **in** T_QUEUE; aRTNMSG : **in**T_MSG);

Inserts a RTNMSG **in** an RTNQ identified by a incoming message.

REMOVE(aMSG : **in out** T_MSG)

Remove a MSG data from an IPC QUEUE.

Triggered by ServerOBCS, either through serverOBCS infinite loop or by a call back dispatch by a IPC event handler.

ShutDown return HRTS_PE.Boolean

Returns a booelan value set to true if the OBCS activities has been stooped by somebody.

Start

Sets the OBCSActive boolean value to true.

Stop

Sets the OBCS Active boolean value to false.

EXCEPTIONS

XOBCSCommuncationErro **raised_by** CONNECT **when** trouble **in** creating the ERQ or RTNQ queues.

REQUIRED_INTERFACE

OBJECT FORMAL_PARAMETERS

Types

T_QUEUE,T

OBJECT HRTS_PE

types

Booelan, T_OPERATION, T_OBJECT;

J.3 Object OBCS

OBCSs is a standardised implementation of client-server interprocess communication through local or distributed fifo queues. **Class** TObcs provides facilities to declare, initiate a client-server sending message queue and associate receiving fifo queues dedicated to handle HOOD operation requests and **return** parameters.

This design shall be a generic **with** formal parameters the type or **class** TQUEUE allowing to instantiate items that are :

- either requiring inter process communication (IPC) queues for protocol constrained operation implementation on a same memory partition,
- or requiring inter VN node (IVN) queues for remote call implementation between different Vns. Note that in case two Vns are allocated on the same memory partition, optimization may possibly shunt the VN queues overhead.

As implementation principle have already been described in section 16 , Figure 96 - *Client and Server spaces interactions through OBCS queues* - illustrates the queuing schemes between client and server threads

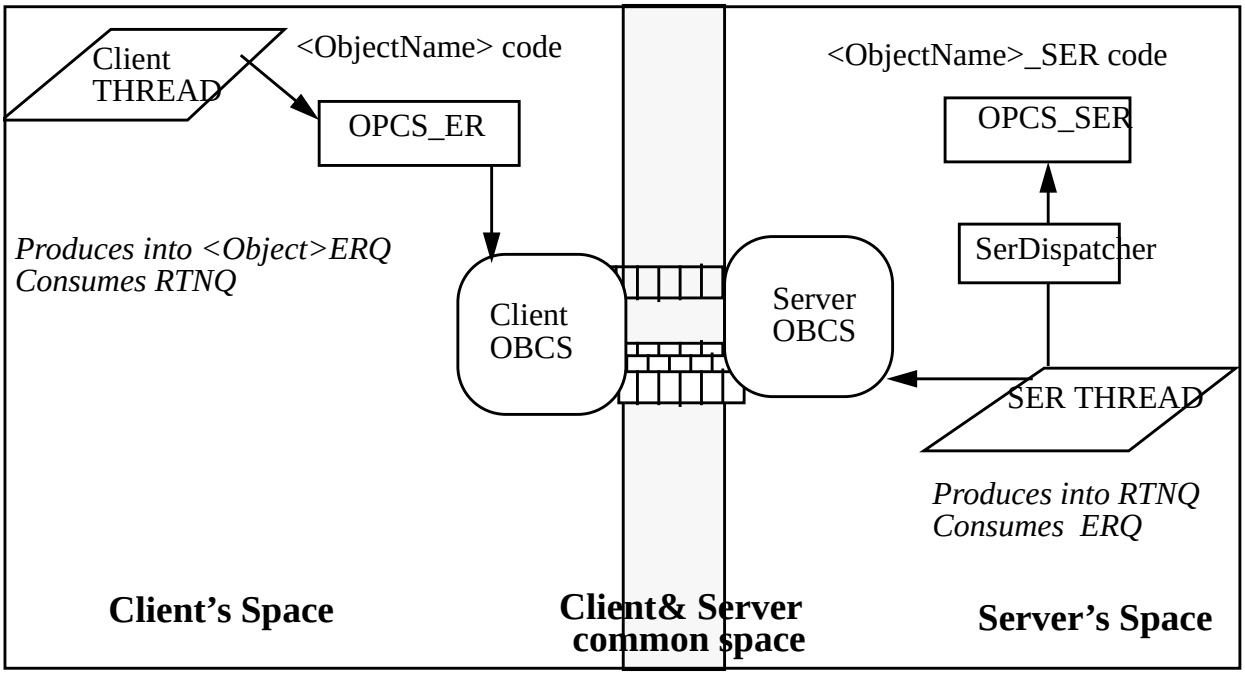


Figure 96 -Client and Server spaces interactions through OBCS queues

```
// ===========================================================================
{  return ( TFsm< T_STATES    ,    T_OPERATION ,              T_FSM_ID    >::pFsm )(*iter)();
}

// ===========================================================================
template < class T_STATES    , class T_OPERATION ,          class T_FSM_ID    >
void TFsmList< T_STATES    ,             T_OPERATION ,            T_FSM_ID    >::Append(
              const TFsm< T_STATES    ,          T_OPERATION ,     T_FSM_ID    >::pFsm pT )
{  lst->append( (RWCollectable*)pT );
}

// ===========================================================================
template < class T_STATES    ,         class T_OPERATION ,          class T_FSM_ID    >
TFsm< T_STATES    ,     T_OPERATION ,     T_FSM_ID    >::pFsm
  TFsmList< T_STATES    ,          T_OPERATION ,          T_FSM_ID    >::Reset()
{  iter->reset();
  return ( TFsm< T_STATES    ,  T_OPERATION ,      T_FSM_ID    >::pFsm )(*iter)();
}

// ===========================================================================
template < class T_STATES    ,         class T_OPERATION ,          class T_FSM_ID    >
TFsmList< T_STATES    ,         T_OPERATION ,         T_FSM_ID    >::TFsmList()
{  lst  = new RWOrdered              ;
  iter = new RWOrderedIterator( *lst );
}

// ===========================================================================
template < class T_STATES    ,         class T_OPERATION ,          class T_FSM_ID    >
TFsmList< T_STATES    ,         T_OPERATION ,         T_FSM_ID    >::~TFsmList()
{  delete iter;
  delete lst ;
  iter = 0;
  lst  = 0;
}

// ===========================================================================
// Definition des fonctions et procedures locales
// ===========================================================================
static void perform_test( const TPE::integer &action )
// ===========================================================================
{
//END OF OBJECT:FSMs
```

```
 TTransition< T_STATES   ,     T_OPERATION ,            T_FSM_ID   >::pTransition T;  //pointeur sur une transition
 TString message;
 message = "etat initial: "       +          tabSTATES[ AUTOM_STATE ] ;
 E = ETATS.FindCurrent( AUTOM_STATE );      //recherche de l'etat courant
 if ( E )  {                  //etat trouve
  T = E->TRANST.Find( Trigger );      //recherche trigger
  if ( T )    {              //trigger trouve
   message += "\ntransition: "  +tabOPERATION[ Trigger ]  + "\netat final: "     +     tabSTATES[ T->NEXT_STATE ] ;
   if ( T->TRACES == TPE::TON )      {
    cerr << message            << endl;
   }
   E->CURRENT_STATE = T->NEXT_STATE;
   AUTOM_STATE     = T->NEXT_STATE;
  }
 }
}

// ====================================================================
template < class T_STATES   ,        class T_OPERATION ,         class T_FSM_ID   >
void TFsm< T_STATES   ,        T_OPERATION ,         T_FSM_ID   >::FIRES(      T_STATES   &State  ,
                      const T_OPERATION &Triggers)
/{ TFsmState  < T_STATES   ,   T_OPERATION , T_FSM_ID   >::pFsmState  E;  //pointeur sur un etat
 TTransition< T_STATES   ,     T_OPERATION ,      T_FSM_ID   >::pTransition T;  //pointeur sur une transition
 TString message;
 message = "etat initial: "       +          tabSTATES[ AUTOM_STATE ] ;
 E = ETATS.FindCurrent( State );      //recherche d'un etat
 if ( E )  {                  //etat trouve
  T = E->TRANST.Find( Triggers );      //recherche trigger
  if ( T )    {              //trigger trouve
   message += "\netat initial force a: " +  tabSTATES[ State ]  +"\ntransition: " +  tabOPERATION[ Triggers ]  +
           "\netat final: "      +    tabSTATES[ T->NEXT_STATE ] ;
   if ( T->TRACES == TPE::TON )      {
    cerr << message            << endl;
   }
   E->CURRENT_STATE = T->NEXT_STATE;
   AUTOM_STATE     = T->NEXT_STATE;
   State          = AUTOM_STATE  ;
  }
 }
}

// ====================================================================
template < class T_STATES   ,        class T_OPERATION ,         class T_FSM_ID   >
TFsm< T_STATES   ,     T_OPERATION ,     T_FSM_ID   >::~TFsm()
/{}

// ====================================================================
template < class T_STATES   ,        class T_OPERATION ,         class T_FSM_ID   >
TFsm< T_STATES   ,     T_OPERATION ,     T_FSM_ID   >::TFsm()
{}

// ====================================================================
// Definition des methodes de la classe TFsmList
// ====================================================================
// ====================================================================
template < class T_STATES   ,        class T_OPERATION ,         class T_FSM_ID   >
// --------------------------------------------------------------------
TFsm< T_STATES   ,     T_OPERATION ,     T_FSM_ID   >::pFsm
 TFsmList< T_STATES   , T_OPERATION ,          T_FSM_ID   >::Find( const T_FSM_ID &Object )
{ TFsm< T_STATES   ,      T_OPERATION ,       T_FSM_ID   >::pFsm F;  //pointeur sur une machine a etats
 F = Reset();
 while ( F )  {          //recherche de la bonne machine
  if ( compare( F->ID  ,   Object ) == TPE::true )    {          //trouve
   return F;
  }    else    {          //pas trouve
   F = Next();  //on voit la machine suivante
  }
 }
 EXCEPTIONS_LOG( "TFsmList::Find"   ,     "Machine[objet] non retrouve" );
 EXCEPTIONS_set( X_BER                    );
 return TPE::OK;;
}

// ====================================================================
template < class T_STATES   ,        class T_OPERATION ,         class T_FSM_ID   >
TFsm< T_STATES   ,     T_OPERATION ,     T_FSM_ID   >::pFsm
 TFsmList< T_STATES   ,          T_OPERATION ,          T_FSM_ID   >::Next()
```

```
}

// ===================================================================
template < class T_STATES   ,          class T_OPERATION ,          class T_FSM_ID   >
TFsmState< T_STATES   , T_OPERATION ,   T_FSM_ID   >::pFsmState
 TFsmStateList< T_STATES   , T_OPERATION ,   T_FSM_ID   >::Next()
{  return ( TFsmState< T_STATES   ,   T_OPERATION ,   T_FSM_ID   >::pFsmState )(*iter)();
}

// ===================================================================
template < class T_STATES   ,   class T_OPERATION ,   class T_FSM_ID   >
TFsmState< T_STATES   ,   T_OPERATION ,   T_FSM_ID   >::pFsmState
 TFsmStateList< T_STATES   ,  T_OPERATION ,    T_FSM_ID   >::Reset()
{  iter->reset();
 return ( TFsmState< T_STATES   ,   T_OPERATION ,  T_FSM_ID   >::pFsmState )(*iter)();
}

// ===================================================================
template < class T_STATES   ,  class T_OPERATION ,   class T_FSM_ID   >
TFsmStateList< T_STATES   , T_OPERATION ,   T_FSM_ID   >::TFsmStateList()
{  lst  = new RWOrdered              ;
 iter = new RWOrderedIterator( *lst );
}

// ===================================================================
template < class T_STATES   ,  class T_OPERATION ,  class T_FSM_ID   >
TFsmStateList< T_STATES   ,T_OPERATION ,   T_FSM_ID   >::~TFsmStateList()
{  delete iter;
 delete lst ;
 iter = 0;
 lst  = 0;
}

// ===================================================================
// Definition des methodes de la classe TFsm
// ===================================================================
// ===================================================================
template < class T_STATES   ,          class T_OPERATION ,          class T_FSM_ID   >
void TFsm< T_STATES   ,         T_OPERATION ,          T_FSM_ID   >::CREATE(
        const TPE::integer &Nb_States    , const TPE::integer &Nb_Triggers   , const T_STATES    &Initial_State )
// ===================================================================
{  TRACES            = TPE::TOFF   ;
 NB_Etats          = 0          ;
 NB_Triggers       = 0          ;
 INITIAL_NB_Etats   = Nb_States   ;
 INITIAL_NB_Triggers = Nb_Triggers  ;
 INITIAL_STATE      = Initial_State;
 AUTOM_STATE        = Initial_State;
}

// ===================================================================
template < class T_STATES   ,          class T_OPERATION ,          class T_FSM_ID   >
void TFsm< T_STATES   ,         T_OPERATION ,          T_FSM_ID   >::TRANS( const T_STATES    &Current_State ,
                        const T_OPERATION &Triggers      ,       const T_STATES    &Next_State    )
{  TFsmState  < T_STATES   ,   T_OPERATION ,       T_FSM_ID   >::pFsmState   E; //pointeur sur un etat
 TTransition< T_STATES   ,             T_OPERATION ,             T_FSM_ID   >::pTransition T; //pointeur sur une transition
 E = ETATS.Find( Current_State );    //recherche du bon ETAT si existe
 if ( !E )   {               //etat n'existe pas
   E       = new TFsmState< T_STATES   ,   T_OPERATION ,      T_FSM_ID   >; //cree l'etat
   NB_Etats = NB_Etats + 1              ;
   E->NB_TRANS = 0          ;
   E->ID       = Current_State;
   ETATS.Append( E );
  }
 T     = new TTransition< T_STATES   ,      T_OPERATION ,       T_FSM_ID   >;
 T->TRIGGER    = Triggers                ;
 T->NEXT_STATE = Next_State              ;
 NB_Triggers   = NB_Triggers + 1          ;
 E->TRANST.Append( T );          //rajoute a la liste
 E->CURRENT_STATE = Current_State;
 E->TRACES       = TPE::TOFF   ;
}

// ===================================================================
template < class T_STATES   , class T_OPERATION ,   class T_FSM_ID   >
void TFsm< T_STATES   ,         T_OPERATION ,          T_FSM_ID   >::FIRE( const T_OPERATION &Trigger)
{  TFsmState  < T_STATES   ,   T_OPERATION ,     T_FSM_ID   >::pFsmState   E;   //pointeur sur un etat
```

```
 TTransitionList< T_STATES   ,  T_OPERATION ,              T_FSM_ID   >::Reset()
{  iter->reset();
 return ( TTransition< T_States ,  T_OPERATION ,    T_FSM_ID   >::pTransition )(*iter)();
}

// ========================================================================
template < class T_STATES   , class T_OPERATION ,    class T_FSM_ID   >
TTransitionList< T_STATES  ,  T_OPERATION ,  T_FSM_ID   >::TTransitionList()
{  lst  = new RWOrdered            ;
 iter = new RWOrderedIterator( *lst );
}

// ========================================================================
template < class T_STATES   ,  class T_OPERATION ,        class T_FSM_ID   >
TTransitionList< T_STATES   ,  T_OPERATION ,    T_FSM_ID   >::~TTransitionList()
{  delete iter;
 delete lst ;
 iter = 0;
 lst  = 0;
}

// ========================================================================
// Definition des methodes de la classe TFsmState
// ========================================================================
// ========================================================================
template < class T_STATES   ,        class T_OPERATION ,        class T_FSM_ID   >
TFsmState< T_STATES   ,       T_OPERATION ,       T_FSM_ID   >::TFsmState()
{}

// ========================================================================
template < class T_STATES   ,        class T_OPERATION ,        class T_FSM_ID   >
TFsmState< T_STATES   ,       T_OPERATION ,       T_FSM_ID   >::~TFsmState()
{}

// ========================================================================
// Definition des methodes de la classe TFsmStateList
// ========================================================================
// ========================================================================
template < class T_STATES   ,        class T_OPERATION ,        class T_FSM_ID   >
TFsmState< T_STATES   ,       T_OPERATION ,       T_FSM_ID   >::pFsmState
 TFsmStateList< T_STATES   ,    T_OPERATION , T_FSM_ID   >::FindCurrent( const T_STATES CurrentState )
// ========================================================================
{  TFsmState::pFsmState E;      //pointeur sur un etat
 E = Reset();       //1er de la liste
 while ( E )  {          //recherche de l 'etat courant
  if ( CurrentState == E->CURRENT_STATE )   {//trouve
   return E;
  }   else   {
   E = Next();//voir etat suivant
  }
 }
 EXCEPTIONS_LOG( "TFsmStateList::FindCurrent" ,          "etat courant non retouve"  );
 EXCEPTIONS_set( X_BER                );
 return TPE::OK;;
}

// ========================================================================
template < class T_STATES   , class T_OPERATION ,class T_FSM_ID   >
TFsmState< T_STATES   , T_OPERATION , T_FSM_ID   >::pFsmState
 TFsmStateList< T_STATES   ,   T_OPERATION ,  T_FSM_ID   >::Find( const T_STATES StateID )
{  TFsmState< T_STATES   ,   T_OPERATION ,  T_FSM_ID   >::pFsmState E; //pointeur sur un etat
 E = Reset();   //1er de la liste
 while ( E )  {       //recherche de l'etat
  if ( StateID == E->ID )   {          //trouve
   return E;
  }   else   {
   E = Next(); //voir etat suivant
  }
 }
 return TPE::OK;;
}

// ========================================================================
template < class T_STATES   ,        class T_OPERATION ,        class T_FSM_ID   >
void TFsmStateList< T_STATES   ,  T_OPERATION ,   T_FSM_ID   >::Append(
    const TFsmState< T_STATES   ,   T_OPERATION ,       T_FSM_ID   > *pT )
{  lst->append( (RWCollectable *)pT );
```

```
{  TFsm::pFsm    p               ;//pointeur sur une Fsm
  TPE::Tboolean trouve = TPE::false;
  p = FSM.Reset();
  while ( ( !trouve ) && p )   {                    //recherche de la bonne machine
    if ( p->ID == Object )     {                   //trouve
      trouve = TPE::true;
    }     else     {        //pas trouve
      p = FSM.Next();        //on voit la machine suivante
    }
  }
  if ( !trouve )   {
    p          = new TFsm       ;
    p->ID      = Object         ;
    p->IFSM    = InitFunc       ;
    NB_MACHINES = NB_MACHINES + 1;
    FSM.Append(p);
  }
  p->IFSM();    //fonction d'initialisation
}

// ============================================================================
// Definition des methodes de la classe TTransition
// ============================================================================
// ============================================================================
template < class T_STATES    ,          class T_OPERATION ,          class T_FSM_ID    >
void TTransition< T_STATES    ,               T_OPERATION ,               T_FSM_ID    >::dump()
{}

// ============================================================================
template < class T_STATES    ,          class T_OPERATION ,          class T_FSM_ID    >
TTransition< T_STATES    ,          T_OPERATION ,          T_FSM_ID    >::TTransition()
{  //    TRACES     = TPE::TOFF             ;
}

// ============================================================================
template < class T_STATES    ,          class T_OPERATION ,          class T_FSM_ID    >
TTransition< T_STATES    ,          T_OPERATION ,          T_FSM_ID    >::~TTransition()
{}

// ============================================================================
// Definition des methodes de la classe TTransitionList
// ============================================================================
// ============================================================================
template < class T_STATES    ,          class T_OPERATION ,          class T_FSM_ID    >
TTransition< T_STATES              T_OPERATION ,          T_FSM_ID    >::pTransition
  TTransitionList< T_STATES    ,               T_OPERATION ,               T_FSM_ID    >::Find( const T_OPERATION &Triggers )
{  TTransition::pTransition T;//pointeur sur une transition
  T = Reset();
  while ( T )   {
    if ( T->TRIGGER == Triggers )     {                         //trouve
      return T;
    }     else     {        //pas trouve
      T = Next();//transition suivante
    }
  }
  EXCEPTIONS_LOG( "TTransitionList::Find" ,               "trigger non autorise"  );
  EXCEPTIONS_set( TPE::X_UNW             );
  return TPE::OK;;
}

// ============================================================================
template < class T_STATES    ,          class T_OPERATION ,          class T_FSM_ID    >
void TTransitionList< T_STATES    ,      T_OPERATION ,         T_FSM_ID    >::Append(      const TTransition< T_STATES    ,
     T_OPERATION ,          T_FSM_ID    > *pT )
{  lst->append( (RWCollectable * )pT );
}

// ============================================================================
template < class T_STATES    ,          class T_OPERATION ,          class T_FSM_ID    >
TTransition< T_STATES    ,          T_OPERATION ,          T_FSM_ID    >::pTransition
  TTransitionList< T_STATES    ,               T_OPERATION ,               T_FSM_ID    >::Next()
{  return ( TTransition< T_States ,    T_OPERATION ,    T_FSM_ID    >::pTransition )(*iter)();
}

// ============================================================================
template < class T_STATES    ,   class T_OPERATION ,          class T_FSM_ID    >
TTransition< T_STATES    ,          T_OPERATION ,          T_FSM_ID    >::pTransition
```

```
template < class T_STATES    ,        class T_OPERATION ,        class T_FSM_ID   >
void TRACE_ALL(      TFsmList< T_STATES    ,                        T_OPERATION ,
                        T_FSM_ID   > &FSM    ,            const T_FSM_ID             &Object ,
           const TPE::T_TRACE          &MODE   )
// ==========================================================================
{  TFsm::pFsm             F;//pointeur sur une machine a etats
 TFsmState::pFsmState     E;//pointeur sur un etat
 TTransition::pTransition T;//pointeur sur une transition
 F = FSM.Find( Object );//recherche de la bonne machine
 if ( F )   {             //trouve
   E = F->ETATS.Reset();//1er de la liste
   while ( E )    {
     T = E->TRANST.Reset();//1er de la liste
     while ( T ) {
       T->TRACES = MODE;
       T = E->TRANST.Next();       //on voit la transition suivante
     }
     E = F->ETATS.Next();//on voit l'etat suivant
   }
 }
}

// ==========================================================================
template < class T_STATES    ,        class T_OPERATION ,        class T_FSM_ID   >
/void TRACE(      TFsmList< T_STATES    ,
                        T_OPERATION ,
                        T_FSM_ID   > &FSM    ,
         const T_FSM_ID             &Object ,
         const T_STATES             &STATE  ,
         const TPE::T_TRACE          &MODE   )
// ==========================================================================
{  TFsm::pFsm             F;//pointeur sur une machine a etats
 TFsmState::pFsmState     E;//pointeur sur un etat
 TTransition::pTransition T;//pointeur sur une transition
 F = FSM.Find( Object );//recherche de la bonne machine
 if (F)   {               //machine trouvee
   E = F->ETATS.FindCurrent( STATE );//recherche de l'ETAT courant
   if ( E )    {         //etat trouve
     T = E->TRANST.Reset();//1er de la liste
     while ( T ) {
       T->TRACES = MODE;
       T = E->TRANST.Next();       //on voit la transition suivante
     }
   }
 }
}

// ==========================================================================
template < class T_STATES    ,        class T_OPERATION ,        class T_FSM_ID   >
void SET(      TFsmList< T_STATES    ,                    T_OPERATION ,
                     T_FSM_ID   > &FSM    ,        const T_FSM_ID             &Object ,
       const T_STATES             &State  )
{  TFsm::pFsm F;       //pointeur sur une machine a etats
 F = FSM.Find( Object );
 if ( F )  {              //objet trouve
   F->AUTOM_STATE = State;//force l'etat
 }
}

// ==========================================================================
template < class T_STATES    ,        class T_OPERATION ,        class T_FSM_ID   >
void REINIT(      TFsmList< T_STATES    ,                       T_OPERATION ,
                       T_FSM_ID   > &FSM    ,          const T_FSM_ID             &Object )
// ==========================================================================
{  TFsm::pFsm F;//pointeur sur une machine a etats
 F = FSM.Find( Object );
 if ( F )   {      //objet trouve
   F->IFSM();//fonction d'initialisation
 }
}

// ==========================================================================
template < class T_STATES    ,        class T_OPERATION ,        class T_FSM_ID   >
void INIT(      TFsmList< T_STATES    ,                    T_OPERATION ,
                     T_FSM_ID   > &FSM    ,        const T_FSM_ID             &Object   ,
        const pFunc                &InitFunc )
// ==========================================================================
```

```
// ===========================================================================
template < class T_STATES    ,          class T_OPERATION ,          class T_FSM_ID    >
// ---------------------------------------------------------------------------
void INIT(       TFsmList< T_STATES    ,                    T_OPERATION ,                    T_FSM_ID    > &FSM     ,
         const T_FSM_ID              &Object   ,          const pFunc               &InitFunc );

void test();
// ===========================================================================
// END OF OBJECT : FSMs
#endif

// BODY OF OBJECT:FSMs
#include "hrts/FSMs.h"
#include "hrts/EXCEPTIONS.h"
static void perform_test   ( const TPE::integer &action );
static void write_menu     (                     );
static void Mngr_ostm_test (                     );
static void STACK_ostm_test(                     );

// ===========================================================================
template < class T_STATES    ,          class T_OPERATION ,          class T_FSM_ID    >
void CREATE(       TFsmList< T_STATES    ,                    T_OPERATION ,
                   T_FSM_ID    > &FSM          ,          const T_FSM_ID              &Object       ,
          const TPE::integer          &Nb_States     ,          const TPE::integer          &Nb_Triggers   ,
          const T_STATES              &Initial_State )
{
 TFsm::pFsm F;//pointeur sur une machine a etats
 F = FSM.Find( Object );//recherche de la bonne machine
 if ( F )   {       //trouve
   F->CREATE( Nb_States     ,
              Nb_Triggers   ,
              Initial_State );
 }
}

// ===========================================================================
template < class T_STATES    ,          class T_OPERATION ,          class T_FSM_ID    >
void TRANS(        TFsmList< T_STATES    ,                    T_OPERATION ,
                   T_FSM_ID    > &FSM          ,          const T_FSM_ID              &Object       ,
          const T_STATES              &Current_State ,          const T_OPERATION           &Triggers     ,
          const T_STATES              &Next_State    )
{  TFsm::pFsm F;//pointeur sur une machine a etats
 F = FSM.Find( Object );//recherche de la bonne machine
 if ( F )   {       //machine trouvee
   F->TRANS( Current_State ,
             Triggers      ,
             Next_State    );
 }
}

// ===========================================================================
template < class T_STATES    ,          class T_OPERATION ,          class T_FSM_ID    >
void FIRE(       TFsmList< T_STATES    ,                    T_OPERATION ,                    T_FSM_ID    > &FSM     ,
         const T_FSM_ID              &Object   ,          const T_OPERATION           &Trigger )
{  TFsm::pFsm F;//pointeur sur une machine a etats
 F = FSM.Find( Object );       //recherche de la bonne machine
 if ( F )   {       //machine trouve
   F->FIRE( Trigger );
 }
}

// ===========================================================================
template < class T_STATES,          class T_OPERATION,          class T_FSM_ID>
void FIRES(       TFsmList< T_STATES    ,                    T_OPERATION ,
                  T_FSM_ID    > &FSM      ,          const T_FSM_ID              &Object   ,
             T_STATES              &State    ,          const T_OPERATION           &Triggers )
{  TFsm::pFsm F;//pointeur sur une machine a etats
 F = FSM.Find( Object );//recherche de la bonne machine
 if ( F )   {       //machine trouve
   F->FIRES( State    ,
             Triggers );
 }
}

// ===========================================================================
```

```
   TPE::integer              NB_Triggers       ; // nbre total de stimulis
   TPE::integer              INITIAL_NB_Etats  ;
   TPE::integer              INITIAL_NB_Triggers; // pour controles
   T_FSM_ID                  ID                ; // ID automate
   pFunc                   IFSM              ; // function d'init
   T_STATES                  INITIAL_STATE     ; // etat pour reset
   T_STATES                  AUTOM_STATE       ; // etat courant
   TFsmStateList< T_STATES   , T_OPERATION ,   T_FSM_ID   > ETATS   ; // les etats de l'AUTOM
   typedef TFsm< T_STATES   , T_OPERATION ,  T_FSM_ID   > *pFsm;
   TFsm ();
   ~TFsm();
   void CREATE( const TPE::integer &Nb_States    ,          const TPE::integer &Nb_Triggers   ,          const T_STATES
    &Initial_State );
  void TRANS ( const T_STATES    &Current_State ,          const T_OPERATION  &Triggers    ,          const T_STATES
    &Next_State    );
   void FIRE  ( const T_OPERATION  &Trigger      );
   void FIRES (      T_STATES     &State        ,          const T_OPERATION  &Triggers     );
};

// ===================================================================
template < class T_STATES   ,         class T_OPERATION ,         class T_FSM_ID   >
class TFsmList
{private:
 RWOrdered        *lst ;
 RWOrderedIterator *iter;
public:
 TFsmList();
 ~TFsmList();
 TFsm< T_STATES   ,       T_OPERATION ,       T_FSM_ID   >::pFsm Find( const T_FSM_ID &Object );
 TFsm< T_STATES ,       T_OPERATION ,       T_FSM_ID   >::pFsm Next();
 TFsm< T_STATES   ,       T_OPERATION ,       T_FSM_ID   >::pFsm Reset();
 void Append( const TFsm< T_STATES   ,                    T_OPERATION ,                    T_FSM_ID   >::pFsm pT );
};

// ===================================================================
template < class T_STATES   ,         class T_OPERATION ,         class T_FSM_ID   >
void CREATE(      TFsmList< T_STATES   ,                    T_OPERATION ,                    T_FSM_ID   > &FSM         ,
         const T_FSM_ID             &Object       ,          const TPE::integer          &Nb_States    ,          const TPE::integer
    &Nb_Triggers   ,          const T_STATES             &Initial_State );

// ===================================================================
template < class T_STATES   ,         class T_OPERATION ,         class T_FSM_ID   >
void TRANS(       TFsmList< T_STATES   ,                    T_OPERATION ,                    T_FSM_ID   > &FSM         ,
         const T_FSM_ID             &Object       ,          const T_STATES             &Current_State ,
         const T_OPERATION          &Triggers     ,          const T_STATES             &Next_State    );

// ===================================================================
template < class T_STATES   ,         class T_OPERATION ,         class T_FSM_ID   >
void FIRE(      TFsmList< T_STATES   ,                    T_OPERATION ,                    T_FSM_ID   > &FSM     ,
         const T_FSM_ID             &Object  ,          const T_OPERATION          &Trigger );

// ===================================================================
template < class T_STATES,          class T_OPERATION,          class T_FSM_ID>
 void FIRES(      TFsmList< T_STATES   ,                    T_OPERATION ,                    T_FSM_ID   > &FSM     ,
        const T_FSM_ID             &Object  ,          T_STATES           &State   ,          const T_OPERATION          &Triggers );

// ===================================================================
template < class T_STATES   ,         class T_OPERATION ,         class T_FSM_ID   >
void TRACE_ALL(      TFsmList< T_STATES   ,                    T_OPERATION ,                    T_FSM_ID   > &FSM
    ,          const T_FSM_ID             &Object ,          const TPE::T_TRACE          &MODE   );

// ===================================================================
template < class T_STATES   ,         class T_OPERATION ,         class T_FSM_ID   >
void TRACE(      TFsmList< T_STATES   ,         T_OPERATION ,         T_FSM_ID   > &FSM    ,          const T_FSM_ID
    &Object ,          const T_STATES             &STATE  ,          const TPE::T_TRACE          &MODE   );

// ===================================================================
template < class T_STATES   ,         class T_OPERATION ,         class T_FSM_ID   >
void SET(      TFsmList< T_STATES   ,                    T_OPERATION ,                    T_FSM_ID   > &FSM     ,
        const T_FSM_ID             &Object ,          const T_STATES             &State  );
/
// ===================================================================
template < class T_STATES   ,         class T_OPERATION ,         class T_FSM_ID   >
void REINIT(      TFsmList< T_STATES   ,                    T_OPERATION ,                    T_FSM_ID   > &FSM     ,
         const T_FSM_ID             &Object );
```

J.2.2.2 G_FSM C++ Code Illustration

```
// SPEC OF OBJECT : FSMs
#ifndef INCLUDE_spec_FSMs
#define INCLUDE_spec_FSMs
#include <rw/ordcltn.h>
#include "hrts/EXCEPTIONS.h"
#include "hrts/HRTS_PE.h"
#include "PROJECT_PE.h"
typedef void  (*pFunc)(void);

// ========================================================================
template < class T_STATES    ,          class T_OPERATION ,          class T_FSM_ID    >
class TTransition : public RWCollectable
{  public:
   T_OPERATION  TRIGGER   ;// stimuli
   T_STATES     NEXT_STATE;      // etat suivant
   TPE::T_TRACE TRACES    ;      // flag traces
   typedef TTransition< T_STATES    ,      T_OPERATION ,       T_FSM_ID    >* pTransition;
   TTransition ();
   ~TTransition();
   void dump();
};

// ========================================================================
template < class T_STATES    ,          class T_OPERATION ,          class T_FSM_ID    >
class TTransitionList
{  private:
   RWOrdered        *lst ;
   RWOrderedIterator *iter;
  public:
   TTransitionList ();
   ~TTransitionList();
   TTransition< T_STATES    ,      T_OPERATION ,    T_FSM_ID    > *Find( const T_OPERATION &Triggers);
   TTransition< T_STATES    ,       T_OPERATION ,    T_FSM_ID    > *Next();
   TTransition< T_STATES    ,       T_OPERATION ,    T_FSM_ID    > *Reset();
   void Append( const TTransition< T_STATES    ,    T_OPERATION ,         T_FSM_ID    > *pT);
};

// ========================================================================
template < class T_STATES    ,     class T_OPERATION ,          class T_FSM_ID    >
class TFsmState : public RWCollectable
{  public:
   T_STATES                  ID         ;
   TPE::integer              NB_TRANS     ;
   TPE::T_TRACE                 TRACES      ;
   T_STATES                  CURRENT_STATE;
   TTransitionList< T_STATES    ,      T_OPERATION ,    T_FSM_ID    > TRANST       ; // la liste des transitions
   typedef TFsmState< T_STATES    ,     T_OPERATION ,    T_FSM_ID    > *pFsmState;
   TFsmState ();
   ~TFsmState();
};

// ========================================================================
template < class T_STATES    ,     class T_OPERATION ,   class T_FSM_ID    >
class TFsmStateList
{  private:
   RWOrdered        *lst ;
   RWOrderedIterator *iter;
  public:
   TFsmStateList ();
   ~TFsmStateList();
   TFsmState< T_STATES    ,   T_OPERATION ,  T_FSM_ID    > *Find( const T_STATES StateID );
   TFsmState< T_STATES    ,    T_OPERATION ,      T_FSM_ID    > *FindCurrent( const T_STATES CurrentState );
   TFsmState< T_STATES    ,   T_OPERATION ,   T_FSM_ID    > *Next();
   TFsmState< T_STATES    ,     T_OPERATION ,    T_FSM_ID    > *Reset();
   void Append( const TFsmState< T_STATES    ,   T_OPERATION ,    T_FSM_ID    > *pT );
};

// ========================================================================
template < class T_STATES    ,  class T_OPERATION ,  class T_FSM_ID    >
class TFsm : public RWCollectable
{  public:
   TPE::T_TRACE              TRACES           ; // flag de traces
   TPE::integer              NB_Etats         ; // nbre total d'etats
```

```
with HRTS_PE; use type HRTS_PE.T_Integer;
with HRTS.EXCEPTIONS;
package body G_TFsm is
  procedure Create    (    Me    : in out Instance;    NbStates     : in HRTS_PE.T_Integer;
    NbTriggers    : in HRTS_PE.T_Integer;    InitialState : in T_State    ) is
   begin
    Me.CurrentState := InitialState;
    Me.InitialState := InitialState;
    Me.NbDefTransitions := 0;
   end Create;
  procedure Trans    (    Me    : in out Instance;    CurrentState : in T_State;    Trigger       : in T_Operation;
                                NextState    : in T_State    ) is
  NbTransitions : HRTS_PE.T_Integer      renames Me.NbDefTransitions;
  NewTrans : T_Transitio      renames Me.Transitions (NbTransitions);
  begin
    NewTrans :=      (      CurrentState,      Trigger,      NextState      );
    NbTransitions := NbTransitions + 1;
  end Trans;

  procedure Fire    (    Me    : in out Instance;    Trigger       : in T_Operation    ) is
  begin
    Fires (Me, Me.CurrentState, Trigger);
  end Fire;

  procedure Fires    (    Me    : in out Instance;    State         : in T_State;
    Trigger       : in T_Operation    ) is
    T             : T_Transition;
  TheTransitions : T_TransitionArray      renames Me.Transitions;
  NbTransitions  : HRTS_PE.T_Integer  renames Me.NbDeftransitions;
  begin
     for I in TheTransitions'first .. NbTransitions loop
     T := TheTransitions (I);
     if (T.PreviousState = State) and (T.Operation = Trigger) then
           Me.CurrentState := T.NextState;
       return;
     end if;
    end loop;
    raise HRTS_PE.X_BADExecutionRequest;
  end Fires;
  procedure TraceAll    (    Me : in out Instance;    Mode          : in HRTS_PE.T_Trace    ) is
  begin
    null;  -- not yet implemented
  end TraceAll;

  procedure Trace    (    Me    : in out Instance;    State         : in T_State;
    Mode          : in HRTS_PE.T_Trace    ) is
  begin
    null;-- not yet implemented
  end Trace;

  procedure Set    (    Me    : in out Instance;    State         : in T_State    ) is
  begin
    Me.CurrentState := State;
  end Set;

  function GetState (Me : in Instance) return T_State is
  begin
    return (Me.CurrentState);
  end GetState;

  procedure Reinit    (    Me           : in out Instance    ) is
  begin
    Me.CurrentState := Me.InitialState;
  end Reinit;
end G_TFsm;
```

X_FSMError **raised_by** CREATE,SET,FIRE **when** mismatch **in** FSMname, or Statename.
REQUIRED_INTERFACE
OBJECT HRTS_PE
types
T_STATES **is** (ETAT1, ETAT2, • • • ETATn);
--enumerated **type** describing all possible states
T_OPERATION **is** (OPERATION1, OPERATION2, • • • •OPERATIONn);
--enumerated **type** describing all possible operations
INTERNALS -- hidden part of the object:
--to be supplied by HRTS Working Group.
END OBJECT FSMs.

J.2.2 HRTS.FSMS sample code

J.2.2.1 G_FSM Ada Code Illustration

```
-------------------------------------------------------------------------
-- Hood Run Time Suppport / generic package for FSM management
-------------------------------------------------------------------------
with HRTS_PE;
generic
     type T_Operation is (<>);
     type T_State is (<>);
     NB_MAX_TRANSITIONS : HRTS_PE.T_Integer;
package G_TFsm is
  type Instance  is tagged private;  -- we use the «instance» naming schema for classes
  X_BADREQUEST    : Exception;
  procedure Create    (    Me          : in out Instance;    NbStates     : in HRTS_PE.T_Integer;       -- unused
    NbTriggers   : in HRTS_PE.T_Integer;       -- unused    InitialState : in T_State    );
  procedure Trans    ( Me   : in out Instance;    CurrentState : in T_State;    Trigger       : in T_Operation;
                             NextState    : in T_State    );
  procedure Fire    (    Me  : in out Instance;    Trigger       : in T_Operation    );
  procedure Fires    (    Me  : in out Instance;    State         : in T_State;    Trigger      : in T_Operation    );
 procedure TraceAll  ( Me  : in out Instance;    Mode          : in HRTS_PE.T_Trace    );
  procedure Trace    (    Me  : in out Instance;    State         : in T_State;    Mode         : in HRTS_PE.T_Trace    );
 procedure Set       (    Me  : in out Instance;    State        : in T_State    );
  function GetState (Me : in Instance) return T_State; -- to get the current state :
  procedure Reinit    (    Me : in out Instance    );

private
  type T_Transition is record
    PreviousState : T_State;
    Operation      : T_Operation;
    NextState     : T_State;
  end record;
  type T_TransitionArray is array (HRTS_PE.T_Integer range 0 .. NB_MAX_TRANSITIONS) of T_Transition;

  typeInstance is tagged record
    Transitions    : T_TransitionArray;
    InitialState  : T_State;
    CurrentState  : T_State;
    NbDefTransitions   : HRTS_PE.T_Integer;
  end record;
end G_TFsm;
```

J.2 Object FSMS

FSMS is a standardised implementation of finite state machine supporting code. **Class** TFsm provides facilities to declare, initialize a finite state machine, and fire operation to execute a transition according to an event. Second it also provides for tracing of automata execution. This HRTS provides both an OOP interface by means of provided TFsm **class** and a modular interface by means or object FSMS provided operations.

J.2.1 Object FSMs IS

DESCRIPTION

Implementation of a finite state machine manager. An FSM can be created using create, a transition id **define**d by the TRANS operation. An automata step **is** executed when triggering FIRE operation. If the current state does not allow firing of a transition associated for the operation event ,the exception X_Bad_Request **is** raised and state **is** unchanged.

If the TRACE toggle was activated, a trace displays the transition data.

IMPLEMENTATION_CONSTRAINTS

The data structure representing the FSM should allow incremental and dynamical definition by means of the trans operation. In order to ga**in** efficiency and trade off **with** reliability and automated code generation support, the object, states and operation shall be implemented as enumerated **type**s. These definitions shall be located **in** the HRTS_PE object **in** order to limit the changes hen a design **is** updated. Whether there **is** one T_STATES, T_OPERATION for a whole project, or weather there are as many such **type**s[4] as objects **is** left to the implementation experience[5]. In the last **case** some memory space **is** spared at the expenses of more **type** definition, and a generic definition of FSMs **with** formal parameters T_STATES and T_OPERATION.

PROVIDED_INTERFACE

TYPES

TFSM **is** implemented by TFSM.TFSM.

-- **class** defining an FSM implementation

OPERATIONS

CREATE(OBJECT : **in** T_OBJET, Nb_States : **in** T_STATES, Nb_Triggers : **in** T_OPERATION, init_State : **in** T_States, error : **in** Tp_OPERATION);

Initializes the data structure for an FSM of name OBJECT, **with** Nb_States and Nb_Triggers and initial state init_state.

TRANS(OBJECT : **in** T_OBJET, CURR_State : **in** T_STATES, Trigger : **in** T_OPERATION, Next_State : **in** T_STATES, Action : **in** Tp_OPERATION);

Defines a transition for a trigger from CURR_State to NextState.

FIRE(OBJECT : **in** T_OBJET, Trigger : **in** T_OPERATION);

Fires a transition **in** currentstate for the Trigger. If successful the FSM **is in** nextstate, else it stays **in** currentstate and raises the exception X_Badrequest.

FIRE(OBJECT : **in** T_OBJET, FORCED_STATE : **inout** T_STATEs; Trigger : **in** T_OPERATION);

Fire a transition from FORCE_STATE for the Trigger. If successful the FSM **is in** nextstate, else it stays **in** currentstate and raises the exception X_Badrequest..

TRACE(OBJECT : **in** T_OBJET, STATE : **in** T_STates; Trace : **in** Tp_OPERATION);

trace all transition for state STATE.

TRACE(OBJECT : **in** T_OBJET, Trace : **in** Tp_OPERATION);

trace all transition

SET(OBJECT : **in** T_OBJET, InaState : **in** T_States)

Sets the FSM currentstate to InaState..

RESET(OBJECT : **in** T_OBJET)

Sets the FSM currentstate to initialstate.

EXCEPTIONS

X_BadRequest **raised_by** FIRE **when** currentstate cannot fire Operation request.

[4]. which could be name T<ObjectNAme>States and T_<ObjectNameOperation

[5]. we know it works **in** Ada95, but we have not yet tested such an implementation **in** C++.

```
    return Project_Exceptions.current();
}//-- end of Opcs bodycode ----------------------
// END OF OBJECT : EXCEPTIONS
```

```
//---------- begin of Opcs code
switch ( action) {
    case 0 :
        break;
    case 1 :
        pEXCEPTIONS=new TExceptions;
        break;
    case 2 :
        delete ( pEXCEPTIONS);
        break;
    case 3 :
        exception = TPE::X_UNW;
        set (exception);
        break;
    case 4 :
        reset ( ) ;
        break;
    case 5 :
        exception = current ( );
        cout << "exception courante =" << exception << endl;
        break;
    case 6 :
        if   ( raised( ) == TPE::true) {
             cout << "exception levee" << endl;
        } else {
             cout << "exception  non levee" << endl;
        }
        break;
    case 7 :
         LOG("TExceptions.test", "ESSAI");
         break;
    case 8 :
        X_LOG.dump( );
        break;
    default :
        cerr <<"action inconnue" << endl ;
    }
}//------------------------------

// NESTED HOOD OBJECT : D_EXCEPTION
// ---------- internal data :
static TExceptions  Project_Exceptions;

// ---------- provided operation bodies :
void Raise(const TPE::T_X_Value an_exception)
{///-- begin of Opcs code -------------------------
    Project_Exceptions.set(an_exception);
}//-- end of Opcs bodycode ----------------------

void LOG(const TString from, const TString reason)
{////-- begin of Opcs code -------------------------
 Project_Exceptions.LOGG(from,reason);
}//-- end of Opcs bodycode ----------------------

void set(const TPE::T_X_Value an_exception)
{////-- begin of Opcs code -------------------------
    Project_Exceptions.set(an_exception);
}//-- end of Opcs bodycode ----------------------

void resetlog()
{///-- begin of Opcs code -------------------------
    Project_Exceptions.resetlog();
}//-- end of Opcs bodycode ----------------------

void reset()
{//-- begin of Opcs code -------------------------
    Project_Exceptions.reset();
}//-- end of Opcs bodycode ----------------------

TPE::Tboolean raised()
{///-- begin of Opcs code -------------------------
    return Project_Exceptions.raised();
}//-- end of Opcs bodycode ----------------------

TPE::T_X_Value current()
{///-- begin of Opcs code -------------------------
```

```
void TExceptions::resetlog()
{///---------- begin of Opcs code
    X_LOG.reset();
}//-------------------------

void TExceptions::reset()
{//---------- begin of Opcs code
    X_VALUE=TPE::X_OK;
}//-------------------------

TPE::Tboolean TExceptions::raised()
{//---------- local Opcs variables
//---------- begin of Opcs code
     if ( X_VALUE == TPE::X_NONE) {
        return TPE::false;
     } else {
        return TPE::true;
     }
}//---------------------------

TPE::T_X_Value TExceptions::current()
{///---------- begin of Opcs code
    return X_VALUE;
}//---------------------------

void TExceptions::LOGG(const TString from, const TString reason)
{///---------- begin of Opcs code
    X_LOG.Write_Log(from,reason);
}//---------------------------

TExceptions::~TExceptions()
{}

TExceptions::TExceptions()
{//---------- begin of Opcs code
    X_VALUE = TPE::X_NONE;
}//---------------------------

void TExceptions::test()
{//---------- local Opcs variables
    TPE::integer action =-1;
//---------- begin of Opcs code
    Tlog Test_Log;
    Test_Log.test ();
    Test_Log.~Tlog();
    cout << "TEST DE LA CLASSE TExceptions" <<endl;
    while ( action != 0 ) {
        write_menu ( );      // affichage menu specifique sur cout
        action = read_action( ); // lecture code action sur cin
        perform_test ( action );
    }
    cout << " Fin TEST DE LA CLASSE TExceptions"<< endl;
}//-------------------------

// ---------- internal operation bodies :
void TExceptions::write_menu()
{////---------- begin of Opcs code
    cout << "Enter"          << endl;
    cout << "1 - trigger constructor"<< endl;
    cout << "2 - trigger destructor"<< endl;
    cout << "3 - Exception.Set  to X_UNW"<< endl;
    cout << "4 - Exception.Reset"<< endl;
    cout << "5 - Print Exception.Current"<< endl;
    cout << "6 - test Exception.raised"<< endl;
    cout << "7 - Exception.LOG (TExceptions.test, ESSAI)"<< endl;
    cout << "8 - Print EXCEPTIONS.LOG file"<< endl;
    cout << "0 - exit"  << endl;
    cout << endl << ">> ";
} //-------------------------


void TExceptions::perform_test(const TPE::integer action)
{//---------- local Opcs variables
    TPE::T_X_Value exception = TPE::X_UNW;
```

```
void Tlog::test()
{//---------- local Opcs variables
    TPE::integer action =-1;
//----------- begin of Opcs code
    cout << "TEST DE LA CLASSE Tlog" <<endl;
    while ( action != 0 ) {
        write_menu ( );      // affichage menu specifique sur cout
        action = read_action( ); // lecture code action sur cin
        perform_test ( action );
    }
    cout << " Fin TEST DE LA CLASSE Tlog"<< endl;
}//-------------------------

// ---------- internal operation bodies :
void Tlog::write_menu()
{//----------- begin of Opcs code
    cout << "Enter"          << endl;
    cout << "1 - trigger constructor"<< endl;
    cout << "2 - trigger destructor"<< endl;
    cout << "3 - Set_log Display ON"<< endl;
    cout << "4 - Set_log Display OFF"<< endl;
    cout << "5 - Set_log Printer ON"<< endl;
    cout << "6 - Set_log Printer OFF"<< endl;
    cout << "7 - Write_log (TlogORIGINE, Tlog DESTINATION)"<< endl;
    cout << "8 - Print EXCEPTIONS.LOG file"<< endl;
    cout << "0 - exit"  << endl;
    cout << endl << ">> ";
}//-------------------------

void Tlog::perform_test(const TPE::integer action)
{//---------- local Opcs variables
    char caractere; // dummy car
//----------- begin of Opcs code
switch ( action) {
    case 0 :
        break;
    case 1 :
        P_UnTlog=new Tlog;
        break;
    case 2 :
        delete(P_UnTlog);
        break;
    case 3 :
        Set_Log(TPE::ON,Print_Log);
        break;
    case 4 :
        Set_Log(TPE::OFF,Print_Log);
        break;
    case 5 :
        Set_Log(Display_Log,TPE::ON);
        break;
    case 6 :
        Set_Log(Display_Log,TPE::OFF);
        break;
    case 7 :
        Write_Log("Tlog ORIGINE", "Tlog DESTINATION");
         break;
    case 8 :
        cout<<"--------debut dump LOG courant---"<<endl;
        dump();
        cout<<"--------fin dump---"<<endl;
         break;
    default :
        cerr <<"action inconnue" << endl ;
    }
}//-------------------------

// NESTED HOOD OBJECT : TExceptions
// ---------- internal data :
static TExceptions* pEXCEPTIONS;
// ---------- provided operation bodies :
void TExceptions::set(const TPE::T_X_Value exception)
{///----------- begin of Opcs code
    X_VALUE = exception;
}//----------------------------
```

```
#define EXCEPTIONS_CURRENT ::current
// END OF OBJECT : EXCEPTIONS
#endif
// BODY OF OBJECT : EXCEPTIONS
// ========== visibility on specification file :
#include "hrts/EXCEPTIONS.h"
// NESTED HOOD OBJECT : Tlog
// ---------- internal data :
static Tlog* P_UnTlog;
// ---------- provided operation bodies :
Tlog::Tlog()
{///---------- begin of Opcs code
    Display_Log = TPE::ON;
    Print_Log = TPE::ON;
    Log_File.open("/tmp/EXCEPTIONS.LOG", ios::app );
    if ( Log_File.fail() ) {
        cerr<< " ECHEC A L' OUVERTURE DE EXCEPTIONS.LOG" <<endl;
    }
}//------------------------

Tlog::~Tlog()
{////---------- begin of Opcs code ---------------------------------
    Log_File.close( );
}//------------------------

void Tlog::Set_Log(const TPE::T_ON_OFF Display, const TPE::T_ON_OFF Printer)
{////---------- begin of Opcs code
    Display_Log = Display;
    Print_Log = Printer;
}//-------------------------

void Tlog::reset()
{///---------- begin of Opcs code
    Log_File.close( );
    Log_File.open("/tmp/EXCEPTIONS.LOG", ios::out);
    if (Log_File.fail() ) {
        cerr << "DE Tlog::reset:  ECHEC A L'OUVERTURE DE EXCEPTIONS.LOG" <<endl;
    }
    Log_File.close( );
}//-------------------------
void Tlog::Write_Log(const TString from, const TString reason)
{//---------- local Opcs variables
    TString Message;
//---------- begin of Opcs code
    Message+="Exception from : ";
    Message+=from;
    Message+="   Reason : ";
    Message+=reason;
     if ( Display_Log ==TPE::ON ) {
        cerr << Message << endl;
    }
    if ( Print_Log == TPE::ON) {
        Log_File << Message <<endl;
        Log_File.flush ();
    }
}//-------------------------

void Tlog::dump()
{//---------- local Opcs variables
    char caractere; // dummy car
//---------- begin of Opcs code
    Log_File.close( );
    Log_File.open("/tmp/EXCEPTIONS.LOG", ios::in);
    if (Log_File.fail() ) {
        cerr << " ECHEC A L'OUVERTURE DE EXCEPTIONS.LOG" <<endl;
    }
    while ( Log_File.get (caractere) ) {
        cout.put ( caractere ); // echo on cout
    }
    Log_File.close( );
    Log_File.open("/tmp/EXCEPTIONS.LOG", ios::app );
    if ( Log_File.fail() ) {
        cerr<< " ECHEC RE OUVERTURE DE EXCEPTIONS.LOG" <<endl;
     }
}//-------------------------
```

J.1.2 HRTS.EXCEPTIONS sample code

```
// SPEC OF OBJECT : EXCEPTIONS
#ifndef INCLUDE_spec_EXCEPTIONS
#define INCLUDE_spec_EXCEPTIONS
// ========== visibility on required objects :
#include "hrts/HRTS_PE.h"
// NESTED HOOD OBJECT : D_EXCEPTION
// ---------- provided not member operations :
void Raise(const TPE::T_X_Value an_exception);
void LOG(const TString from, const TString reason);
void set(const TPE::T_X_Value an_exception);
void reset();
void resetlog();
TPE::Tboolean raised();
TPE::T_X_Value current();
// NESTED HOOD OBJECT : Tlog
// ---------- provided types and constants :
class Tlog
{private:
    TPE::T_ON_OFF Print_Log ;
    TPE::T_ON_OFF Display_Log ;
    fstream  Log_File ;
public:
    Tlog();
    ~Tlog();
    void Set_Log(const TPE::T_ON_OFF Display, const TPE::T_ON_OFF Printer);
    void Write_Log(const TString from, const TString reason);
    void reset();
    void dump();
    void test();
private:
    void write_menu();
    void perform_test(const TPE::integer action);
};

// NESTED HOOD OBJECT : TExceptions
// ---------- provided types and constants :
class TExceptions
{protected:
    Tlog   X_LOG; // exception_logging
    TPE::T_X_Value X_VALUE; // code interne d'exeception
public:
    void set(const TPE::T_X_Value exception);
    void reset();
    void resetlog();
    TPE::Tboolean raised();
    TPE::T_X_Value current();
    void LOGG(const TString from, const TString reason);
    ~TExceptions();
    TExceptions();
    void test();
private:
    void write_menu();
    void perform_test(const TPE::integer action);
};

#define EXCEPTIONS_LOG ::LOG
#define EXCEPTIONS_set ::set                                                    OS
#define EXCEPTIONS_reset ::reset
#define EXCEPTIONS_resetlog ::resetlog
#define EXCEPTIONS_RAISE(enumer,string1,string2) \
      ::LOG((string1),(string2));\
      ::set((enumer))
#define EXCEPTIONS_HANDLE() \
    if (::raised()) { \
        X_CATCH(); \
        return; \
    }
```

J.1.1 Object EXCEPTIONS IS

DESCRIPTION

Code to provide a software interface for encapsulating and mapping the exceptions handling **with** similar behaviour for different target languages. A global exception data **is** updated whenever an exception **is** raised by a server operation. This data can then be tested by the client code on **return** from the server operation.
Also a number of tracing and logging functions are supported allowing to have standardized management of error and exceptions logging and tracing.

PROVIDED_INTERFACE

TYPES

TExceptions ;
-- **class** that **define**s an exception management system
Tlog;
-- **class** that **define**s a logging facility on a file system
T_TOGGLE **is** (NONE,TTY, WINDOW, REMOTE_WINDOW);
-- enumerated **type** defining various displays for the exceptions logging file
T_TOGGLEMODE **is** (ON,OFF);
-- enumerated **type** defining setting of the toggles

OPERATIONS

SET(an_exception : **in** HRTS_PE.T_XValue);
Updates the internal global variable «CurrentException» **with** a T_XValue.
RAISE(an_exception : **in** T_XValue);
Updates the global variable «CurrentException» **with** a T_XValue,and **return**s to caller
RESET;
Updates the global variable «CurrentException» **with** the T_XValue X_OK.
CurrentException return HRTS_PE.TString;
Returns a string associated to the current Xvalue data
LOG(fromstring : **in** TString, whystring : **in** TString);
Adds fromstring and whystring to a logging file called by **default** «EXCEPTIONS.LOG»
TRACE(toggle : **in** T_toggle, mode: **in** T_ToggleMode);
Allows to set various toggle for displaying the loggingfile:
ToggleModes are ON and OFF
Toggles are : TTY, WINDOW,REMOTEWINDOW
DUMP;
displays the contents of the LoggingFile

REQUIRED_INTERFACE

OBJECT HRTS_PE
types
T_XValue **is** (X_OK, X_KO, X_Bad_ExceutionRequest• • •);
-- enumerated **type** defining exception values for a project
T_TString **is** implmented by a String **class**;
-- Type hiding an implmentation provided string asbtract data **type** implementation

INTERNALS -- hidden part of the object:
--to be supplied by HRTS Working Group.

END EXCEPTIONS.

Complete specifications of these **class**es and objects are given below; implementation on specific targets may use directly target operating system primitives , or intermediate Real Time library services possibly standardised such as those specified in [EXTRA] or [OMG-CORBA] , **public** domain C++ library such as ACE [SCHMIDT94] encapsulating SUN/UNIX RPC software.

J.1 Object Exceptions

Since exception management is not supported and standardised by all target languages, an abstract exception type implementation has been defined as standardised interface between HOOD designs and specific target machines and operating systems.

Exceptions provides primary facilities to declare, raise and test exceptions, and secondary facilities to log, trace and monitor exceptions events through files and possibly Xwindows. Figure 95 - *Client_Server view of EXCEPTIONS object* - gives an overview of its architecture in terms of objects and **class**es. The object D_Exceptions is a «global exception data» for a whole project, updated each time an exception is raised, and which may be tested by a client code on server operation return. This design allows the definition of a modular interface for the object EXCEPTIONS. For better flexibility, the **class**es Texceptions or Tlog may be instanciated several times in a project in order to define sas many exception management systems.

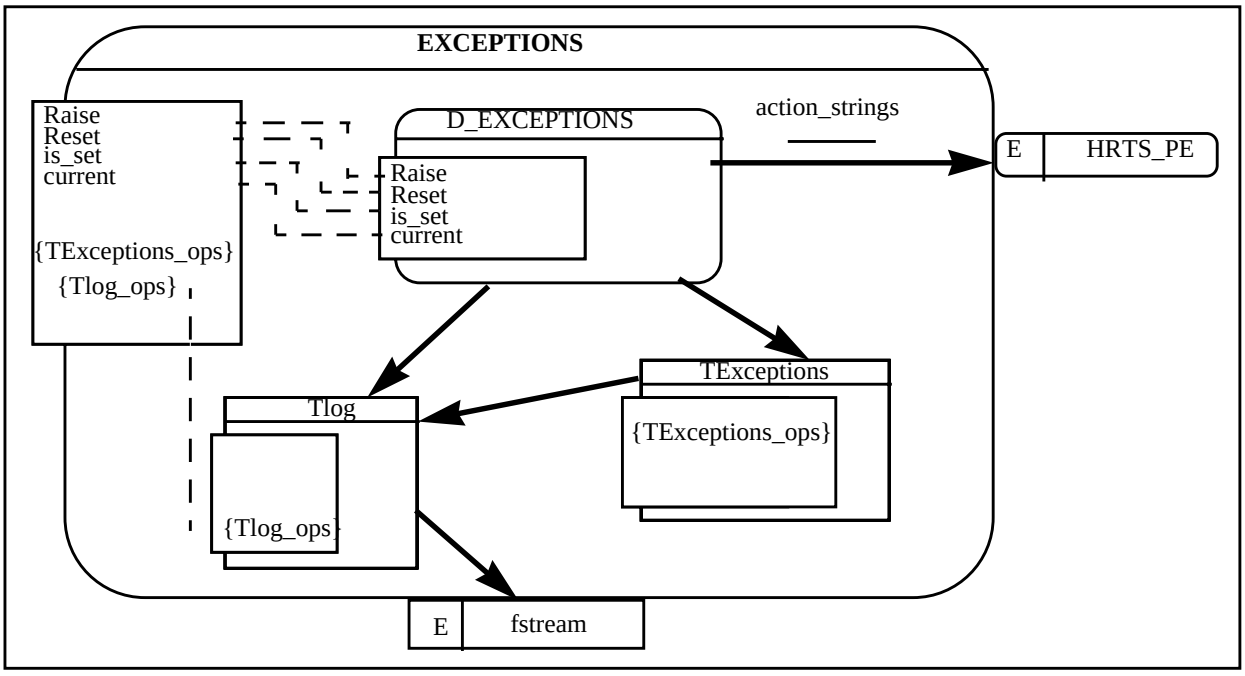


Figure 95 -Client_Server view of EXCEPTIONS object

J HOOD RUN TIME SUPPORT LIBRARY

A set of HOOD RUN TIME SUPPORT (HRTS) objects have been defined in order to provide a virtual machine independent of any target. The following HRTS objects and **class**es have been identified so far:

- Object Exceptions exceptions management for Ada and non ada target
- Object Fsms FSM management for OSTD support
- Object Obcs ObcsClient and ObcsServer part management
- Object Semaphores Semaphores for Mutual Exclusion management
- Object Timers Timers for time out management
- Object HRTSCHEDs Schedulers for Hard Real-Time attributes management
- Object HRTS_PE[3] a standard HOOD object used as a container of project and dynamic data for code generation as well as target dependent characteristic data and definitions.

Figure below gives an illustration of the HRTS library structure and its relationship to the OPCS and host OS.

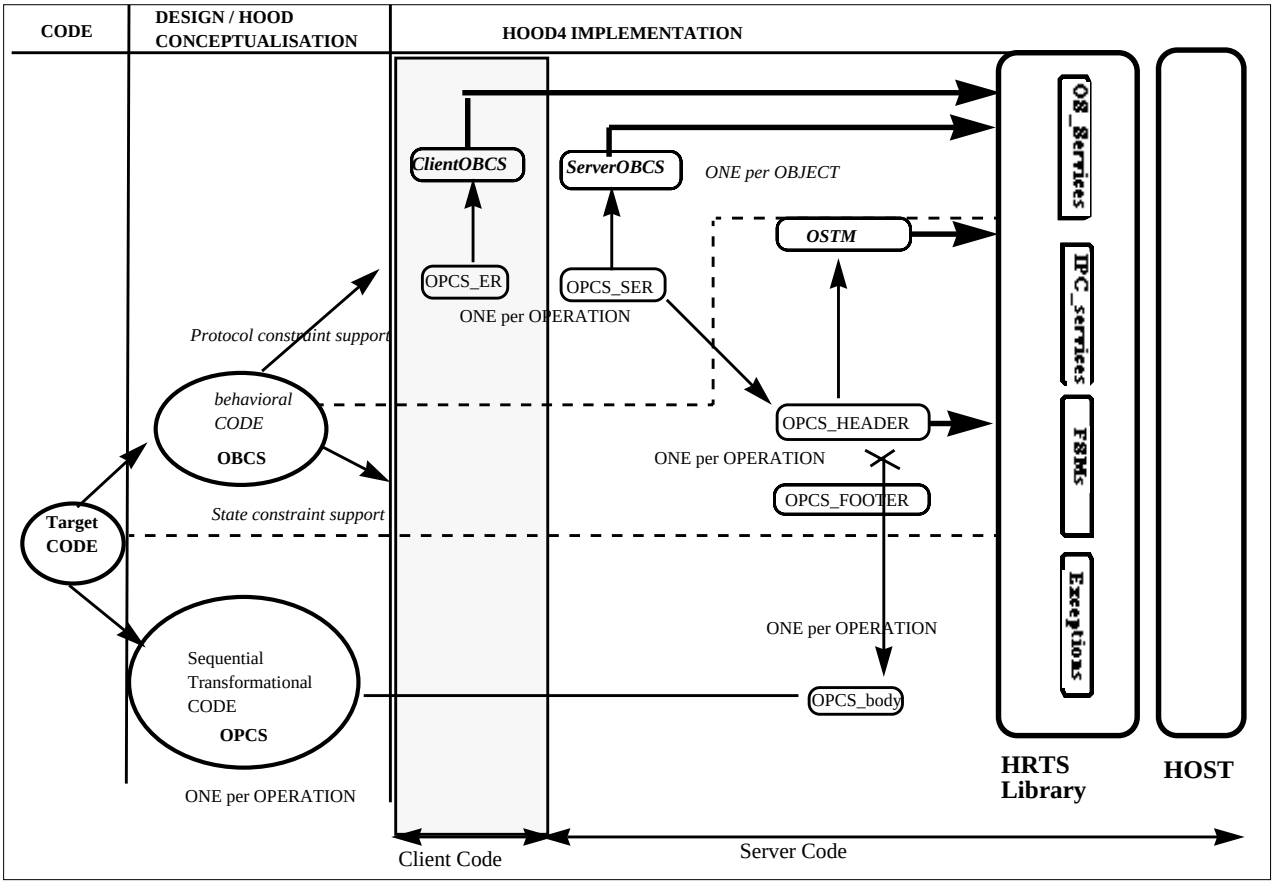


Figure 94 -The HOOD RUN TIME SUPPORT LIBRARY and Its relationships to OPCS and host OS.

[3]. stands for HRTS Project Environment

I.2.2 OSTM code illustration in C++

```
#ifndef STACK_OSTM_H
#define STACK_OSTM_H
#include "hrts/HRTS_PE.h"
template < class T_STATES,          class T_OPERATION,
        class T_FSM_ID    > class TFsm; // incomplete TFsm class redeclaration
class TStack_OSTM
{public:
    enum T_OSTDSTates    {          // Definition of T_OSTDSTates
     INIT,     N_E_N_F,     FULL,     EMPTY,     EXIT};

   enum T_OSTDOperations    {   // Definition of T_OSTDOperations
     START,     STOP,     PUSH,     POP,     RESET,
     GETMAXSIZE,     GETCURRENTSIZE,     GETSTATUS,     UNDEFINED};

   TString STATESStrings   [EXIT     + 1];
   TString OPERATIONStrings[UNDEFINED + 1];

    TFsm< T_OSTDSTates,          T_OSTDOperations,
            TPE::integer    > * fsm;            // definition of a generic TFsm dedicated for TStack
 public:
   TStack_OSTM();
   ~TStack_OSTM();
};
#endif /*STACK_OSTM_H*/
```

```
// body of TStack_OSTM
#include "STACK_OSTM.h"
#include "hrts/FSMs.h"
TStack_OSTM::TStack_OSTM()
{fsm = new TFsm< T_OSTDSTates, T_OSTDOperations, TPE::integer    >;/ /fsm dedicated for TStack
 STATESStrings[INIT] = TString("INIT");
 STATESStrings[N_E_N_F] = TString("N_E_N_F");
 STATESStrings[FULL] = TString("FULL");
 STATESStrings[EMPTY] = TString("EMPTY");
 STATESStrings[EXIT] = TString("EXIT");
 OPERATIONStrings[START] = TString("START");
 OPERATIONStrings[STOP] = TString("STOP");
 OPERATIONStrings[PUSH] = TString("PUSH");
 OPERATIONStrings[POP] = TString("POP");
 OPERATIONStrings[RESET] = TString("RESET");
 OPERATIONStrings[GETMAXSIZE] = TString("GETMAXSIZE");
 OPERATIONStrings[GETCURRENTSIZE] = TString("GETCURRENTSIZE");
 OPERATIONStrings[GETSTATUS] = TString("GETSTATUS");
 OPERATIONStrings[UNDEFINED] = TString("UNDEFINED");
 fsm->CREATE(2,             4,             INIT);
 fsm->TRANS(INIT,            START,            INIT);
 fsm->TRANS(INIT,            PUSH,             N_E_N_F);
 fsm->TRANS(INIT,            POP,            INIT);
 fsm->TRANS(N_E_N_F,            PUSH,            N_E_N_F);
 fsm->TRANS(N_E_N_F,            POP,            INIT);
 fsm->TRANS(INIT,            STOP,           INIT);
 fsm->TRANS(N_E_N_F,            STOP,            INIT);
}
TStack_OSTM::~TStack_OSTM()
 {delete fsm;}
```

I.2 OSTM SYNTAX

The code for FSM handling calls operations to create and define transitions of a finite state machine. This code can be automatically generated from a tool that analyzes an OSTD description enforcing the SIF syntax as defined in *Appendix G - Standard Interchange Format* - . We give an illustration of such code in ADA and C++ below.

I.2.1 OSTM Code Illustration in Ada

```
with HRTS_PE;
with TFSMs;use type HRTS_PE.T_Integer;
package Stack_OSTM is

    type T_OSTDState is (EMPTY, FULL, NON_E_F, STOPPED, UNDEFINED);

    type T_OSTDOperation is (START, STOP, PUSH, POP);-- only state constrained operations

NB_MAX_STATES:          constant HRTS_PE.T_Integer          := T_OSTDState'pos (T_OSTDState'last);
NB_MAX_OPERATIONS :     constant HRTS_PE.T_Integer          :=T_OSTDOperation'pos (T_OSTDOperation'last);
NB_MAX_TRANSITIONS:     constant HRTS_PE.T_Integer          := NB_MAX_STATES*NB_MAX_OPERATIONS;

package TFsm is new G_TFsm  (
                        T_Operation     =>  T_OSTDOperation,
                        T_State         =>  T_OSTDState,
                        NB_MAX_TRANSITIONS  =>          NB_MAX_TRANSITIONS);

function Stack_FSM return TFsm.Instance;

end Stack_OSTM;


package body Stack_OSTM is

function Stack_FSM return TFsm.Instance is
FSM: TFsm.Instance;
begin
   TFsm.Create (FSM, 4, 1, STOPPED);    -- initial state is STOPPED
    TFsm.Trans (FSM, EMPTY, START, EMPTY); -- no restarting an active stack
    TFsm.Trans (FSM, EMPTY, PUSH, NON_E_F);
    TFsm.Trans (FSM, EMPTY, STOP, STOPPED);
    TFsm.Trans (FSM, NON_E_F, PUSH, NON_E_F);
    TFsm.Trans (FSM, NON_E_F, POP, NON_E_F);
    TFsm.Trans FSM, NON_E_F, STOP, STOPPED);
    TFsm.Trans (FSM, FULL, POP, NON_E_F);
    TFsm.Trans (FSM, FULL, STOP, STOPPED);
    TFsm.Trans (FSM, STOPPED, START, EMPTY);
  return FSM;
end Stack_FSM;

end Stack_OSTM;
```

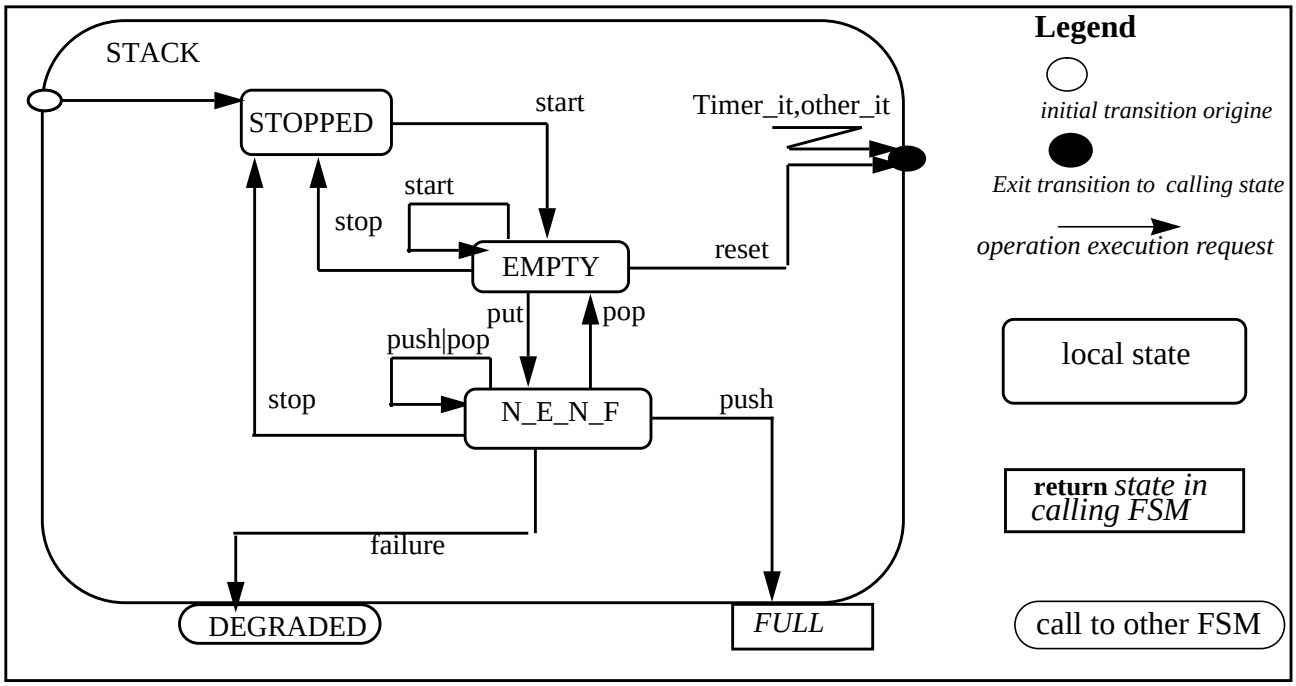



Figure 92 -Graphical Formalism for OSTD

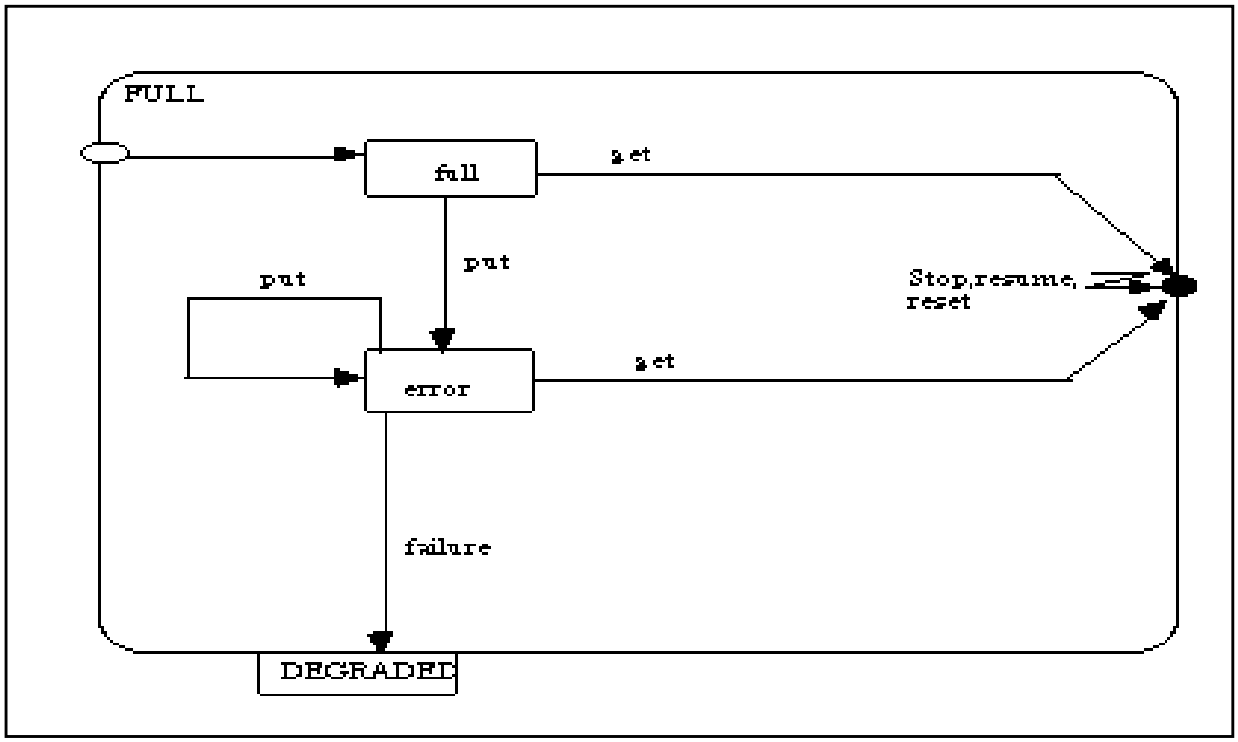


Figure 93 -Graphical Formalism for FSM named FULL

I OSTM AND OSTD DESCRIPTIONS AND SYNTAX

I.1 OSTD SYNTAX

The OSTD is a hierarchical description formalism of state transitions since it allows :

- to structure numerous transitions into a subset of less complexity
- to stay at the description level of HAREL's StateCharts[HAREL90] and several other implementation as in [OMT91], [BOOCH95] and others.

The following concepts are defined :

- an FSM is a set of states and transitions
- a subFSM is a special FSM which when executed, returns either to the calling state[2] or to a specific «**return** state» of the calling FSM.
  Additionnaly the trigger that made **return** from a FSM, may be fired again in the new FSM .

Figure 92 - *Graphical Formalism for OSTD* - illustrates a graphical formalism derived from the OSTD given in Figure 5 - *Example of an Object State Transition Diagram* - . The graphical items are the following :

- an FSM is defined as rounded corner box, **with** its name at the top left corner
- subFSM appear as uncle boxes
- states are little boxes holding the name of the state
- transitions are labelled arrows **with** (possibly ored) ER names
- transitions can cycle on a state
- transitions to and back uncles are bolded (to show a call to a subFSM)
- initial state is marked **with** a trigger (possibly labelled if several initial states)
- **return** transition to calling FSM are transitions going from a child state to the parent border. (seeFigure 93 - *Graphical Formalism for FSM named FULL* -)

A short hand is provided to specify triggers that apply to each state and make **return** from current FSM as an annotated trigger. (see Figure 93 - *Graphical Formalism for FSM named FULL* -)

2. -- exactly like subprogram calls!

ODS-14. The *internals* of the *object_descriptor*(3) of an OP_CONTROL contains the following sections :
entity_declaration_section(39)
[*internal_obcs_section*(29)]
operation_control_structures(30)--
The *operation_control_structures* shall contain only one *opcs* alternative.

ODS-15. The *control_structure* alternative shall contain the following sections :
description_section(7)
used_operation_section(33)
propagated_exception_section(34)
handled_exception_section(35)
pseudo_code_section(36)
code_section(37)

ODS-16. The *raised_by_clause*(50) shall only appear in the *exception_declaration*(47) of a *provided_interface_section*(14).

ODS-17. The *membership_clause*(49) shall only appear in the *operation_declaration*(45) or *operation_set_declaration*(46) of a *provided_interface_section*(14).

ODS-18. The **DATA** alternatives **with**in an *entity_declaration_section*(39) shall only appear **with**in the *internals*(19) of a terminal object, **class** *object_descriptor*(3).

ODS-19. The alternative which contains the value **IMPLEMENTED_BY** in the alternatives of the *entity_declarations_section*(39) shall only appear in the *internals*(19) of the *object_descriptor*(3) of a non terminal object or generic.

ODS-20. The alternative which contains the value **IMPLEMENTED_BY** in the alternatives of the *internal_obcs_section*(29) shall only appear in the *internals*(19) of the *object_descriptor*(3) of a non terminal active object or generic.

ODS-21. The **OPERATION_SETS** and **EXCEPTIONS** alternatives **with**in an *entity_declaration_section*(39) shall only appear **with**in the *internals*(19) of non terminal object, generic or OP_CONTROL *object_descriptor*(3).

ODS-22. The ODS of a VIRTUAL NODE is an *object_descriptor*(3) composed of the following sections:
module_kind(4) *object_identifier* ***IS*** *[annotation_list object_kind*(5) *]*
description_section(7)
implementation_constraints_section(8)
provided_interface_section(14)
required_interface_section(15)
data_flow_section(10)
exception_flow_section(11)
internals(19)

data_flow_section(10)
exception_flow_section(11)

ODS-5. The ODS of a PASSIVE object, **class**, generic or instance contains the same sections as the ACTIVE one, but OBCS subsections are reduced to the OSTD and OSTM fields .

ODS-6. Within an *object_descriptor*(3) section the *object_kind*(5) of a **class** shall not contain the value **ENVIRONMENT**.

ODS-7. The ODS of an object viewed as ENVIRONMENT is an *object_descriptor*(3) **with**out *internals*(19) alternative.

ODS-8. The ODS of an OP_CONTROL is an *object_descriptor*(3) composed of the following sections:
module_kind(4) *object_identifier* ***IS*** *[annotation_list object_kind*(5) *]*
description_section(7)
implementation_constraints_section(8)
required_interface_section(15)
data_flow_section(10)
exception_flow_section(11)
internals(19)

ODS-9. The **FORMAL_PARAMETERS** alternative in the *required_object_header*(17) shall only appear once **with**in the *required_interface_section*(15) of a **descendant of a** GENERIC.

ODS-10. The *instance_range*(57) alternative can only appear in the *object_descriptor*(3) of an **INSTANCE_OF** of a GENERIC.

ODS-11. The *internals* of the *object_descriptor*(3)of a TERMINAL ACTIVE or PASSIVE object, or **class** shall contain the following sections:
[*child_object_section*(20)]
{*entity_declaration_section*(39)}
internal_obcs_section(29)
operation_control_structures(30)

ODS-12. The *child_object_section* shall not be present for **class**es or generic **class**es, or if present it must have **NONE** as content.

ODS-13. The *internals* of the *object_descriptor*(3) of a NON TERMINAL ACTIVE or PASSIVE object (except **class**es and generic **class**es) shall contain the following sections:
child_object_section(20)
{*entity_declaration_section*(39)}
internal_obcs_section(29)
The *child_object_section* shall not have **NONE** as content.

H ODS DEFINITION

The ODS is a set of textual sections constrained by the following rules applied to the BNF definition of the SIF (see appendix D).

In all rules, *italic*$^{(n)}$ refers to BNF syntactic category number n.

ODS-1. The ODS of an OBJECT or CLASS (either ACTIVE or PASSIVE) is an *object_descriptor*$^{(3)}$ composed of the following sections:

ODS-2. *module_kind*$^{(4)}$ *object_identifier* ***IS*** *[annotation_list object_kind*$^{(5)}$ *]*
-description_section$^{(7)}$
implementation_constraints_section$^{(8)}$
provided_interface_section$^{(14)}$
visible_obcs_section$^{(21)}$
required_interface_section$^{(15)}$
data_flow_section$^{(10)}$
exception_flow_section$^{(11)}$
internals$^{(19)}$

ODS-3. The ODS of a GENERIC is an *object_descriptor*$^{(3)}$ composed of the following sections:
module_kind$^{(4)}$ *object_identifier* ***IS*** *[annotation_list object_kind*$^{(5)}$ *]*
formal_parameter_section$^{(51)}$
description_section$^{(7)}$
implementation_constraints_section$^{(8)}$
provided_interface_section$^{(14)}$
visible_obcs_section$^{(21)}$
required_interface_section$^{(15)}$
data_flow_section$^{(10)}$
exception_flow_section$^{(11)}$
internals$^{(19)}$

ODS-4. The ODS of an INSTANCE_OF a GENERIC is an *object_descriptor*$^{(3)}$ composed of the following sections:
module_kind$^{(4)}$ *object_identifier* ***IS*** *[annotation_list object_kind*$^{(5)}$ *]*
instance_range$^{(57)}$
actual_parameters_section$^{(52)}$
description_section$^{(7)}$
implementation_constraints_section$^{(8)}$
provided_interface_section$^{(14)}$
visible_obcs_section$^{(21)}$
required_interface_section$^{(15)}$

handledExceptionsSectionOpt : | handledExceptionsSection;
implementationConstraintsSectionOpt : | implementationConstraintsSection;
INHERITEDOpt : | INHERITED;
instanceRangeOpt : | instanceRange;
internalsOpt : | internals;
internalObcsSectionOpt : | internalObcsSection;
membershipClauseOpt : | membershipClause;
objectDescriptorRpt1 : objectDescriptor | objectDescriptorRpt1 objectDescriptor;
opcsRpt1 : opcs | opcsRpt1 opcs;
operationAssociationRpt1 : operationAssociation | operationAssociationRpt1 operationAssociation;
operationControlStructuresOpt : | operationControlStructures;
operationDeclarationRpt1 : operationDeclaration | operationDeclarationRpt1 operationDeclaration;
opParameterPartOpt : | opParameterPart;
opReturnPartOpt : | opReturnPart;
opSetDeclarationRpt1 : opSetDeclaration | opSetDeclarationRpt1 opSetDeclaration;
opTypeQualifierOpt : | opTypeQualifier;
ostdSectionOpt : | ostdSection;
parameterAssociationSectionRpt1 : parameterAssociationSection | parameterAssociationSectionRpt1 parameterAssociationSection;
pragmaParameterPartOpt : | pragmaParameterPart;
pragmaRpt0 : | pragmaRpt0 pragma;
propagatedExceptionsSectionOpt : | propagatedExceptionsSection;
protocolOpt : | protocol;
providedInterfaceSectionOpt : | providedInterfaceSection;
pseudoCodeSectionOpt : | pseudoCodeSection;
ptrOrRefOpTypeQualifierRpt1 : ptrOrRefOpTypeQualifier | ptrOrRefOpTypeQualifierRpt1 ptrOrRefOpTypeQualifier;
r10Rpt0 : | r10Rpt0 r10;
r11Rpt0 : | r11Rpt0 r11;
r12Opt : | r12;
r13Rpt0 : | r13Rpt0 r13;
r1Opt : | r1;
r2Rpt1 : r2 | r2Rpt1 r2;
r3Rpt1 : r3 | r3Rpt1 r3;
r4Rpt1 : r4 | r4Rpt1 r4;
r5Rpt1 : r5 | r5Rpt1 r5;
r6Rpt1 : r6 | r6Rpt1 r6;
r7Rpt1 : r7 | r7Rpt1 r7;
r8Rpt0 : | r8Rpt0 r8;
r9Rpt0 : | r9Rpt0 r9;
requiredEntitySectionRpt1 : requiredEntitySection | requiredEntitySectionRpt1 requiredEntitySection;
requiredInterfaceSectionOpt : | requiredInterfaceSection;
requiredObjectRpt1 : requiredObject | requiredObjectRpt1 requiredObject;
SEMICOLONOpt : | SEMICOLON;
sifHeaderOpt : | sifHeader;
tdOpt : | td;
typeAttributesOpt : | **type**Attributes;
typeDeclarationRpt1 : **type**Declaration | **type**DeclarationRpt1 **type**Declaration;
usedOperationsSectionOpt : | usedOperationsSection;
variableDeclarationRpt1 : variableDeclaration | variableDeclarationRpt1 variableDeclaration;
visibleObcsSectionOpt : | visibleObcsSection;

```
r13
    : comma pragmaParameter;

pragmaParameter /* (75) */
    : identifier RARROW pragmaParameterValue
    | pragmaParameterValue;

pragmaParameterValue /* (76) */
    : opReference
    | numericLiteral
    | freeText;

identifier /* (0.1) */
    : NOM;

integer /* (0.2) */
    : ENTIER;

real /* (0.3) */
    : REEL;

stringLiteral /* (0.4) */
    : STRING2;

text /* (0.5) */
    : FREETEXT;

/* optional and repeated items : */

ABSTRACTOpt :  | ABSTRACT;
actualParametersSectionOpt :  | actualParametersSection;
ASSIGNAndTdOpt :  | ASSIGNAndTd;
associationRpt1 : association | associationRpt1 association;
childObjectSectionOpt :  | childObjectSection;
codeSectionOpt :  | codeSection;
codeSubSectionRpt0 :  | codeSubSectionRpt0 codeSubSection;
COLONAndReferenceOpt :  | COLONAndReference;
CommaAndIdentifierRpt0 :  | CommaAndIdentifierRpt0 CommaAndIdentifier;
COMMAAndReferenceRpt0 :  | COMMAAndReferenceRpt0 COMMAAndReference;
COMMAOpt :  | COMMA;
concurrencyOpt :  | concurrency;
constantDeclarationRpt1 : constantDeclaration | constantDeclarationRpt1 constantDeclaration;
constrainedOperationRpt1 : constrainedOperation | constrainedOperationRpt1 constrainedOperation;
constrainedOperationSectionOpt :  | constrainedOperationSection;
constraintAttributeRpt0 :  | constraintAttributeRpt0 constraintAttribute;
constraintAttributeRpt1 : constraintAttribute | constraintAttributeRpt1 constraintAttribute;
dataFlowSectionOpt :  | dataFlowSection;
descriptionSectionOpt :  | descriptionSection;
entityDeclarationSectionRpt0 :  | entityDeclarationSectionRpt0 entityDeclarationSection;
entityDeclarationSectionRpt1 : entityDeclarationSection | entityDeclarationSectionRpt1 entityDeclarationSection;
ENVIRONMENTOpt :  | ENVIRONMENT;
exceptionDeclarationRpt1 : exceptionDeclaration | exceptionDeclarationRpt1 exceptionDeclaration;
exceptionFlowSectionOpt :  | exceptionFlowSection;
flowRpt1 : flow | flowRpt1 flow;
formalParametersSectionOpt :  | formalParametersSection;
freeTextOpt :  | freeText;
GENERICOpt :  | GENERIC;
```

```
r10
    : SEMICOLON opParameter;

opReturnPart /* (62) */
    : RETURN reference opTypeQualifierOpt;

opParameter /* (63) */
    : identifier COLON opParameterMode reference opTypeQualifierOpt ASSIGNAndTdOpt;

ASSIGNAndTd
    : ASSIGN td;

opTypeQualifier /* (64) */
    : BYVALUE
    | ptrOrRefOpTypeQualifierRpt1;

ptrOrRefOpTypeQualifier /* (65) */
    : BYPOINTER
    | BYREFERENCE;

opParameterMode /* (66) */
    : IN
    | OUT
    | IN OUT;

numericLiteral /* (67) */
    : integer
    | real;

annotationList /* (68) */
    : r11Rpt0;

r11
    : freeText
    | pragma;

td /* (69) */
    : text;

freeText /* (70) */
    : text;

semiColon /* (71) */
    : r12Opt;

r12
    : SEMICOLON annotationList;

comma /* (72) */
    : COMMAOpt;

pragma /* (73) */
    : PRAGMA identifier pragmaParameterPartOpt SEMICOLONOpt;

pragmaParameterPart /* (74) */
    : LP pragmaParameter r13Rpt0 RP;
```

```
variableDeclaration /* (48) */
    : identifier COLONAndReferenceOpt tdOpt semiColon;

membershipClause /* (49) */
    : MEMBEROF identifier;

raisedByClause /* (50) */
    : RAISEDBY opReference r9Rpt0;

r9
    : comma opReference;

formalParametersSection /* (51) */
    : FORMALPARAMETERS entityDeclarationSectionRpt1
    | FORMALPARAMETERS NONE;

actualParametersSection /* (52) */
    : PARAMETERS parameterAssociationSectionRpt1
    | PARAMETERS NONE;

parameterAssociationSection /* (53) */
    : TYPES associationRpt1
    | TYPES NONE
    | CONSTANTS associationRpt1
    | CONSTANTS NONE
    | OPERATIONS operationAssociationRpt1
    | OPERATIONS NONE
    | OPERATIONSETS associationRpt1
    | OPERATIONSETS NONE;

association /* (54) */
    : identifier RARROW referenceOrTd semiColon;

operationAssociation /* (55) */
    : operationIdentifier RARROW opReference semiColon;

referenceOrTd /* (56) */
    : reference
    | td;

instanceRange /* (57) */
    : INSTANCERANGE integer DOT DOT integer;

operationIdentifier /* (58) */
    : identifier opParameterPartOpt opReturnPartOpt
    | stringLiteral opParameterPartOpt opReturnPartOpt;

reference /* (59) */
    : identifier DOT identifier
    | identifier;

opReference /* (60) */
    : identifier DOT operationIdentifier
    | operationIdentifier;

opParameterPart /* (61) */
    : LP opParameter r10Rpt0 RP;
```

```
    : identifier td
    | identifier NONE;

entityDeclarationSection /* (39) */
    : TYPES typeDeclarationRpt1
    | TYPES NONE
    | CONSTANTS constantDeclarationRpt1
    | CONSTANTS NONE
    | OPERATIONS operationDeclarationRpt1
    | OPERATIONS NONE
    | OPERATIONSETS opSetDeclarationRpt1
    | OPERATIONSETS NONE
    | EXCEPTIONS exceptionDeclarationRpt1
    | EXCEPTIONS NONE
    | DATA variableDeclarationRpt1
    | DATA NONE;

typeDeclaration /* (40) */
    : identifier ABSTRACTOpt classInheritance typeAttributesOpt tdOpt semiColon
    | identifier typeAttributesOpt tdOpt semiColon
    | identifier IMPLEMENTEDBY reference semiColon;

classInheritance /* (41) */
    : INHERITANCE reference COMMAAndReferenceRpt0
    | INHERITANCE NONE;

COMMAAndReference
    : COMMA reference;

typeAttributes /* (42) */
    : ATTRIBUTES attributeDeclaration r8Rpt0 ATTRIBUTES NONE;

r8
    : COMMA attributeDeclaration;

attributeDeclaration /* (43) */
    : identifier COLONAndReferenceOpt;

COLONAndReference
    : COLON reference;

constantDeclaration /* (44) */
    : identifier COLONAndReferenceOpt tdOpt semiColon
    | identifier IMPLEMENTEDBY reference semiColon;

operationDeclaration /* (45) */
    : operationIdentifier ABSTRACTOpt INHERITEDOpt membershipClauseOpt semiColon
    | operationIdentifier IMPLEMENTEDBY opReference semiColon;

opSetDeclaration /* (46) */
    : identifier membershipClauseOpt semiColon
    | identifier IMPLEMENTEDBY reference semiColon;

exceptionDeclaration /* (47) */
    : identifier tdOpt semiColon
    | identifier tdOpt raisedByClause SEMICOLON annotationList
    | identifier IMPLEMENTEDBY reference semiColon;
```

```
protocol /* (26) */
    : LSER
    | HSER
    | ASER
    | RASER
    | RLSER;

concurrency /* (27) */
    : MTEX
    | RWER
    | ROER;

constraintAttribute /* (28) */
    : TO numericLiteral
    | BYIT td;

internalObcsSection /* (29) */
    : OBJECTCONTROLSTRUCTURE controlStructure
    | OBJECTCONTROLSTRUCTURE IMPLEMENTEDBY identifier CommaAndIdentifierRpt0 semiColon
    | OBJECTCONTROLSTRUCTURE NONE;

CommaAndIdentifier
    : comma identifier;

operationControlStructures /* (30) */
    : OPERATIONCONTROLSTRUCTURES opcsRpt1
    | OPERATIONCONTROLSTRUCTURES NONE;

opcs /* (31) */
    : OPERATION operationIdentifier controlStructure ENDOPERATION operationIdentifier;

controlStructure /* (32) */
    : annotationList descriptionSectionOpt usedOperationsSectionOpt propagatedExceptionsSectionOpt handledExcep-
    tionsSectionOpt pseudoCodeSectionOpt codeSectionOpt;

usedOperationsSection /* (33) */
    : USEDOPERATIONS r4Rpt1
    | USEDOPERATIONS NONE;

propagatedExceptionsSection /* (34) */
    : PROPAGATEDEXCEPTIONS r2Rpt1
    | PROPAGATEDEXCEPTIONS NONE;

handledExceptionsSection /* (35) */
    : HANDLEDEXCEPTIONS r2Rpt1
    | HANDLEDEXCEPTIONS NONE;

pseudoCodeSection /* (36) */
    : PSEUDOCODE freeText
    | PSEUDOCODE NONE;

codeSection /* (37) */
    : CODE tdOpt codeSubSectionRpt0
    | CODE NONE;

codeSubSection /* (38) */
```

```
    | OPERATIONS pragmaRpt0 NONE
    | OPERATIONSETS pragmaRpt0 r5Rpt1
    | OPERATIONSETS pragmaRpt0 NONE
    | EXCEPTIONS pragmaRpt0 r6Rpt1
    | EXCEPTIONS pragmaRpt0 NONE;

r2
    : reference semiColon;

r3
    : reference semiColon;

r4
    : opReference semiColon;

r5
    : reference semiColon;

r6
    : reference semiColon;

internals /* (19) */
    : INTERNALS childObjectSectionOpt entityDeclarationSectionRpt0 internalObcsSectionOpt operationControl-
    StructuresOpt;

childObjectSection /* (20) */
    : OBJECTS r7Rpt1
    | OBJECTS NONE;

r7
    : identifier semiColon;

visibleObcsSection /* (21) */
    : OBJECTCONTROLSTRUCTURE annotationList descriptionSectionOpt ostdSectionOpt constrainedOperation-
    SectionOpt
    | OBJECTCONTROLSTRUCTURE NONE;

ostdSection /* (22) */
    : OSTD annotationList
    | OSTD NONE;

constrainedOperationSection /* (23) */
    : CONSTRAINEDOPERATIONS annotationList constrainedOperationRpt1
    | CONSTRAINEDOPERATIONS NONE;

constrainedOperation /* (24) */
    : operationIdentifier semiColon
    | operationIdentifier CONSTRAINEDBY constrainedLabel freeTextOpt semiColon;

constrainedLabel /* (25) */
    : STATE protocolOpt concurrencyOpt constraintAttributeRpt0
    | protocol concurrencyOpt constraintAttributeRpt0
    | concurrency constraintAttributeRpt0
    | constraintAttributeRpt1
    | freeText
    | ASERBYIT td
    | LSERTOER numericLiteral
    | HSERTOER numericLiteral;
```

```
objectTrailer /* (6) */
    : ENDOBJECT identifier annotationList
    | END identifier annotationList;

descriptionSection /* (7) */
    : DESCRIPTION annotationList
    | DESCRIPTION NONE;

implementationConstraintsSection /* (8) */
    : implementationConstraintsKeyword annotationList
    | implementationConstraintsKeyword NONE;

implementationConstraintsKeyword /* (9) */
    : IMPLEMENTATIONCONSTRAINTS
    | IMPLEMENTATIONORSYNCHRONIZATIONCONSTRAINTS
    | IMPLEMENTATIONORSYNCHRONISATIONCONSTRAINTS;

dataFlowSection /* (10) */
    : DATAFLOWS flowRpt1
    | DATAFLOWS NONE;

exceptionFlowSection /* (11) */
    : EXCEPTIONFLOWS flowRpt1
    | EXCEPTIONFLOWS NONE;

flow /* (12) */
    : identifier flowDirection identifier semiColon
    | identifier flowDirection OPERATION operationIdentifier semiColon;

flowDirection /* (13) */
    : LARROW
    | BIARROW
    | RARROW;

providedInterfaceSection /* (14) */
    : PROVIDEDINTERFACE entityDeclarationSectionRpt1
    | PROVIDEDINTERFACE NONE;

requiredInterfaceSection /* (15) */
    : REQUIREDINTERFACE requiredObjectRpt1
    | REQUIREDINTERFACE NONE;

requiredObject /* (16) */
    : requiredObjectHeader semiColon requiredEntitySectionRpt1
    | requiredObjectHeader semiColon NONE;

requiredObjectHeader /* (17) */
    : OBJECT identifier
    | FORMALPARAMETERS;

requiredEntitySection /* (18) */
    : TYPES pragmaRpt0 r2Rpt1
    | TYPES pragmaRpt0 NONE
    | CONSTANTS pragmaRpt0 r3Rpt1
    | CONSTANTS pragmaRpt0 NONE
    | OPERATIONS pragmaRpt0 r4Rpt1
```

G.4 Example of YACC implementation

This section gives a possible implementation of the HOOD Standard Interchange Format in YACC syntax[1].

```
%token ABSTRACT ACTIVE ASER ASERBYIT ATTRIBUTES BYIT BYPOINTER BYREFERENCE BYVALUE
    CLASS CODE CONSTANTS CONSTRAINEDBY CONSTRAINEDOPERATIONS DATA DATAFLOWS DE-
    SCRIPTION END ENDOBJECT ENDOPERATION ENVIRONMENT EXCEPTIONS EXCEPTIONFLOWS
    FORMALPARAMETERS GENERIC HANDLEDEXCEPTIONS HSER HSERTOER IMPLEMENTATIONCON-
    STRAINTS IMPLEMENTATIONORSYNCHRONISATIONCONSTRAINTS IMPLEMENTATIONORSYN-
    CHRONIZATIONCONSTRAINTS IMPLEMENTEDBY IN INHERITANCE INHERITED INSTANCEOF
    INSTANCERANGE INTERNALS IS LSER LSERTOER MEMBEROF MTEX NONE OBJECT OBJECTS OB-
    JECTCONTROLSTRUCTURE OPERATION OPERATIONS OPERATIONCONTROLSTRUCTURES OPERA-
    TIONSETS OPCONTROL OSTD OUT PARAMETERS PASSIVE PRAGMA PROPAGATEDEXCEPTIONS
    PROTOCOL PROVIDEDINTERFACE PSEUDOCODE RAISEDBY RASER REQUIREDINTERFACE RE-
    TURN RLSER ROER RWER SIFVERSION STATE TARGETLANGUAGE TO TOOLNAME TOOLVERSION
    TYPES USEDOPERATIONS VIRTUALNODE

%token LP RP COMMA DOT COLON ASSIGN SEMICOLON LARROW BIARROW RARROW

%token ENTIER FREETEXT NOM REEL STRING2

%%

sif /* (1) */
    : sifHeaderOpt objectDescriptorRpt1;

sifHeader /* (2) */
    : SIFVERSION freeText semiColon TOOLNAME freeText semiColon TOOLVERSION freeText semiColon TAR-
    GETLANGUAGE freeText semiColon;

objectDescriptor /* (3) */
    : moduleKind identifier IS annotationList r1Opt formalParametersSectionOpt instanceRangeOpt actualParame-
    tersSectionOpt descriptionSectionOpt implementationConstraintsSectionOpt providedInterfaceSectionOpt visible-
    ObcsSectionOpt requiredInterfaceSectionOpt dataFlowSectionOpt exceptionFlowSectionOpt internalsOpt
    objectTrailer;

r1
    : objectKind annotationList;

moduleKind /* (4) */
    : OBJECT
    | CLASS
    | OPCONTROL
    | VIRTUALNODE;

objectKind /* (5) */
    : GENERICOpt ENVIRONMENTOpt ACTIVE
    | GENERICOpt ENVIRONMENTOpt PASSIVE
    | INSTANCEOF identifier
    | PROTOCOL freeText
    | CLASS ENVIRONMENTOpt ACTIVE
    | CLASS ENVIRONMENTOpt PASSIVE
    | OPCONTROL
    | VIRTUALNODE;
```

[1] special thanks to [TNI] for providing us **with** this YACC implementation.

STATE_description(78){STATE_description}
END_FSM *operation*_identifier

(78) STATE_description ::=
STATE state_identifier
TRANSITION_description(79){TRANSITION_description}
END_STATE *operation*_identifier

(79) TRANSITION_description ::=
TRANS *constrained*_operation_identifier *STATE*_identifier |
PUSHTRANS *constrained*_operation_identifier *FSM*_identifier |
POPTRANS *constrained*_operation_identifier [*STATE*_identifier] |

miscellaneous

(67) numeric_literal ::=
integer | real

(68) annotation_list ::=
{free_text | pragma}

(69) td ::=
text

(70) free_text ::=
text

(71) semi_colon ::=
[";" annotation_list]

(72) comma ::=
[","]

pragma

(73) pragma ::=
PRAGMA *pragma*_identifier [pragma_parameter_part] [**";"**]

(74) pragma_parameter_part ::=
"(" pragma_parameter(75) {comma pragma_parameter} ")"

(75) pragma_parameter ::=
*parameter*_identifier "**=>**" pragma_parameter_value(76)
-- named association
| pragma_parameter_value
-- positional association

(76) pragma_parameter_value ::=
op_reference(60)
-- allows reference to operations or other entities
| numeric_literal
| free_text

default FSM description syntax

(77) FSM_description ::=
FSM FSM_identifier

(57) instance_range ::=
INSTANCE_RANGE integer "." "." integer

identifiers and references

(58) operation_identifier ::=
identifier [op_parameter_part(61)] [op_return_part(62)]
| string_literal [op_parameter_part] [op_return_part]

(59) reference ::=
*object*_identifier "." *entity*_identifier
| *entity*_identifier

(60) op_reference ::=
*object*_identifier "." operation_identifier(58)
| operation_identifier(58)

(61) op_parameter_part ::=
"(" op_parameter {";" op_parameter(63)} ")"

(62) op_return_part ::=
RETURN *type*_reference(59) {op_type_qualifier(64)}

(63) op_parameter ::=
*parameter*_identifier ":" op_parameter_mode(66) *type*_reference
[op_type_qualifier(64)] [":=" td_*expression*]

(64) op_type_qualifier ::=
BY_VALUE
| ptr_or_ref_op_type_qualifier(65) {ptr_or_ref_op_type_qualifier}

(65) ptr_or_rep_op_type_qualifier ::=
BY_POINTER -- C/C++ pointer
BY_REFERENCE -- C++ reference

(66) op_parameter_mode ::=
IN
| **OUT**
| **IN OUT**

instances and generics

(51) formal_parameters_section ::=
FORMAL_PARAMETERS
entity_declaration_section(39) {entity_declaration_section}
| **FORMAL_PARAMETERS**
NONE

(52) actual_parameters_section ::=
PARAMETERS
param_association_section(53) {param_association_section}
| **PARAMETERS**
NONE

(53) param_association_section ::=
TYPES
*type*_association(54) {*type*_association}
| **TYPES**
NONE
| **CONSTANTS**
*constant*_association(54) {*constant*_association}
| **CONSTANTS**
NONE
| **OPERATIONS**
operation_association(55) {operation_association}
| **OPERATIONS**
NONE
| **OPERATION_SETS**
*op_set*_association(54) {*op_set*_association}
| **OPERATION_SETS**
NONE

(54) association ::=
*entity*_identifier "=>" reference_or_td semi_colon

(55) operation_association ::=
operation_identifier(58) "=>" op_reference semi_colon

(56) reference_or_td ::=
reference
| td_*expression*

ATTRIBUTES
attribute_declaration(43) {“,” attribute_declaration }
| **ATTRIBUTES**
NONE

(43) attribute_declaration ::=
*attribute*_identifier [":" *type*_reference]

(44) constant_declaration ::=
*constant*_identifier [":" *type*_reference]
[td_*constant_definition*] semi_colon
| *constant*_identifier **IMPLEMENTED_BY**
*constant*_reference semi_colon

(45) operation_declaration ::=
operation_identifier [**ABSTRACT**] [**INHERITED**]
[membership_clause(49)] semi_colon
| operation_identifier(58) **IMPLEMENTED_BY** op_reference(60) semi_colon

(46) operation_set_declaration ::=
*op_set*_identifier [membership_clause(49)] semi_colon
| *op_set*_identifier **IMPLEMENTED_BY**
*op_set*_reference semi_colon

(47) exception_declaration ::=
*exception*_identifier [td_*exception_definition*] semi_colon
| *exception*_identifier [td_*exception_definition*] raised_by_clause(50) ";"
annotation_list
-- (see rule ODS-15)
| *exception*_identifier **IMPLEMENTED_BY**
*exception*_reference semi_colon

(48) variable_declaration ::=
*variable*_identifier [":" *type*_reference]
[td_*variable_definition*] semi_colon

(49) membership_clause ::=
MEMBER_OF *op_set*_identifier
-- (see rule ODS-16)

(50) raised_by_clause ::=
RAISED_BY op_reference(60) {comma op_reference}

entities

(39) entity_declaration_section ::=
TYPES
type_declaration(40) {type_declaration}
| **TYPES**
NONE
| **CONSTANTS**
constant_declaration(44) {constant_declaration}
| **CONSTANTS**
NONE
| **OPERATIONS**
operation_declaration(45) {operation_declaration}
| **OPERATIONS**
NONE
| **OPERATION_SETS** -- (see rule ODS-20)
operation_set_declaration(46) {operation_set_declaration}
| **OPERATION_SETS**
NONE
| **EXCEPTIONS** -- (see rule ODS-20)
exception_declaration(47) {exception_declaration}
| **EXCEPTIONS**
NONE
| **DATA**
variable_declaration(48) {variable_declaration}
-- internals of terminal objects/**class** or op_control
-- (see rule ODS-17)
| **DATA**
NONE

(40) type_declaration ::=
class_identifier [**ABSTRACT**] **class**_inheritance(41)
[type_attributes(42)] [td_**class**_*definition*] semi_colon
| *type*_identifier [type_attributes] [td_*type_definition*] semi_colon
| **class**_*or_type*_identifier **IMPLEMENTED_BY** **class**_*or_type*_reference semi_colon

(41) **class**_inheritance ::=
INHERITANCE
*inherited*_**class**_reference {“,” *inherited*_**class**_reference }
| **INHERITANCE**
NONE

(42) type_attributes ::=

[used_operations_section(33)]
[propagated_exceptions_section(34)]
[handled_exceptions_section(35)]
[pseudo_code_section(36)]
[code_section(37)]

(33) used_operations_section ::=
USED_OPERATIONS
op_reference(60)
{semi_colon op_reference} semi_colon
| **USED_OPERATIONS**
NONE

(34) propagated_exceptions_section ::=
PROPAGATED_EXCEPTIONS
*exception*_reference
{semi_colon *exception*_reference}semi_colon
| **PROPAGATED_EXCEPTIONS**
NONE

(35) handled_exceptions_section ::=
HANDLED_EXCEPTIONS
*exception*_reference
{semi_colon *exception*_reference} semi_colon
| **HANDLED_EXCEPTIONS**
NONE

(36) pseudo_code_section ::=
PSEUDO_CODE
free_text
| **PSEUDO_CODE**
NONE

(37) code_section ::=
CODE
[td_*code*] {code_sub_section}
| **CODE**
NONE

(38) code_sub_section ::=
*sub_section*_identifier td_*code*
| *sub_section*_identifier **NONE**

| **LSER_TOER** numeric_literal -- for HOOD3 compatibility
| **HSER_TOER** numeric_literal -- for HOOD3 compatibility

(26) protocol ::=
LSER
| **HSER**
| **ASER**
| **RASER**
| **RLSER**

(27) concurrency ::=
MTEX -- mutual exclusion
| **RWER** -- writer exclusion
| **ROER** -- reader exclusion

(28) constraint_attribute ::=
TO numeric_literal -- timeout constraint
| **BY_IT** td_*interrupt_label* -- hardware interrupt constraint

(29) internal_obcs_section ::=
OBJECT_CONTROL_STRUCTURE
control_structure(32) -- only for terminal object
| **OBJECT_CONTROL_STRUCTURE** -- (see rule ODS-19)
IMPLEMENTED_BY *object*_identifier -- only for non terminal objects
{comma *object*_identifier} semi_colon -- list of child objects
| **OBJECT_CONTROL_STRUCTURE**
NONE

(30) operation_control_structures ::=
OPERATION_CONTROL_STRUCTURES
opcs(31) {opcs}
| **OPERATION_CONTROL_STRUCTURES**
NONE

(31) opcs ::=
OPERATION operation_identifier(58)
control_structure(32)
END_OPERATION operation_identifier

(32) control_structure ::=
annotation_list(68)
[description_section(7)]

(20) child_object_section ::=
OBJECTS
*object*_identifier semi_colon {*object*_identifier semi_colon}
| **OBJECTS**
NONE

control structures

(21) visible_obcs_section ::=
OBJECT_CONTROL_STRUCTURE
annotation_list(68)
[description_section(7)]
[ostd_section(22)]
[constrained_operation_section(23)]
| **OBJECT_CONTROL_STRUCTURE**
NONE

(22) ostd_section ::=
OSTD
annotation_list(68) -- free_texts for FSM_description
OSTD
NONE

(23) constrained_operation_section ::=
CONSTRAINED_OPERATIONS
annotation_list(68)
constrained_operation(24) {constrained_operation}
| **CONSTRAINED_OPERATIONS**
NONE

(24) constrained_operation ::=
operation_identifier(58) semi_colon -- for HOOD3 compatibility
| operation_identifier **CONSTRAINED_BY**
constrained_label(25) [free_text] semi_colon

(25) constrained_label ::=
STATE [protocol(26)] [concurrency(27)] {constraint_attribute(28)}
| protocol(26) [concurrency(27)] {constraint_attribute(28)}
| concurrency(27) {constraint_attribute(28)}
| constraint_attribute(28) {constraint_attribute}
| *label*_free_text
| **ASER_BY_IT** td_*interrupt_label* -- for HOOD3 compatibility

(17) required_object_header ::=
OBJECT *object*_identifier
| **FORMAL_PARAMETERS** -- (see rule ODS-8)

(18) required_entity_section ::=
TYPES {pragma[73]} -- can be used for graphical pragmas
*type*_reference semi_colon
{*type*_reference semi_colon}
| **TYPES** {pragma}
NONE
| **CONSTANTS** {pragma}
*constant*_reference semi_colon
{*constant*_reference semi_colon}
| **CONSTANTS** {pragma}
NONE
| **OPERATIONS** {pragma}
op_reference[60] semi_colon
{op_reference semi_colon}
| **OPERATIONS** {pragma}
NONE
| **OPERATION_SETS** {pragma}
*op_set*_reference semi_colon
{*op_set*_reference semi_colon}
| **OPERATION_SETS** {pragma}
NONE
| **EXCEPTIONS** {pragma}
*exception*_reference semi_colon
{*exception*_reference semi_colon}
| **EXCEPTIONS** {pragma}
NONE

internals

(19) internals ::=
INTERNALS
[child_object_section][20] -- only for non terminal objects
-- (see rules ODS-10..13)
{entity_declaration_section[39]} -- only for terminal objects
-- (see rules ODS-16..19)
[internal_obcs_section[29]] -- only for active objects
-- (see rules ODS-10/12)
[operation_control_structures[30]]

(11) exception_flow_section ::=
    EXCEPTION_FLOWS
        flow(12) {flow}
  | **EXCEPTION_FLOWS**
        NONE

(12) flow ::=
    *flow*_identifier flow_direction(13) *object*_identifier semi_colon
  | *flow*_identifier flow_direction **OPERATION** operation_identifier semi_colon

(13) flow_direction ::=
    "**<=**" -- for object flows : data and exception flows, or
              -- for operation flows : out parameters, or propagated exceptions
  | "**<=>**" -- for object flows : data flows, or
              -- for operation flows : in out parameters
  | "**=>**" -- for object flows : data flows, or
              -- for operation flows : in parameters

provided interface

(14) provided_interface_section ::=
    PROVIDED_INTERFACE
        entity_declaration_section(39) {entity_declaration_section}
  | **PROVIDED_INTERFACE**
        NONE

required interface

(15) required_interface_section ::=
    REQUIRED_INTERFACE
        required_object(16) {required_object}
  | **REQUIRED_INTERFACE**
        NONE

(16) required_object ::=
    required_object_header(17) semi_colon
        required_entity_section(18) {required_entity_section}
  | required_object_header(17) semi_colon
        NONE

(5) object_kind ::=
[**GENERIC**] [**ENVIRONMENT**] **ACTIVE**
| [**GENERIC**] [**ENVIRONMENT**] **PASSIVE**
| **INSTANCE_OF** *generic*_identifier
| **PROTOCOL** *protocol*_free_text -- only for virtual nodes (optional)
| **CLASS** [**ENVIRONMENT**] **ACTIVE** --only for HOOD3 **class**es
| **CLASS** [**ENVIRONMENT**] **PASSIVE** -- only for HOOD3 **class**es
| **OP_CONTROL** -- only for HOOD3 OP_CONTROLs
| **VIRTUAL_NODE** -- only for HOOD3 virtual nodes

(6) object_trailer ::=
END_OBJECT *object*_identifier annotation_list
-- for HOOD3 compatibility
| **END** *object*_identifier annotation_list
-- mandatory HOOD4 keyword

Description and flows

(7) description_section ::=
DESCRIPTION
annotation_list(68)
| **DESCRIPTION**
NONE

(8) implementation_constraints_section ::=
implementation_constraints_keyword(9)
annotation_list(68)
| implementation_constraints_keyword(9)
NONE

(9) implementation_constraints_keyword ::=
IMPLEMENTATION_CONSTRAINTS -- mandatory HOOD4 keyword
| **IMPLEMENTATION_OR_SYNCHRONIZATION_CONSTRAINTS**
-- only for backward compatibility
| **IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINT**
-- only for backward compatibility

(10) data_flow_section ::=
DATAFLOWS
flow(12) {flow}
| **DATAFLOWS**
NONE

System Configuration

(1) sif ::=
[sif_header](2) -- optional for HOOD3, mandatory in HOOD4
object_descriptor(3) {object_descriptor}

(2) sif_header ::=
SIF_VERSION free_text semi_colon -- 4.0
TOOL_NAME free_text semi_colon
TOOL_VERSION free_text semi_colon
TARGET_LANGUAGE free_text semi_colon

Object Descriptor

(3) object_descriptor ::=
module_kind(4) *object*_identifier **IS**
[annotation_list object_kind(5)] -- mandatory for objects or **class**es
-- optional for virtual nodes
-- absent for OP_CONTROLs
annotation_list
[formal_parameters_section(51)] -- only for generics
-- (see rule ODS-2)
[instance_range(57)] -- only for instances
-- (see rule ODS-9)
[actual_parameters_section(52)] -- only for instances
-- (see rule ODS-3)
[description_section(7)]
[implementation_constraints_section(8)]
[provided_interface_section(14)]
[visible_obcs_section(21)] -- only for active objects and **class**es,
-- and virtual nodes (see rule ODS-1/3)
[required_interface_section(15)]
[data_flow_section(10)]
[exception_flow_section(11)]
[internals(19)]
object_trailer(6)

(4) module_kind ::=
OBJECT
| **CLASS**
| **OP_CONTROL**
| **VIRTUAL_NODE**

G.2.9 Free text and target dependent fields

A target language dependent field (td) is used to convey code of a given target language.

Both free text and target language dependent fields are considered as text fields.

A text field starts **with** the begin of text marker (bot), followed by a sequence of graphic characters and an end of text (eot) marker:

```
text ::= bot {graphic_character} eot
bot ::= --|
eot ::= |--
```

If the sequence of graphic characters contains the eot marker, it must be followed by a second eot marker to indicate that it does not end the text.

G.2.10 Annotations

An annotation is either a pragma or some text.

A sequence of annotations are only allowed at the following places:

- at the places explicitly defined by the BNF definition,
- after a semicolon delimiter, but not **with**in a formal part,
- after the reserved words **passive**, **active** and **op_control**.

Additional restrictions may exist for the placement of tool set specific pragmas.

G.2.11 Allowable Content of Sections

The syntactic categories whose names end **with** “section” all follows a general pattern. A section consist of a section title followed by the section content.

The allowed content of a section is one of the following:

- The content specified by the BNF definition.
- The reserved word **none**.
-
-
-
-
-

G.3 Syntax Summary

In all rules, *italic*$^{(n)}$ refers to BNF syntactic category number n.

G.2.4 Numeric Literals

There are two **class**es of numeric literals: real literals and integer literals. A real literal is a numeric literal that **include**s a point; an integer literal is a numeric literal **with**out a point.

integer ::= digit {digit}
real ::= integer [. integer]

G.2.5 String Literals

A string literal is formed by a sequence of graphic characters (possibly none) enclosed between two quotation characters used as string brackets.

string_literal::= “{graphic_character}”

A string literal denotes the sequence of characters enclosed by quotation characters. If a quotation character is to be represented in the sequence of characters, then a pair of adjacent quotation characters must be written at the corresponding place **with**in the string literal.

G.2.6 Comments

A comment starts **with** the begin of comment marker --{ and ends **with** the end of comment marker }--.

comment ::= --{ {graphic_character} }--

Comments may be nested to allow comments to appear **with**in comments.

A comment can appear between any lexical element of a SIF.

G.2.7 Reserved Words

The identifiers listed in appendix B are called reserved words and are reserved for special significance in the language.

Reserved words differing only in the use of upper and lower case letters are considered as the same.

G.2.8 Pragmas

A pragma is used to convey information to the SIF analyser tool.

The pragmas defined by the standard are described in chapter 14: they must be supported by every implementation. In addition, an implementation may provide implementation-defined pragmas.

A pragma that is not language-defined has no effect if its identifier is not recognised by the implementation. Furthermore, a pragma has no effect if its replacement or its arguments do not correspond to what is allowed for the pragma. The region of text over which a pragma has effect depends on the pragma.

- the space character
- special characters:

  " # & ( ) * + , - . / : ; < = > _ | ! $ % ? @ [\ ] ^ ' { } ~
- format effectors: the ISO (and ASCII) characters called horizontal tabulation, vertical tabulation, carriage return, line feed, and form feed
- other special characters (extended ASCII set) as defined by ISO 8859 - 1.

G.2.2 Lexical Elements, Separators, and Delimiters

A lexical element is either a delimiter, a reserved word, an identifier, a numeric literal, a string literal, or a comment.

In some cases an explicit separator is required between adjacent lexical elements (namely, when **with**out separation, interpretation as a single lexical element is possible). A separator is any of a space character, a format effector, or the end of a line. A space character is a separator except **with**in a comment or a string literal. Format effectors other than horizontal tabulation are always separators. Horizontal tabulation is a separator except **with**in a comment. The end of a line is always a separator.

One or more separators are allowed between any two adjacent lexical elements. At least one separator is required between an identifier and a numeric literal, or between two adjacent identifiers or numeric literals.

A delimiter is either one of the following special characters

( ) , . : ;

or one of the following compound delimiters

=> <= <=> :=

Each of the special characters listed for single character delimiters is a single delimiter, except if this character is used as a character of a compound delimiter, or as a character of a comment or string literal.

The other lexical elements are defined in the remainder of this section.

G.2.3 Identifiers

Identifiers are used as names and also as reserved words.

```
identifier ::= letter {[ _ ] letter_or_digit}
letter_or_digit ::= letter | digit
```

All characters of an identifier are significant, including the underscore characters. Identifiers differing only in the use of corresponding upper and lower case letters are considered as the same.

G STANDARD INTERCHANGE FORMAT

G.1 General

The goals of the definition of a Standard Interchange Format (SIF) are twofold:

- to allow interchange of design data between toolsets and platforms,
- to ease the integration of a HOOD toolset **with**in a Software Development Environment.

The Object Description Skeleton (ODS) is a normalised description of HOOD objects, **class**es and virtual nodes which can be mapped into a Standard Interchange Format for transfer between tools and platforms.

The Standard Interchange Format is be based on a **full ASCII** representation specified using BNF (Backus Naur Form). This representation can thus be stored in one or more files containing one or several ODSs according to the **system configuration** definition file and to the choice of the designer.

A HOOD toolset will allow as a minimum to exchange:

- a system configuration,
- a complete design tree,
- one object or **class** (ODS),
- one object and all its descendants.

The syntactic category: object_description from the BNF will be of type informal_text and will be based on the output of the logical activities of the HOOD Design Process.

G.2 Lexical Elements

This section defines the lexical elements of the language.

G.2.1 Character Set

All language constructs may be represented **with** a **graphic character** set which is subdivided as follows:

- upper case letters:

  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
- lower case letters:

  a b c d e f g h i j k l m n o p q r s t u v w x y z
- digits:

  0 1 2 3 4 5 6 7 8 9

F.8 Pragma PROTECTED

This pragma may be added **with**in the ODS code field in Ada of the object toindicate that ROER and RWER constraints should be mapped into entries of a protected record.

to be completed after validation of Ada code generation.

F.9 Pragma IPC

This pragma may be added **with**in the ODS code field of the object to capture additional attributes of a design related to inter process communication, by specifying a sepcific protocol.

to be completed after validation of VN code generation.

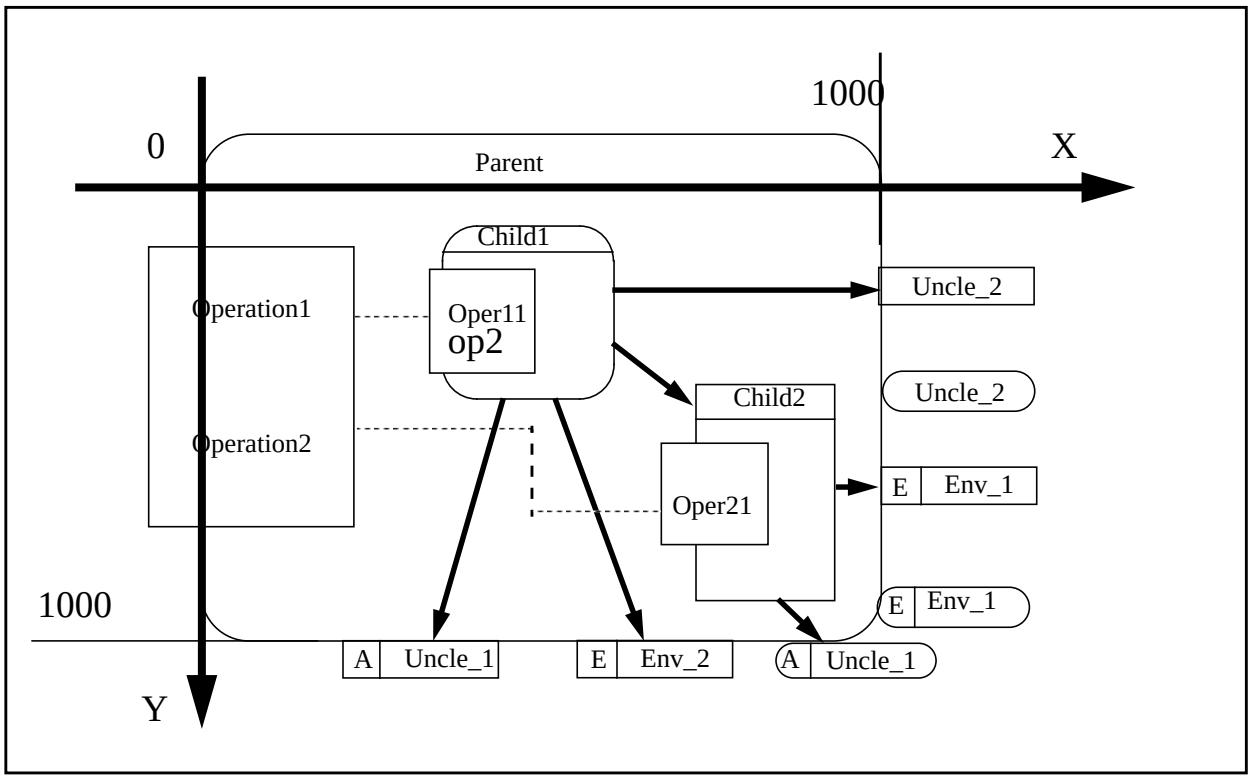
x2_coordinate==> integer, y2_coordinate=> integer,

. . . . .

xn_coordinate==> integer, yn_coordinate=> integer,

where :

- the pragma is placed:
  - at the beginning of the ODS for defining the location of the object, and
  - in the REQUIRED-INTERFACE section to provide the list of apex locations (too aas a minimum) of the links having the current module as target within the parent diagram and this for each of the different graviews.
- *x_coordinate*, *y_coordinate* are integer values expressed in a scale of 1/1000 of the associated values x_range and y _range cprresponding to the full size of the parent object.



F.5 Pragma ODS_ANNOTATION

This pragma may be added **with**in the ODS field of the object to capture additional attributes of a design such as performance, real-time attributes or other evaluation data. These annotations have as goals to allow additional text labels to be attached to ODS fields, in order to process them from SIF for interfacing additional development tools (performance prediction tools, simulators, analysis tools, etc ....)

This pragma shall appear only in the SIF format (see appendix D).

The syntax of this pragma is explained below where :

- <evaluation> stands for : performance, or HRT, or UNDEFINED_YET

PRAGMA ODS_ANNOTATION (*<evaluation >* **=>** text)

F.6 Pragma GRAPHICAL_ANNOTATION

This pragma may be added to the DESCRIPTION field of the object to identify the graphical annotations in the informal text of the DESCRIPTION.
These graphical annotations have as goals to allow additional text labels to be attached to graphical items of the HOOD diagrams, in order to make them more understandable and/or to process them from SIF for interfacing additional development tools (simulators, Message Charts [SDL], event trace diagrams [OMT91], object interaction diagrams [OOSE] drawing tools, auditors, etc ..)

This pragma shall appear only in the SIF format (see appendix D).

The possible syntax of this pragma is explained below

PRAGMA GRAPHICAL_ANNOTATION (*x_coordinate==> integer*
y_coordinate=> integer,)

- where :
- *x_coordinate*, *y_coordinate* are integer values expressed in a scale of 1/1000 of the associated values x_range and y _range corresponding to the full size of the parent object.

F.7 Pragma LOCATION

This pragma may be added some ODS fields of the object to specify the graphical location of the elements of a HOOD diagram in the SIF file.
The syntax of this pragma is explained below

PRAGMA LOCATION (*x1_coordinate==> integer,* y1_coordinate=> integer,

test software for the Object.
The following parameter field are defined :

- **Description**
  This field shall define in informal text the description of the set of test scenarii applied to the object to achieve an incremental integrated testing of the object.
- **Scenario**
  This field shall define the pre and post-conditions as well as the operation sequence associated to a test scenario.The associated code shall be provided(possibly automated by means of **template**s or macros) by the designer for later extraction/generation tool for building test environment data. There may be as many scenario definition as needed.

The syntax of this pragma is given below as

PRAGMA OTS (**Description**=> text
| Scenario => name, PreConditionsCODE=> td_text,
OPSequence=> td_text,
PostConditionsCODE=> td_text,
TestRequiredCODE=> td_text,
TestENVCODE=> td_text
Testresult=> pathname)

where PreConditionsCODE, PostConditionsCODE, TestResult define similar code as for the OP_TEST case, and TestRequiredCODE and TestENVCODE defines the used operations associated to the scenario.

formation fields to support the assembling of test software for the operation.
The following Description fields may be defined :

- **PreCondition**
  This field shall define the pre-conditions settings for performing a unit test upon operation. It **include**s initialization of variables, and/or calls to other operation. The associated code shall be provided (possibly automated by means of **template**s or macros) by the designer for later extraction/generation tool for building the executable code for unit testing.
- **PostCondition**
  This field shall define the post-conditions after performing a unit test upon operation. It **include**s all code to check and use results associated to the test activation of the operation. Especially it may use test oracles. In many cases however this code summarizes in the comparison of value resulted from test **with** expected values. The associated code shall be provided (possibly automated by means of **template**s or macros) by the designer for later extraction/generation tool for building test environment data.
- **TestRequired**
  This field shall specify all operation needed for performing a test execution of the operation under test.
- **CallSequence**
  This field shall specify all operation needed for performing a test execution of the operation under test.
- **TestEnv**
  This field shall specify all objects needed for performing a test execution of the operation under test. This information may be processed later by a test environment tool to provide either the full code of the objects if already tested, or stub code, to simulate the required object behaviour, if not tested or not available.
- **TestResult**
  This field shall specify a pathname for accessing the result of a test.

The syntax of this pragma is given below as

PRAGMA OP_TEST (**OPERATION =>**Operation_signature, **TestName**=>test_id,
DescriptionField=> text ,**CODE**=> td_text)

where **DescriptionField** may take the values PreCondition, PostCondition, TestRequired, TestEnv, TestResult).

F.4.2 Pragma OTS

This pragma OTS (Object Test Strategy) may be added before the end _object field of the ODS in order to a number of information fields to support the assembling of unit

F HOOD PRAGMAS

F.1 Pragma TARGET_LANGUAGE

This pragma may be added at the beginning of the ODS to identify the target language in which the design will be implemented. This pragma may be used by a tool to generate the right code following the relevant translation schemes. By default, the target language is Ada:

PRAGMA Target_language (**name =>** Ada)

The ODS fields which are target language dependent shall be filled according to the target language syntax and rules.

F.2 Pragma NOMUTEX

This pragma may be added within ODS OPCS_HEADER section in order to request suppression of MUTEX code when accessing the states associated to the OSTD . :

PRAGMA NOMUTEX

F.3 Pragma HCS

This pragma may be added to the DESCRIPTION field of the object to identify the associated HOOD chapter **with**in the informal text of the DESCRIPTION.
These HOOD chapters are related to the activities of the HOOD Design Process described in the HOOD User Manual.

PRAGMA HCS (**chapter =>** title_text)

This pragma shall appear only in the SIF format (see appendix D).

F.4 Pragma for TESTING SUPPORT

In order to allow for automation of the unit testing work, and a**void**ing emergence of tool specific syntaxes, two pragmas dedicated for describing test code and standardized interface towards test environments have been defined :

- OP_TEST : Operation Unit testing
- OTS : Object Test Strategy

F.4.1 Pragma OP_TEST

This pragma OP_TEST (OPeration Unit Testing) may be added to the OPERATION CONTROL STRUCTURE field of each operation in order to associate a number of in-

PSEUDO_CODE
RAISED_BY
RASER
RLSER
ROER
RWER
REQUIRED_INTERFACE
RETURN
SERVER
SERVEROBCS
SERVERVNCS
TYPES
TOER,
USED_OPERATIONS
VIRTUAL_NODE
VNCS

E.2 HOOD and Ada Reserved words

IN
IS
OUT
PRAGMA

E RESERVED WORD LIST

E.1 HOOD Reserved words

ACTIVE
ASER
ASER_BY_IT
ATTRIBUTES
CLASS
CLIENT
CLIENTOBCS
CLIENTVNCS
CODE
CONSTANTS
CONSTRAINED_BY
CONSTRAINED_OPERATIONS
DATA
DATAFLOWS
DESCRIPTION
END_MODULE
END_OPERATION
ENVIRONMENT
EXCEPTION_FLOWS
EXCEPTIONS
FORMAL_PARAMETERS
GENERIC
HANDLED_EXCEPTIONS
HSER
HSER_TOER
IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
IMPLEMENTATION_CONSTRAINTS
IMPLEMENTED_BY
INHERITANCE
INTERNALS
INSTANCE_OF
INSTANCE_RANGE
LSER
LSER_TOER
MEMBER_OF
MTEX, MASER,MHSER, MLSER, MRASER,MRLSER,
MTASER,MTHSER, MTLSER, MTRASER,MTRLSER,
NONE
OBJECT
OBJECTS
OBJECT_CONTROL_STRUCTURE
OPERATION
OPERATIONS
OPERATION_CONTROL_STRUCTURES
OPERATION_SETS
OP_CONTROL
OSTD
OSTM
PARAMETERS
PASSIVE
PROPAGATED_EXCEPTION
PROVIDED_INTERFACE

- OSTD Object State Transition Diagram
- OSTM Object State Transition Machine
- PDL Program Design Language
- RASER Reporting Asynchronous Execution Request
- RLSER Reporting Loosely Synchronous Execution Request
- SIF Standard Interchange Format
- STD State Transition Diagram
- TOER Timed Out Execution Request
- VN Virtual Node
- VNT Virtual Node Tree

D ABBREVIATION LIST

- ADT Abstract Data Type
- AM Abstract Machine
- APSE Ada Programming Support Environment
- ASER ASynchronous Execution Request
- ASCII American Standard Code for Interchange of Information
- BNF Backus Naur Form
- CDT **Class** Design Tree
- FSM Finite State Machine
- ER Execution Request
- FIFO First In First Out
- HDL HOOD Design Language
- HCS HOOD Chapter Skeleton
- HDT HOOD Design Tree
- HOOD Hierarchical Object Oriented Design
- HRM HOOD Reference Manual
- HRTS HOOD RUN TIME SUPPORT or Virtual Machine
- HSER Highly Synchronous Execution Request
- HTG HOOD Technical Group
- HUM HOOD User Manual
- HUG HOOD User Group
- HW Hardware
- HWG HOOD Working Group
- ISR Interrupt Service Routine
- LSER Loosely Synchronous Execution Request
- MTEX Mutual Exclusion constraint
- OBCS OBject Control Structure
- ODS Object Description Skeleton
- OOD Object Oriented Design
- OPCS OPeration Control Structure
- OR Observation Report
- OS Operating System

C.25 Passive Object

- An object which has no provided constrained operations or only state constrained operations and which does not have any semantics relatedthread interaction
- An object to which is associated ‘thread empty target code’

C.26 Physical node

- The physical processor or computer on which virtual nodes can be configured.

C.27 Provided Interface

- Part of the ODS that defines the items of the object which are visible from outside.

C.28 Required Interface

- Part of the ODS that defines servers and associated items required for the implementation of that object.

C.29 Root object

- The top level object of an HDT or CDT, and element of the system configuration.

C.30 System Configuration

- The description of the set of root objects and **class**es defining the scope of a design.

C.31 Terminal Object

- An object which is not decomposed into child objects.

C.32 Use relationship

- An object is said to use another object if the former requires one or more of the operations provided by the latter.

C.33 Virtual Node

- A unit of distribution defined by allocation of HOOD objects, and used to define distributed software.

C.17 Internal operation

- An operation that is defined in a terminal object to support the step-wise refinement of an OPCS and implementation of provided operations. Internal operations are not shown in the graphics.

C.18 Non-terminal Object

- An object which is decomposed into child objects.

C.19 Object Description Skeleton - ODS

- The formal textual description of an object.

C.20 Object Control Structure - OBCS

- Part of the ODS of an active object that defines control flow interactions between the thread executing constrained operations.

C.21 Object State Transition Diagram - OSTD

- Part of the ODS of an active or passive object that defines the possible sequence operation execution requests as transitions in a state transition diagram.

C.22 Operation Control Structure - OPCS

- Part of the ODS of a terminal object that defines the control structure / logic of the operation (external or internal) in terms of pseudocode and code.

C.23 Op_Control object

- An object that implements the mapping between one parent operation (constrained / unconstrained) and several child object operation(s) (unconstrained / constrained).

C.24 Operation_Set

- An operation which stands for a list of operations in order to ease representation of long lists of operations.

C.9 Data flow

- Flow of data exchanged between client and server objects.

C.10 Design Process

- Successive break down of a system to design from the root object until terminal objects are reached.

C.11 Environment object

- A view on the provided interface of a root object which is not part of the current HDT or GDT.

C.12 Exception flow

- Flow of exceptions propagated from a server to a client along the USE relationship.

C.13 Formal parameters

- TYPES, CONSTANTS and OPERATIONS parameters of a generic.

C.14 HDT (HOOD Design Tree)

- The hierarchy of objects resulting from a design process applied on a root and consisting of the successive decompositions of parent objects into child objects.

C.15 Include relationship

- Relationship expressing that an object is fully decomposed into a set of child objects that collectively provide the same functionality as the parent.

C.16 Instance

- An object that is instanciated from a generic and customized through actual parameters.
- A Data that represents the instance of a HOOD **class**.

C GLOSSARY

C.1 Active object

- An object which is not passive.

C.2 Class

- An object exporting a type of same name(or of name ‘Instance’ in Ada), and whose provided operations have a receiver parameter of name ‘me’ of type the **class**.

C.3 Abtract Class

- An object exporting a type of same name, and with one of its provided operations having a receiver parameter of name me of type the **class**, specified with the keyword abstract. An abstrtact **class** cannot be instancied.(it can only be inherited)

C.4 Actual parameters

- Parameters allowing to define instances of a generic.

C.5 GDT (GENERIC Design Tree)

- The hierarchy of objects resulting from a design process applied on a Generic (which is a root) and consisting of the successive break down of parent objects into child objects.

C.6 Generic

- An object pattern to represent a reusable object with types,**class**es, data and operations as formal parameters.

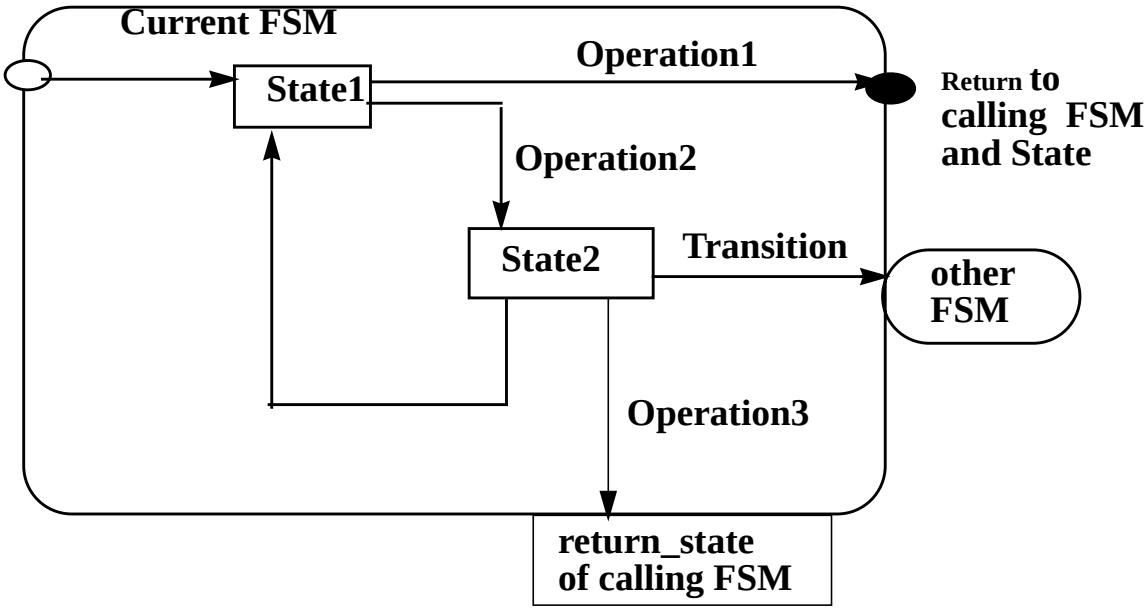
C.7 Constrained operation

- An operation that is constrained in its execution either by the internal state of the object or by a communication protocol.

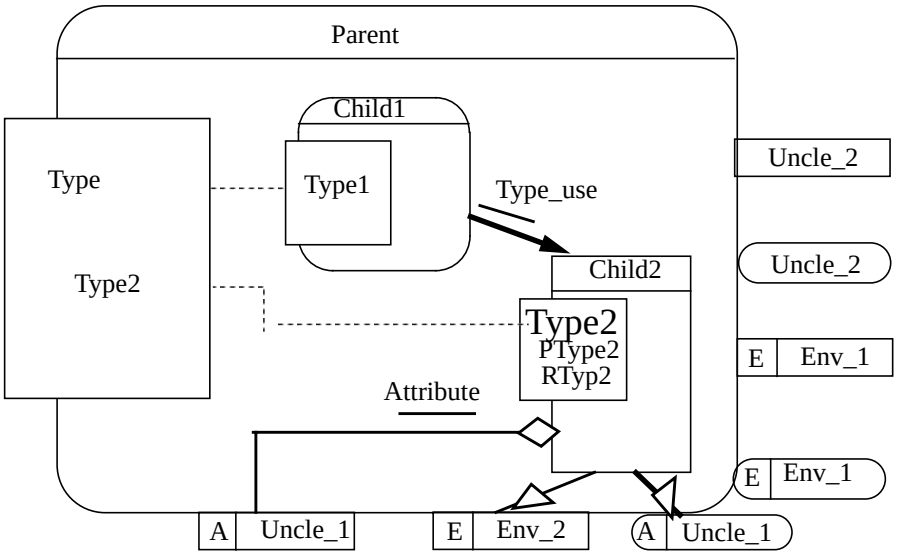
C.8 Control flow

- Direction where the execution of a thread goes when executing an operation of a server. Indicated by the USE relationship.

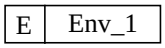
B.20 OSTD graphical formalism



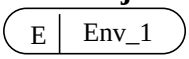
B.19 Include relationship (Structural view)



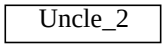
Environment class



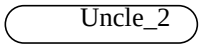
Environment object



Passive uncle class



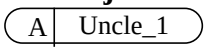
Passive uncle object



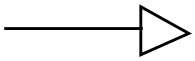
Active uncle class

Implemented_by link

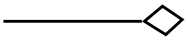
Active uncle object



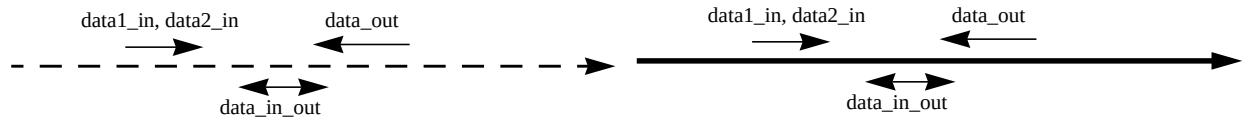
Inheritance link



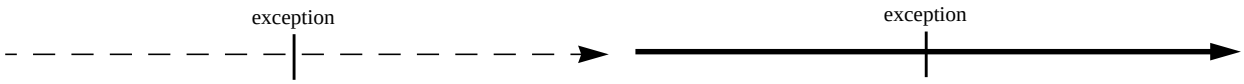
Attribution link



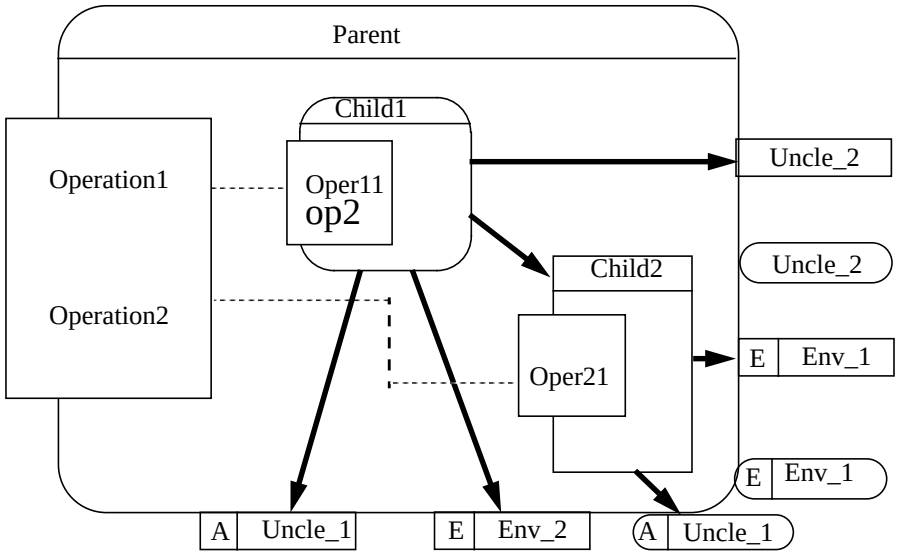
B.16 Dataflow



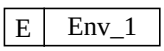
B.17 Exception flow



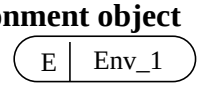
B.18 Include relationship (client_server view)



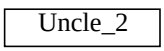
Environment class



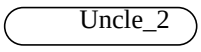
Environment object



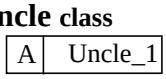
Passive uncle class



Passive uncle object

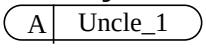


Active uncle class



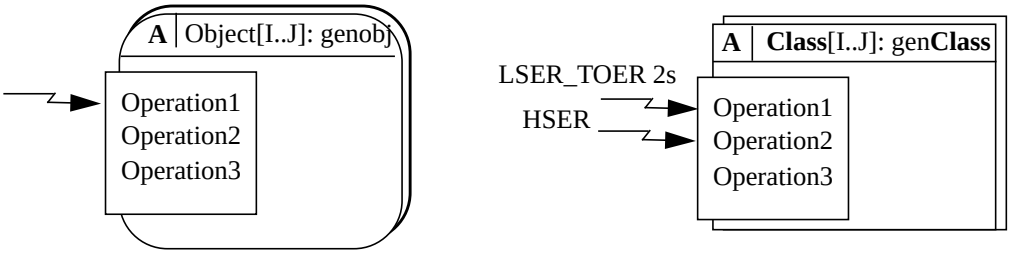
Implemented_by link

Active uncle object

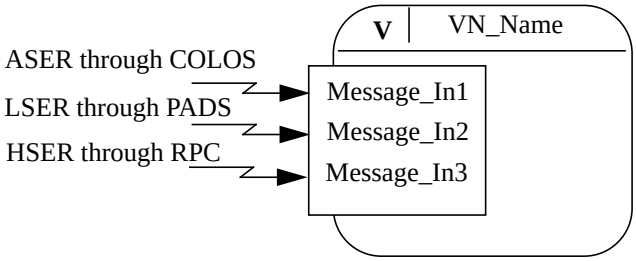


B.12 Multiple instance of passive objects or classes

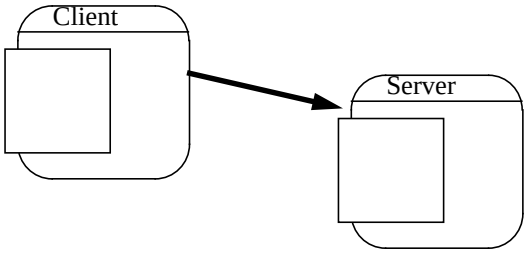
B.13 Multiple instance of active objects or classes



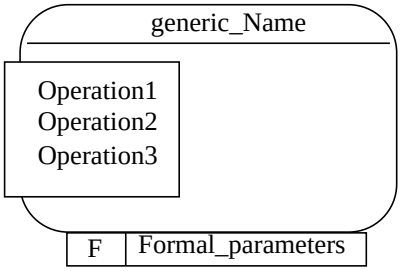
B.14 Virtual node



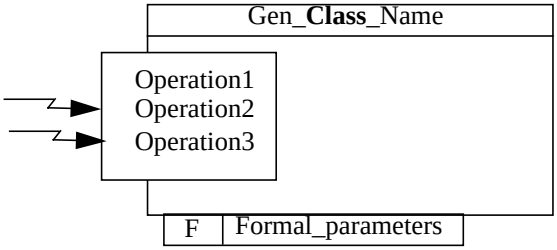
B.15 Use relationship



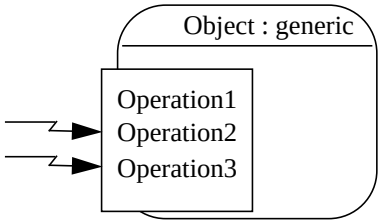
B.7 Generic object



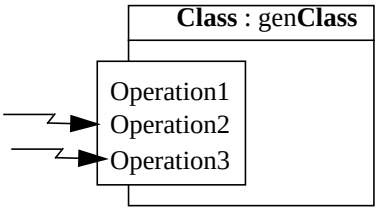
B.8 Generic class



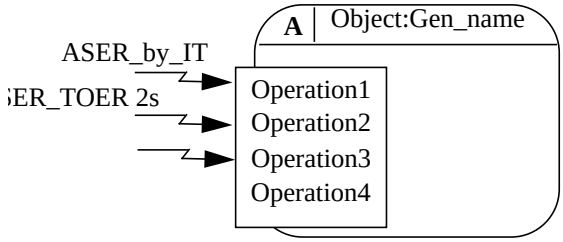
B.9 Passive object instance



B.10 Passive class instance

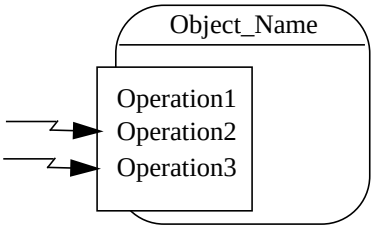


B.11 Active object instance

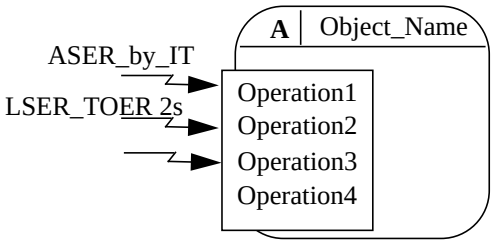


B HOOD GRAPHICAL SYMBOLS

B.1 Passive object

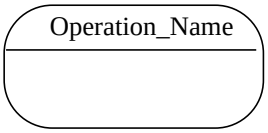


B.2 Active object

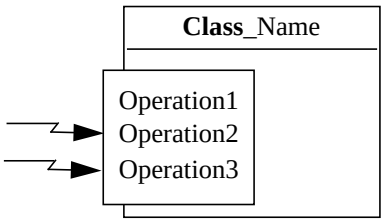


B.3 Constrained operation

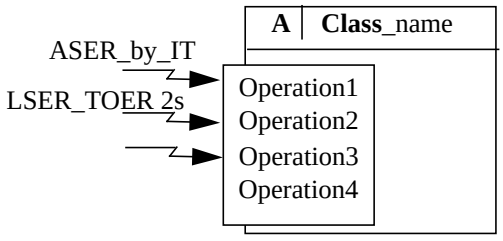
B.4 OP_Control



B.5 Passive class



B.6 Active class



S.VINOSKI, D.SCHMIDT, **Comparing Alternative Client_Side Distributed Programming Techniques,** C++ Report, 1995 May/ June 1995 issues

[WEGNER87]

P.Wegner, **Dimensions of object-based language design,** Proceedings of OOPSLA, 1987

[WELLINGS88]

WELLINGS AJ. **Distributed execution - Units of partitioning**, Proceedings of the International Workshop on Ada Real Time Issues. ACM ADA LETTERS Vol 7, Fall 1988

RUMBAUGH J, BALHA M, PREMERLANI W, EDDY F, LORENSEN W, **Object Oriented Modelling and Design ,** Prentice Hall , 1991

[OMG-CORBA]

OMG group**, The common Object Request Broker : Architecture and Specification,** OMG doc num 91.12.1, 1991

S.Vinoski, **Distributed Object Computing with CORBA** , C++ Report vol 5 July/August 1993

[ODDEL94]

ODDEL J.J.. **Six different kinds of Composition**, Journal of Object-Oriented Programming,Vol 5, N=8

[PARNAS79]

PARNAS D.L. **Designing Software for Ease of Extension and Contraction**, IEEE Transaction on Software Engineering VOL SE-5 N02 March 79

[REI85]

W. REISIG, **Petri nets: an Introduction**, Springer Berlin 198[REI85]

[ROSEN95]

JP ROSEN, **A naming Convention for Classes in Ada95**, Ada Letters, March-April 1995

[SH&M92]

S.SHLAER and S.J.MELLOR, **Object Life-Cycles : modeling the world in States**, Yopurdon Press 1992

[SOUR95]

JL Sourouille, H.Lecoueche, **Integrating State in OO Concurent Model**, proceedings of TOOLS'EUROPE95, Prentice Hall 1995.

[STROU91]

B.STROUSOUP **The Annoted C++ Reference Manual**, Addison-Wesley Press &çç&

[SCHMIDT94]

D.SCHMIDT, **ASX : An Object-Oriented Framework for Developping Distributed Applications**, Proceedings of the 6th USENIX C++ Conference, Cambridge, MA, April 1994

[SCHMIDT5]

D.SCHMIDT,P.STEPHENSON, **Using Design Patterns to Evolve System Software from Unix to WIndows NT** C++ Report, 1995 March 1995

[VINO95]

[GOOD86]

E.SEIDEWITZ and STARK, **General Object Oriented Software Development,** NASA, SEL Series-86-002

[KORSON90].

T.KORSON, J.D. McGREGOR, **Understanding Object Oriented : A Unifying Paradigm,** Communications of the ACM, Sept 1990/ Vol 33/no9

[HAREL87]

D.HAREL, **Statecharts : a visual formalism for complex systems.** Sccience of Computer Programming 8, 1987, pp 231-274t

[HRTOSK]

Hard Real Time Operating System kernel Study (TP 8879) Task 3 development of a design methodology.,ESA/ESTEC report

[HRM3.1]

B.DELATTE, M.HEITZ, JF MULLER /HOOD Technical Group, **HOOD REFERENCE MANUAL3.1,** Masson and Prentice Hall 1993

[HUM89]

WME/89-353/JB **HOOD3.0 USER MANUAL** release 3.0

[HUM96]

HUM/93-12/V3.1 **HOOD3.1 USER MANUAL** (to be available on ESTEC HOOD repository)

[MACH85]

GALINIER M., MATHIS A., **Guide du concepteur MACH**, Thomson CSF, DSE & IGL Technology, 1985

[MEYER88]

MEYER B., **Object Oriented Software Construction,** Prentice Hall 1988

[MOTTET91]

G.Mottet,JC Billaut **Hierachical Object-Oriented Design of a syntactic editor** Technology of Object-Oriented Languages and Systems, Vol4, Prentice Hall, pp335-345, 1991

[MOTTET95]

G.Mottet,A Marpinard, JC Geffroy **Design of Dependable Ada Software,** Prentice Hall 1995

[MULLER89]

MULLENDER S., **Distributed Systems,** ACM Press, Frontier Series, 1989

[OMT91]

APPENDIXES

A REFERENCES

[ADA83]

The Ada Language Reference Manual, AINSI / MIL-STD 1815A (section 1.5)

[ADA]

Ada9x Mapping/Revision Team, **Annoted Draft version 5 of the Programming Language Ada**,Intermetrics, ISO/IECJTC1/SC22 WG9 n193

Ada9x Mapping/Revision Team, **Rationale for the Programming Language Ada**,Intermetrics, ISO/IECJTC1/SC22 WG9 n207

[ATKINSON]

ATKINSON C, MORETON T, NATALI A, **Ada for Distributed Systems**, Ada Companion Series, Cambridge University Press

[BOOCH86]

BOOCH Grady, **Object-Oriented Development**, IEEE Transactions on Software Engineering Vol SE-12, February 86

[BOOCH91]

BOOCH Grady, **Object Oriented Design**, Addison-Wesley 1991

[BURNS90]

A.Burns, A.Wellings, **Real Time Systems and their Programming Languages** Addison-Wesley 1990

[BSSC91]

Board for Software Standardisation and Control (BSSC) ESA SOFTWARE ENGINEERING STANDARDS, PSS-05-0, Issue 2, February 1991

[CCITT89]

CCITT (International Telecommunication Union), **Instruction for SDL Users**, recommendation Z 100- Annex D, Geneva 1989

[COAD91]

P.COAD, E.YOURDON, **Object Oriented Design**, Prentice Hall, 1991

[EXTRA]

EXTRA Working Group,**EXTRA Draft**

[GAMMA94]

E, R.Helm, R.Johnson and J.Vlissides , **Design Patterns : Elements of Reusable Object Oriented Software**, MA : Addison-Wesley