COMPUTER COMMAND AND CONTROL COMPANY

2300 CHESTNUT STREET, SUITE 230 • PHILADELPHIA, PA 19103
215-854-0555 FAX: 215-854-0665

2

APPENDICES TO FINAL REPORT FOR THE SYSTEM ENGINEERING AUTOMATION (SEA) FOR DISTRIBUTED SYSTEMS

CONTRACT NO. N00014-91-C-0183
CDRL SEQUENCE NO. A002

AUGUST 1992

Prepared for:

Department of the Navy
Office of Naval Research
Code 1211
Arlington, VA 22217-5000

LIST OF APPENDICES

FINAL REPORT

APPENDIX A

ADA SPECIFICATIONS FOR REPRESENTING LOGICAL MODEL

SYSTEM ENGINEERING AUTOMATION (SEA)
FOR DISTRIBUTED SYSTEMS

CONTRACT NO. N00014–91–C–0183
CDRL SEQUENCE NO. A002

AUGUST 1992

```
with text_io;
use  Text_io;

package DIS_logical_model is

        —====================================================================—
        —                    DIS DECLARATIONS - LOGICAL MODEL.
        —====================================================================—

        MaxProcesses       : INTEGER := 100;
        MaxConditions      : INTEGER := 100;
        MAX_LENGTH         : INTEGER := 100;
               — Should be set to the desirable number.

        subtype TBD is INTEGER;
               — (To be determined) and assigned appropriate type or value.
               — This is only for compilation purposes.

        subtype diagram_id_type is INTEGER;
               — Is used to identify the flow diagram.

        subtype tool_id_type     is INTEGER;
               — Is used to identify the tool.

        subtype DIS_flow_object_id_type is INTEGER;
               — Is used to identify the flow object.

        subtype DIS_flow_edge_id_type    is INTEGER;
               — Is used to identify the flow edge.

        subtype DIS_transition_id_type  is INTEGER;
               — Is used to identify the transition.

        subtype DIS_state_id_type          is INTEGER;
               — Is used to identify the state.

        subtype DIS_process_id_type       is INTEGER;
               — Is used to identify the process.
```

```
—/*==============================================================*/
—/*                        LOGICAL  MODEL                        */
—/*                                                              */
—/* As stated, the logical model describes the functional and    */
—/* behavioral views of the system. The emphasis within this design */
—/* capture model is on what the system should do as opposed to how */
—/* it should do it.  The logical model contains informations    */
—/* representing functional decomposition of system and the      */
—/* interactions between the decomposed functions of the system  */
—/* through the functional view, and the dynamic operations of the */
—/* decomposed functions at a different time under the different */
—/* situations and conditions through the behavioral view of the */
—/* system.                                                      */
—/*============================================================= */

type DIS_logical_view_type;
type DIS_logical_view_ptr is access DIS_logical_view_type;
   — This is an open declaration for DIS logical view type
   — to make a pointer to the dynamic storage location in memory.

type DIS_functional_view_type;
type DIS_functional_view_ptr is access DIS_functional_view_type;
   — This is an open declaration for DIS functional view type
   — to make a pointer to the dynamic storage location in memory.

type DIS_behavioral_view;
type DIS_behavioral_view_ptr is access DIS_behavioral_view;
   — This is an open declaration for DIS behavioral view type
   — to make a pointer to the dynamic storage location in memory.

type DIS_logical_view_type is
```

```
—/*=============================================================*/
—/*              Declaration  for  the  logical    model             */
—/*—————————————————————————————————————————————————————————————*/

record
   functional_view_list         : DIS_functional_view_ptr;
— Pointer to the functional view.

   behavioral_view_list         : DIS_behavioral_view_ptr;
— Pointer to the behavioral view.

   logical_view_previous        : DIS_logical_view_ptr;
— Pointer to the previous logical view.

   logical_view_next            : DIS_logical_view_ptr;
— Pointer to the next logical view.

end  record;

—/*=============================================================*/
—/*                                                                 */
—/*                        FUNCTIONAL  VIEW                         */
—/*                                                                 */
—/* The functional view encapsulates the information captured when  */
—/* following the conventional structured analysis methodology for  */
—/* systems analysis.  This view describes the system structure in  */
—/* such a way how functions in the system are decomposed and how   */
—/* interact with each other. The representation of this view is    */
—/* graphical and hierachical such that the system engineers can    */
—/* analyze the functional strcuture of the system.                 */
—/* Additionally, this view specifies the non-functional aspects    */
—/* of the system with System Design Factor.                        */
—/*=============================================================*/

type DIS_flow_diagram_type;
type DIS_flow_diagram_ptr is access DIS_flow_diagram_type;
    — This is an open declaration for DIS flow diagram type
    — to make a pointer to the dynamic storage location in memory.

type DIS_functional_view_type is

—/*—————————————————————————————————————————————————————————————*/
—/* Each functional view contains a list of the flow diagram and    */
—/* pointers of the linking relations.                              */
—/*—————————————————————————————————————————————————————————————*/
```

```
record
   flow_diagram_list                 : DIS_flow_diagram_ptr;
   — Pointer to a list of the flow diagram.

   parent_logical_view               :  DIS_logical_view_ptr;
   —  Pointer to a list of parent logical view.

   functional_view_next              : DIS_functional_view_ptr;
   — Pointer to the next available functional view.

   functional_view_previous          : DIS_functional_view_ptr;
   — Pointer to the previous available functional view.

end  record;

type DIS_flow_object_type;
type DIS_flow_object_ptr is access DIS_flow_object_type;
    — This is an open declaration for DIS flow object type
    — to make a pointer to the dynamic storage location in memory.

type DIS_flow_edge_type;
type DIS_flow_edge_ptr is access DIS_flow_edge_type;
    — This is an open declaration for DIS flow edge type
    — to make a pointer to the dynamic storage location in memory.

type DIS_flow_diagram_type is

—/*——————————————————————————————————————————————————————————*/
—/* The flow diagram is a directed graph represented by the    */
—/* flow objects and flow edges, where the flow object implies */
—/* a decomposed function of the system and the flow edges does*/
—/* the interactive relation between the flow object.          */
—/*——————————————————————————————————————————————————————————*/

record
   diagram_id                        : diagram_id_type;
   diagram_name                      : string(1..MAX_LENGTH);
   — Identifier and name of the flow diagram

   diagram_tool_id                   : tool_id_type;
   — Identifier of the tools needed for this flow diagram.

   object_list                       : DIS_flow_object_ptr;
   — Pointer to the list of the flow objects belonging to this
   — flow diagram.

   flow_edge_list                    : DIS_flow_edge_ptr;
   — Pointer to the list of the flow edges belonging to this
   — flow diagram.
```

```
    parent_functional_view              :  DIS_functional_view_ptr;
    —Pointer to the parent functional view.
    previous_flow_diagram               : DIS_flow_diagram_ptr;
    — Pointer to the previous available flow diagram.

    next_flow_diagram                   :  DIS_flow_diagram_ptr;
    — Pointer to the next available flow diagram.
end record;

type DIS_object_type  is

—/*-------------------------------------------------------------------*/
—/*                     Types of the flow object.                     */
—/*-------------------------------------------------------------------*/

    (FunctionalBubble,
    — Functional description

    Table,
    — Internal table

    Database,
    — Database system

    Note,
    — Display/report generation

    HumanOperator,
    — People

    ExternalAgent
    — Evironment
);

type DIS_flow_object_type is

    —/*-------------------------------------------------------------*/
    —/*   A flow object represents a decomposed function of the     */
    —/*   system and contains the flow object informations as follow: */
    —/*                                                             */
    —/*   1. The hierachical, sibling and nesting relations between */
    —/*         flow objects.                                       */
    —/*   2. The nested flow object list including their flow edges. */
    —/*-------------------------------------------------------------*/

record
```

```
    object_id                         : DIS_flow_object_id_type;
    object_name                       : string(1..MAX_LENGTH);
    — Indentifier and name of flow object.

    object_type                       : DIS_object_type;
    — Type of the flow object.

    object_view_level                 : integer;
    — View level specifies how deep it is from the root level.
    — It implies the decomposition level of the system.

    object_flow_edge_list             : DIS_flow_edge_ptr;
    — Pointer to a list of the nested flow edges.

    object_children_list              : DIS_flow_object_ptr;
    — Pointer to a list of the nested flow objects.

    parent_flow_diagram               :  DIS_flow_diagram_ptr;
    —Pointer to the parent flow diagram.

    object_parent                     : DIS_flow_object_ptr;
    — Pointer to the parent flow object.

    next_flow_object                  :  DIS_flow_object_ptr;
    previous_flow_object              :  DIS_flow_object_ptr;
    — Pointers to the next and previous available flow objects.

    object_description                : TBD;
    — Description of the flow object.

    object_design_charactization_list : TBD;
    — List of the flow object characterization.

    object_design_factor_list         : DIS_SDF_Template;
    — List of the flow object design factor.

end record;

type flow_type is

    —/*———————————————————————————————————————————————————————*/
    —/*                   Types of the flow object.           */
    —/*———————————————————————————————————————————————————————*/

    (Control,
    — This edge represents control flow information.

    Data,
    — this edge represents datalflow information.
    Analog
    — this edge represents analog data.
);

type DIS_flow_edge_attributes_type;
type DIS_flow_edge_attributes_ptr is access DIS_flow_edge_attributes_type;
      — This is an open declaration for DIS flow edge attribute type
      — to make a pointer to the dynamic storage location in memory.

type DIS_flow_edge_type is

    —/*—————————————————————————————————————————————————————————*/
    —/*  A flow edge represents the relation between decomposed     */
    —/*  functions of the system or the flow objects in the flow    */
    —/*  diagram and contains the following flow edge informations: */
    —/*                                                              */
    —/*  1. The hierachical, sibling relations between flow edges.  */
    —/*  2. The direction of the flow under the predefined conditions. */
    —/*  3. The additional flow edge information including design   */
    —/*       factor, such as frequency, duration, unit, acuuracy, etc. */
    —/*—————————————————————————————————————————————————————————*/

record
    flow_edge_id                : DIS_flow_edge_id_type;
    flow_edge_name              : string(1..MAX_LENGTH);
    — Identifier and name of the flow edge.

    flow_edge_typ               : flow_type;
    — Type of the flow edge.

    flow_edge_from              : DIS_flow_object_ptr;
    flow_edge_to                : DIS_flow_object_ptr;
    — Pointers to the flow objects from which this flow
    — edeg starts and to which this edge ends.

    flow_edge_condition         : string(1..MAX_LENGTH);
    — Condition of the flow.

    flow_edge_attributes        : DIS_flow_edge_attributes_ptr;
    — Pointer ot the flow edeg attributes.

    flow_edge_design_factor_list:  DIS_SDF_Template_ptr
    —Pointer to a list of system design factor for flow edge.
    parent_flow_diagram         :  DIS_flow_diagram_ptr;
    —Pointer to parent flow_diagram.
    previous_flow_edge          :  DIS_flow_edge_ptr;
    next_flow_edge            :  DIS_flow_edge_ptr;
    —Pointers to the next and previous available flow edges.

end  record;

type DIS_flow_edge_attributes_type is
record
    flow_edge_id              : DIS_flow_edge_id_type;
    flow_edge_name            : string(1..MAX_LENGTH);
    — Identifier and name of the floe edge attributes.

    flow_edge_frequency       : TBD;
    flow_edge_duration        : TBD;
    flow_edge_unit            : TBD;
    flow_edge_range           : TBD;
    flow_edge_increment       : TBD;
    flow_edge_accuracy        : TBD;
    flow_edge_format          : TBD;
    — Attributes of the flow edge and their types will be defined
    — in the later stage of DIS development. These are only
    — compilation purpose.

    parent_flow_edge          :  DIS_flow_edge_ptr;
    —Pointer to the parent flow edge.

end  record;

—/*=============================================================*/
—/*                    BEHAVIORAL  VIEW                          */
—/*                                                              */
—/* The behavioral view describes the dynamic behavior of the    */
—/* system under the controls. This view captures the operations */
—/* of the system at a diffrent time under the different situations */
—/* and conditions. Similar to the functional view of the system, */
—/* this behavioral view represents states of the system and their */
—/* transitions as weli as the process activations in a graphical */
—/* and hierachical way, such that the system engineers can analyze */
—/* the bahavioral construction of real-time informations of the */
—/* system, such as deadline and reconfiguration.                */
—/*=============================================================*/

type DIS_state_transition_diagram;
type DIS_state_transition_diagram_ptr is access DIS_state_transition_diagram;
      — This is an open declaration for DIS state transition type
      — to make a pointer to the dynamic storage location in memory.
```

```
type DIS_process_activation_table;
type DIS_process_activation_table_ptr is access DIS_process_activation_table;
    — This is an open declaration for DIS  process activation table type
    — to make a pointer to the dynamic storage location in memory.

type DIS_behavioral_view is

    —/*——————————————————————————————————————————————————————————*/
    —/*  Each behavioral view contains state transition diagram and     */
    —/*  process activation table.                                      */
    —/*——————————————————————————————————————————————————————————*/

    record
    state_transition_diagram       : DIS_state_transition_diagram_ptr;
    — Pointer to the state transition diagram.

    process_activation_table       : DIS_process_activation_table_ptr;
    — Pointer to the process activation table.

    parent_logical_view            :  DIS_logical_view_ptr;
    —Pointer to parent logical view.

    next_behavioral_view           :  DIS_behavioral_view_ptr;
    previous_behavioral_view       :
    —Pointers to the next and previous behavioral views.

end  record;

type DIS_state_type;
type DIS_state_ptr is access DIS_state_type;
    — This is an open declaration for DIS state type
    — to make a pointer to the dynamic storage location in memory.

type DIS_transition_type;
type DIS_transition_ptr is access DIS_transition_type;
    — This is an open declaration for DIS  transiticn table type
    — to make a pointer to the dynamic storage location in memory.

type DIS_state_transition_diagram is

    —/*——————————————————————————————————————————————————————————*/
    —/*  Each state transition diagram is a directed graph, in          */
    —/*  which nodes represent the states of the system and edges       */
    —/*  represent state transitions under the different situations     */
    --/*  and conditions.                                                */
    —/*——————————————————————————————————————————————————————————*/
```

```
record
    state_list                          : DIS_state_ptr;
    — Pointer to a list of the states.

    transition_list                     : DIS_transition_ptr;
    — Pointer to a list of the transitions.

    next_state_transition_diagram       : DIS_state_transition_diagram_ptr;
    previous_state_transition_diagram   : DIS_state_transition_diagram_ptr;
    — Pointer to the next and previous available state transition
    — table.

    parent_behavioral_view              :  DIS_bevioral_view_ptr;
    —pointer to parent behavioral view.

end record;

type DIS_state_type is

    —/*————————————————————————————————————————————————————————————*/
    —/*                    Type  of  State.                        */
    —/*————————————————————————————————————————————————————————————*/

record
    state_id                            : DIS_state_id_type;
    state_name                          : string(1..MAX_LENGTH);
    — Identifier and name of the state.

    outgoing_edge_list                  : DIS_transition_ptr;
    — Pointer to a list of the transitions which start from
    — this state. These transitions are represented as edges.

    next_state                          : DIS_state_ptr;
    previous_state                      : DIS_state_ptr;
    — Pointers to the next and previous available states.



    parent_state_transition_diagram     :  DIS_state_transition_diagram_ptr;
    —Pointer to parent state transition diagram.

end  record;

type DIS_transition_type is

—/*————————————————————————————————————————————————————————————————*/
—/*                      Type  of  Transition.                     */
—/*————————————————————————————————————————————————————————————————*/
```

```
record
   transition_id                        : DIS_transition_id_type;
   transition_name                      : string(1..MAX_LENGTH);
   — Identifier and name of the transition.

   transition_from_state                : DIS_state_id_type;
   transition_to_state                  : DIS_state_id_type;
   — Pointers to the states from which this transition starts
   — and to which this transition ends.

   transition_enable_condition          : string(1..MAX_LENGTH);
   transition_output_condition          : string(1..MAX_LENGTH);
   — Each transition may be labelled with enabling condition
   — (i.e. input) and output condition, which becomes true
   — as a result of taking this transition.

   next_transition                      : DIS_transition_ptr;
   previous_transition                  : DIS_transition_ptr;
   — Pointers to the next and previous available transitions.

   parent_state                         :  DIS_state_ptr;
   —Pointer to parent state.

   parent_state_transition_diagram      : DIS_state_transition_diagram_ptr;
   — Pointer to parent stransition diagram.

end record;

— Process activation table needs be further elaborated.

type activation_table_type is array(1..MaxProcesses, 1..MaxConditions)
                                                               of INTEGER;
— Activation table type as two dimensional arrays defined by
— the number of the processes and conditions.

type DIS_process_name_id_pair_type;
type DIS_process_name_id_pair_ptr is access DIS_process_name_id_pair_type;
— This is an open declaration for DIS process name id pair type
— to make a pointer to the dynamic storage location in memory.

type DIS_process_activation_table is

—/*—————————————————————————————————————————————————————————————*/
—/*                    Process  Activation  Table.               */
—/*—————————————————————————————————————————————————————————————*/

record
   process_name_id_pair_list            : DIS_process_name_id_pair_ptr;
   — Pointer to a list of process id pair.
```

```
        activation_table                     : activation_table_type;
        — Pointer to the activation table.

        parent_behavioral_view               : DIS_behavioral_view_ptr;
        pointer to parent behavioral view.

    end record;

    type DIS_process_name_id_pair_type is

    —/*————————————————————————————————————————————————————————————————*/
    —/*                    Process name id pair type.                  */
    —/*————————————————————————————————————————————————————————————————*/

    record
        process_name                         : string(1..MAX_LENGTH);
        process_id                           : DIS_process_id_type;
        — Identifier and name of the process name id pair type.

        next_process_name_id_pair            : DIS_process_name_id_pair_ptr;
        previous_process_name_id_pair        : DIS_process_name_id_pair_ptr;
        — Pointers to the next and previous available pais.

        patent_activation_table              : DIS_process_activation_table_ptr;
        —Pointer to parent process activation table.

    end  record;

end DIS_logical_model;
```

FINAL REPORT

APPENDIX B

C++ SPECIFICATIONS FOR REPRESENTING LOGICAL MODEL

SYSTEM ENGINEERING AUTOMATION (SEA)
FOR DISTRIBUTED SYSTEMS

CONTRACT NO. N00014-91-C-0183
CDRL SEQUENCE NO. A002

AUGUST 1992

```
#include <stdio.h>
#include <string.h>
#include "DIS_system_design_factor.c"

/***********************************************************************************************
//                              DIS DECLARATION-LOGICAL MODEL
**********************************************************************************************/

int MaxProcesses   = 100;
int MaxConditions = 100;
int MAX_LENGTH      = 100;
            // Should be set to the desirable number.

typedef int TBD;
           // (To be determined) and assigned appropriate type or value.
           // This is only for compilation purposes.

typedef int diagram_id_type;
           // Is used to identify the flow diagram.

typedef int tool_id_type;
           // Is used to identify the tool.

typedef int DIS_flow_object_id_type;
           // Is used to identify the flow object.

typedef int DIS_flow_edge_id_type;
           // Is used to identify the flow edge.

typedef int DIS_transition_id_type;
           // Is used to identify the transition.

typedef int DIS_state_id_type;
           // Is used to identify the state.

typedef int DIS_process_id_type;
           // Is used to identify the process.

class String { public: String () {}; // constructor
                  private: int len; char *str;
};             //String class for naming structures
```

```
/*=============================================================== */
/*                         LOGICAL  MODEL                         */
/*                                                                */
/*  As stated, the logical model describes the functional and    */
/*  behavioral views of the system. The emphasis within this design */
/*  capture model is on what the system should do as opposed to how */
/*  it should do it.  The logical model contains informations    */
/*  representing functional decomposition of system and the       */
/*  interactions between the decomposed functions of the system   */
/*  through the functional view, and the dynamic operations of the */
/*  decomposed functions at a different time under the different */
/*  situations and conditions through the behavioral view of the */
/*  system.                                                       */
/*=============================================================== */

class DIS_functional_view;
        // This is an open declaration for DIS functional view class
        // to make a pointer to the dynamic storage location in memory.

class DIS_behavioral_view;
        // This is an open declaration for DIS behavioral view class
        // to make a pointer to the dynamic storage location in memory.

class DIS_logical_view {
/*=============================================================== */
/*Declaration  for  the  logical    model                         */
/*--------------------------------------------------------------- */
            DIS_functional_view              *functional_view_list;
                                             // Pointer to the functional view.

            DIS_behavioral_view              *behavioral_view_list;
                                             // Pointer to the behavioral view.

            DIS_logical_view                 *next_logical_view;
            DIS_logical_view                 *previous_logical_view;
                                             // Pointers to the next and
                                             // previous available logical view.

};
```

```
/*==============================================================*/
/*                                                              */
/*                     FUNCTIONAL  VIEW                         */
/*                                                              */
/* The functional view encapsulates the information captured when */
/* following the conventional structured analysis methodology for */
/* systems analysis.  This view describes the system structure in */
/* such a way how functions in the system are decomposed and how  */
/* interact with each other. The representation of this view is   */
/* graphical and hierachical such that the system engineers can   */
/* analyze the functional strcuture of the system.                */
/* Additionally, this view specifies the non-functional aspects   */
/* of the system with System Design Factor.                       */
/*==============================================================*/

class DIS_flow_diagram;
        // This is an open declaration for DIS flow diagram class
        // to make a pointer to the dynamic storage location in memory.

class DIS_functional_view {
        /*______________________________________________________*/
        /* Each functional view contains a list of the flow diagram and */
        /* pointers of the linking relations.                   */
        /*______________________________________________________*/

        DIS_flow_diagram                *flow_diagram_list;
                                        // Pointer to a list of the flow
                                        // diagram.

        DIS_logical_view                *parent_logical_view;
                                        // Pointer to a list of parent
                                        // logical view.

        DIS_functional_view             *next_functional_view;
        DIS_functional_view             *previous_functional_view;
                                        // Pointers to the next and previous
                                        // available functional view.
};

class DIS_flow_object_type;
        // This is an open declaration for DIS flow object class
        // to make a pointer to the dynamic storage location in memory.
```

```
class DIS_flow_edge_type;
        // This is an open declaration for DIS flow edge class
        // to make a pointer to the dynamic storage location in memory.

class DIS_flow_diagram {
        /*------------------------------------------------------------------*/
        /* The flow diagram is a directed graph represented by the          */
        /* flow objects and flow edges, where the flow object implies       */
        /* a decomposed function of the system and the flow edges does      */
        /* the interactive relation between the flow object.                */
        /*------------------------------------------------------------------*/

        diagram_id_type                 diagram_id;
        String                          diagram_name(MAX_LENGTH);
                                        // Identifier and name of the
                                        // flow diagram

        tool_id_type                    diagram_tool_id;
                                        // Identifier of the tools needed
                                        // for this flow diagram.

        DIS_flow_object_type            *object_list;
                                        // Pointer to the list of the flow
                                        // objects belonging to this flow
                                        // diagram.

        DIS_flow_edge_type              *flow_edge_list;
                                        // Pointer to the list of the flow
                                        // edges belonging to this flow
                                        // diagram.

        DIS_functional_view             *parent_functional_view;
                                        // Pointer to the parent
                                        // functional view.

        DIS_flow_diagram                *next_flow_diagram;
        DIS_flow_diagram                *previous_flow_diagram;
                                        // Pointer to the previous
                                        // available flow diagram.

};
```

```
enum DIS_object_type (
        /*-------------------------------------------------------------*/
        /*                  Types of the flow object.                  */
        /*-------------------------------------------------------------*/

        FunctionalBubble,
        // Functional description

        Table,
        // Internal table

        Database,
        // Database system

        Note,
        // Display/report generation

        HumanOperator,
        // People

        ExternalAgent
        // Evironment
);

class DIS_flow_object_type {
        /*-------------------------------------------------------------*/
        /*   A flow object represents a decomposed function of the     */
        /*   system and contains the flow object informations as follow: */
        /*                                                             */
        /*   1. The hierachical, sibling and nesting relations between */
        /*        flow objects.                                        */
        /*   2. The nested flow object list including their flow edges. */
        /*-------------------------------------------------------------*/

        DIS_flow_object_id_type          object_id;
        String                           object_name(MAX_LENGTH);
                                         // Indentifier and name of
                                         // flow object.

        DIS_object_type                  object_type;
                                         // Type of the flow object.

        int                              object_view_level;
                                         // View level specifies how deep
                                         // it is from the root level.
                                         // It implies the decomposition
                                         // level of the system.
        DIS_flow_edge_type              *object_flow_edge_list;
                                        // Pointer to a list of the nested
                                        // flow edges.

        DIS_flow_object_type            *children_object_list;
                                        // Pointer to a list of the nested
                                        // flow objects.

        DIS_flow_diagram                *parent_flow_diagram;
                                        // Pointer to the parent flow
                                        // diagram.

        DIS_flow_object_type            *parent_object;
                                        // Pointer to the parent flow object.

        DIS_flow_object_type            *next_flow_object;
        DIS_flow_object_type            *previous_flow_object;
                                        // Pointers to the next and previous
                                        // available flow objects.

        TBD                             object_description;
                                        // Description of the flow object.

        TBD                             object_design_characterization_list;
                                        // List of the flow object
                                        // characterization.

        DIS_SDF_Template                object_design_factor_list;
                                        // List of the flow object design
                                        // factor.
};

enum flow_type (
        /*------------------------------------------------------------*/
        /*                 Types of the flow object.                  */
        /*------------------------------------------------------------*/

        Control,
        // This edge represents control flow information.

        Data,
        // This edge represents datalflow informatio.

        Analog
        // This edge represents analog data.
);

class DIS_flow_edge_attributes_type;
        // This is an open declaration for DIS flow edge attribute class
        // to make a pointer to the dynamic storage location in memory.

class DIS_flow_edge_type {
        /*-------------------------------------------------------------------*/
        /* A flow edge represents the relation between decomposed            */
        /* functions of the system or the flow objects in the flow           */
        /* diagram and contains the following flow edge informations:        */
        /*                                                                   */
        /* 1. The hierachical, sibling relations between flow edges.         */
        /* 2. The direction of the flow under the predefined conditions.     */
        /* 3. The additional flow edge information including design          */
        /*      factor, such as frequency, duration, unit, acuuracy, etc.    */
        /*-------------------------------------------------------------------*/

        DIS_flow_edge_id_type           flow_edge_id;
        String                          flow_edge_name(MAX_LENGTH);
                                        // Identifier and name of the
                                        // flow edge.

        flow_type                       flow_edge_type;
                                        // Type of the flow edge.

        DIS_flow_object_type            *flow_edge_from;
        DIS_flow_object_type            *flow_edge_to;
                                        // Pointers to the flow objects
                                        // from which this flow edeg
                                        // starts and to which this edge ends.

        String                          flow_edge_condition(MAX_LENGTH);
                                        // Condition of the flow.

        DIS_flow_edge_attributes_type   *flow_edge_attributes;
                                        // Pointer ot the flow edeg
                                        // attributes.

        DIS_SDF_Template                *flow_edge_design_factor_list;
                                        // Pointer to a list of system
                                        // design factor for flow edge.

        DIS_flow_diagram                *parent_flow_diagram;
                                        // Pointer to parent flow diagram.
        DIS_flow_edge_type                      *next_flow_edge;
        DIS_flow_edge_type                      *previous_flow_edge;
                                                // Pointers to the next and
                                                // previous available flow edges.
};

class DIS_flow_edge_attributes_type {
        DIS_flow_edge_id_type                   flow_edge_id;
        String                                  flow_edge_name(MAX_LENGTH);
                                                // Identifier and name of the
                                                // flow edge attributes.

        TBD                                     flow_edge_frequency;
        TBD                                     flow_edge_duration;
        TBD                                     flow_edge_unit;
        TBD                                     flow_edge_range;
        TBD                                     flow_edge_increment;
        TBD                                     flow_edge_accuracy;
        TBD                                     flow_edge_format;
                                                // Attributes of the flow edge
                                                // and their types will be
                                                // defined in the later stage
                                                // of DIS development. These
                                                // are only compilation purpose.

        DIS_flow_edge_type                      *parent_flow_edge;
                                                // Pointer to the parent flow
                                                // edge.
};

/*================================================================*/
/*                      BEHAVIORAL  VIEW                          */
/*                                                                */
/* The behavioral view describes the dynamic behavior of the      */
/* system under the controls. This view captures the operations   */
/* of the system at a diffrent time under the different situations */
/* and conditions. Similar to the functional view of the system,  */
/* this behavioral view represents states of the system and their */
/* transitions as well as the process activations in a graphical  */
/* and hierachical way, such that the system engineers can analyze */
/* the bahavioral construction of real-time informations of the   */
/* system, such as deadline and reconfiguration.                   */
/*================================================================*/
```

```
class DIS_state_transition_diagram;
        // This is an open declaration for DIS state transition class
        // to make a pointer to the dynamic storage location in memory.

class DIS_process_activation_table;
        // This is an open declaration for DIS  process activation table class
        // to make a pointer to the dynamic storage location in memory.

class DIS_behavioral_view {
        /*----------------------------------------------------------------*/
        /*  Each  behavioral  view  contains  state  transition  diagram  and     */
        /*  process  activation  table.                                         */
        /*----------------------------------------------------------------*/

        DIS_state_transition_diagram              *state_transition_diagram;
                                                  // Pointer to the state
                                                  // transition diagram.

        DIS_process_activation_table              *process_activation_table;
                                                  // Pointer to the process
                                                  // activation table.

        DIS_logical_view                          *parent_logical_view;
                                                  // Pointer to parent logical
                                                  // view.

        DIS_behavioral_view                       *next_behavioral_view;
        DIS_behavioral_view                       *previous_behavioral_view;
                                                  // Pointers to the next and previous
                                                  // behaviora; views.
};

class DIS_state_type;
        // This is an open declaration for DIS state class
        //  to make a pointer to the dynamic storage location in memory.

class DIS_transition_type;
        // This is an open declaration for DIS  transition tablei class
        // to make a pointer to the dynamic storage location in memory.
```

```
class DIS_state_transition_diagram {
        /*------------------------------------------------------------*/
        /* Each state transition diagram is a directed graph, in     */
        /* which nodes represent the states of the system and edges  */
        /* represent state transitions under the different situations */
        /* and conditions.                                            */
        /*------------------------------------------------------------*/

        DIS_state_type                   *state_list;
                                         // Pointer to a list of the states.

        DIS_transition_type              *transition_list;
                                         // Pointer to a list of the
                                         // transitions.

        DIS_behavioral_view              *parent_behavioral_view;
                                         // Pointer to parent behavioral
                                         // view.

        DIS_state_transition_diagram     *next_state_transition_diagram;
        DIS_state_transition_diagram     *previous_state_transition_diagram;
                                         // Pointer to the next and previous
                                         // available state transition table.
};

class DIS_state_type {
        /*------------------------------------------------------------*/
        /* Type of State.                                             */
        /*------------------------------------------------------------*/

        DIS_state_id_type                state_id;
        String                           state_name(MAX_LENGTH);
                                         // Identifier and name of the state.


        DIS_transition_type              *outgoing_edge_list;
                                         // Pointer to a list of the
                                         // transitions which start from
                                         // this state. These transitions
                                         // are represented as edges.
        DIS_state_transition_diagram        *parent_state_transition_diagram;
                                            // Pointer to parent state
                                            // transition diagram.

        DIS_state_type                      *next_state;
        DIS_state_type                      *previous_state;
                                            // Pointers to the next and previous
                                            // available states.
};

class DIS_transition_type {
        /*-----------------------------------------------------------------*/
        /*    Type of Transition.                                          */
        /*-----------------------------------------------------------------*/

        DIS_transition_id_type              transition_id;
        String                              transition_name(MAX_LENGTH);
                                            // Identifier and name of the
                                            // transition.

        DIS_state_id_type                   transition_from_state;
        DIS_state_id_type                   transition_to_state;
                                            // Pointers to the states from
                                            // which this transition starts
                                            // and to which this transition ends.

        String                              transition_enable_condition(MAX_LENGH);
        String                              transition_output_condition(MAX_LENGTH);
                                            // Each transition may be labelled
                                            // with enabling condition  (i.e. input)
                                            // and output condition, which becomes
                                            // true as a result of taking this
                                            // transition.

        DIS_state_type                      *parent_state;
                                            // Pointer to the parent state.

        DIS_state_transition_diagram        *parent_state_transition_diagram;
                                            // Pointer to parent transition diagram.

        DIS_transition_type                 *next_transition;
        DIS_transition_type                 *previous_transition;
                                            // Pointers to the next and previous
                                            // available transitions.
};
```

```
        // Process activation table needed to be further elaborated.

typedef int activation_table_type[MaxProcesses][MaxConditions];
        // Activation table type as two dimensional arrays defined by
        // the number of the processes and conditions.

class DIS_process_name_id_pair_type;
        // This is an open declaration for DIS process name id pa` type
        // to make a pointer to the dynamic storage location in memory.

class DIS_process_activation_table {
        /*-------------------------------------------------------------------*/
        /*                    Process  Activation  Table.                     */
        /*-------------------------------------------------------------------*/

        DIS_process_name_id_pair_type       *process_name_id_pair_list;
                                            // Pointer to a list of process id pair.

        activation_table_type               activation_table;
                                            // Pointer to the activation table.

        DIS_behavioral_view                 *parent_behavioral_view;
                                            // Pointer to parent hebavioral view.
};

class DIS_process_name_id_pair_type {
        /*-------------------------------------------------------------------*/
        /*                    Process  name  id  pair  type.                  */
        /*-------------------------------------------------------------------*/

        String                              process_name(MAX_LENGTH);
        DIS_process_id_type                 process_id;
                                            // Identifier and name of the
                                            // process name id pair type.

        DIS_process_activation_table        *parent_activation_table;
                                            // Pointer to parent process
                                            // activation table.
        DIS_process_name_id_pair_type       *next_process_name_id_pair;
        DIS_process_name_id_pair_type       *previous_process_name_id_pair;
                                            // Pointers to the next and
                                            // previous available pair.

};

DIS_logical_model () {
} /*end of DIS_logical_model*/
```

FINAL REPORT

APPENDIX C

ADA SPECIFICATIONS FOR REPRESENTING IMPLEMENTATION MODEL

SYSTEM ENGINEERING AUTOMATION (SEA) FOR DISTRIBUTED SYSTEMS

CONTRACT NO. N00014-91-C-0183
CDRL SEQUENCE NO. A002

AUGUST 1992

```
with Text_io;
use  Text_io;

package DIS_implementation_model is

        —============================================================—
        —        DIS DECLARATIONS - IMPLEMENTATION MODEL.
        —============================================================—

        MAX_LENGTH : INTEGER := 100;
                 — Should be set to the desirable number.

        subtype  TBD                              is INTEGER;
                 — (To be determined) and assigned appropriate type or value.

        subtype  DIS_id_type                      is INTEGER;
                 — Is used for type of indenfication. If it is necessary to
                 — modify indentifier of type, this type can be changed as
                 — to the desired type.

        subtype  DIS_sw_structure_diagram_id      is INTEGER;
                 — Is used to identify the software structure diagram.

        subtype  DIS_sw_module_id                 is INTEGER;
                 — Is used to identify the software/ module.

        subtype  DIS_sw_module_edge_id            is INTEGER;
                 — Is used to identify the software module edge.

        subtype  DIS_sw_task_id                   is INTEGER;
                 — Is used to identify the software module task.

        subtype  DIS_sw_task_edge_id              is INTEGER;
                 — Is used to identify the software task edge.

        subtype  DIS_hw_structure_diagram_id      is INTEGER;
                 — Is used to identify the hardware structrure diagram.

        subtype  DIS_hw_group_node_id             is INTEGER;
                 — Is used to identify the hardware group node.

        subtype  DIS_hw_group_link_id             is INTEGER;
                 — Is used to identify the hardware group link.

        subtype  DIS_hw_node_id                   is INTEGER;
                 — Is used to identify the hadware node.

        subtype  DIS_hw_link_id                   is INTEGER;
                 — Is used to identify the hardware link.
```

```
subtype  DIS_mapping_view_id                    is INTEGER;
         — Is used to identify the mapping view.

subtype  DIS_allocation_tool_id                 is INTEGER;
         — Is used to identify tool id.

subtype  DIS_preference_range                   is INTEGER;
         — Is used to define range of the preference value.

subtype  DIS_data_attribute_id                  is INTEGER;
         — Is used to identify the DIS data attribute.

subtype  DIS_data_size                          is INTEGER;
         — Is used to represent DIS data size.

subtype  DIS_resource_amount                    is INTEGER;
         — Is used to represent DIS resource amount.

subtype  DIS_user_extensible_type               is INTEGER;
         — Is used to represent the type of DIS_user_extensible.
         — This type will be defined in the later stage of
         — the implementation specification development.

subtype  DIS_user_extensible_value              is INTEGER;
         — Is used to represent the value of DIS_user_extensible type.
         — This type will be defined in the later stage of
         — the implementation specification development.

subtype FIELDS is string(1..MAX_LENGTH);
         — Is used to define string type, especially names.


—==========================================================—
—                    IMPLEMENTATION VIEW
—==========================================================—

type DIS_implementation_view ;
type DIS_implementation_view_ptr is access DIS_implementation_view ;
   — This is an open declaration for DIS implemenation view type
   — to make a pointer to the dynamic storage location in memory.

type DIS_sw_structure_diagram;
type DIS_sw_structure_diagram_ptr is access DIS_sw_structure_diagram;
   — This is an open declaration for software structure diagram type
   — to make a pointer to the dynamic storage location in memory.

type DIS_hw_structure_diagram;
type DIS_hw_structure_diagram_ptr is access DIS_hw_structure_diagram;
```

```
                — This is an open declaration for hardware structure diagram type
                — to make a pointer to the dynamic storage location in memory.

        type DIS_mapping_view;
        type DIS_mapping_view_ptr is access DIS_mapping_view;
            — This is an open declaration for mapping view type to make
            — a pointer to the dynamic storage location in memory.

        type DIS_implementation_view   is

—/*=========================================================== */
—/*             Declaration  for  the  implementation  model              */
—/*———————————————————————————————————————————— */


        record

            sw_structure_diagram                     : DIS_sw_structure_diagram_ptr;
            — Pointer to the software structure diagram

            hw_structure_diagram                     : DIS_hw_structure_diagram_ptr;
            — Pointer to the hardware structure diagram

            imp_mapping_list                         : DIS_mapping_view_ptr;
            — Pointer to the mapping view

            implementation_view_next                 : DIS_implementation_view_ptr;
            — Pointer to the next implementation view: successor view

            implementation_view_previous             : DIS_implementation_view_ptr;
            — Pointer to the previous implementation view: predecessor
            — view.

        end record;

            —implementation_view_list                : DIS_implementation_view_ptr;
```

```
—/*===================================================== */
—/*                                                       */
—/*                  SOFTWARE STRUCTURE                   */
—/*                                                       */
—/* Each software structure diagram is represented by a list of    */
—/* modules and a list of edges between modules. Modules can be */
—/* nested and each module includes its own task graph. The Task */
—/* graph cannot be nested since the node of a task graph cannot */
—/* be a module; however, nested relations between tasks can be   */
—/* captured using nested modules. The task represents a           */
—/* computational entity.                                           */
—/*----------------------------------------------------- */

  type DIS_sw_module ;
  type DIS_sw_module_ptr is access DIS_sw_module ;
       — This is an open declaration for software module type to make
       — a pointer to the dynamic storage location in memory.

  type DIS_sw_module_edge ;
  type DIS_sw_module_edge_ptr is access DIS_sw_module_edge ;
       — This is an open declaration for software module edge type to make
       — a pointer to the dynamic storage location in memory.

type DIS_SDF_Template;
type DIS_SDF_Template_ptr is access DIS_SDF_Template ;
       — This is an open declaration for system design factor template
       — type to make a pointer to the dynamic storage location in memory.

type DIS_sw_structure_diagram is

       —/*------------------------------------------------         */
       —/* The software structure diagram is used to reference a     */
       —/* collection of directed graphs, drawn with respect to a    */
       —/* selected methodology, that captures information about a set  */
       —/* of components and their relations along with any hierachical */
       —/* decomposition. For example, a tree of data flow diagrams    */
       —/* may be considered as one type of structure diagram.        */
       —/*------------------------------------------------         */

       record

           sw_structure_diagram_id       : DIS_sw_structure_diagram_id ;
           sw_structure_diagram_name     : FIELDS;
           — Identifier and name of software structure diagram
```

```
        sw_module_list                : DIS_sw_module_ptr;
        — Pointer to the double-linked list of the software modules
        — as children of this software stucture diagram

        sw_module_edge_list           : DIS_sw_module_edge_ptr;
        — Pointer to the double-linked list of the software module
        — edges interconnecting the children modules of this software
        — stucture diagram

        parent_implementation_view:   DIS_implementation_view_ptr;
        — Pointer to the parent implementation view of this software
        — structure diagram.

        next_sw_diagram               : DIS_sw_structure_diagram_ptr;
        — Pointer to the next software structure diagram: successor

        previous_sw_diagram           : DIS_sw_structure_diagram_ptr;
        — Pointer to the previous software structure diagram:
        — predecessor

    end record;

type DIS_sw_task_node;
 type DIS_sw_task_node_ptr is access DIS_sw_task_node;
        — This is an open declaration for software task node type to
        — make a pointer to the dynamic storage location in memory.

type DIS_sw_task_edge;
type DIS_sw_task_edge_ptr is access DIS_sw_task_edge;
        — This is an open declaration for software task edge type to
        — make a pointer to the dynamic storage location in memory.

type DIS_user_extensible;
type DIS_user_extensible_ptr is access DIS_user_extensible;
        — This is an open declaration for user extensible variable
        — type to make a pointer to the dynamic storage location in
        — memory.


type DIS_sw_module   is
```

```
—/*————————————————————————————————————————————————————————————*/
—/* A software module class contains the following information: */
—/*                                                             */
—/* 1. The hierachical, sibling and nesting relations between   */
—/*      modules.                                               */
—/ *2. The identity of task graphs that belong to the module.   */
—/*                                                             */
—/* In addition, there are two special kind of edges (called    */
—/* entry_super_edge and exit_super_edge). They are used to     */
—/* identify the entry and exit points of the task graph at     */
—/* the module level.                                           */
—/*————————————————————————————————————————————————————————————*/

record

      module_id                 : DIS_sw_module_id ;
      module_name               : FIELDS;
      — Identifier and name of software module

      parent_sw_structure       : DIS_sw_structure_diagram_ptr;
      — Pointer to parent software structure diagram

      parent_module             : DIS_sw_module_ptr;
      — Pointer to the parent software module if any

      next_module               : DIS_sw_module_ptr;
      previous_module           : DIS_sw_module_ptr;
      — Pointer to previous/next software modules as
      — successor/predecesso

      submodule_list            : DIS_sw_module_ptr;
      — Pointer to the list of the children submodules
      — define links between super edges of the submodules

      module_edge_list          : DIS_sw_module_edge_ptr;
      — Pointer to the list of software module edge defining links
      — between super edges of the submodules

      task_node_lis             : DIS_sw_task_node_ptr;
      task_edge_list            : DIS_sw_task_edge_ptr;
      — Pointers to the lists of the software task nodes belongs
      — to this module and of software task edges defining links
      — between tasks: that is, the task graph. A task graph is a
      — directed graph: each node denotes a schedulable
      — computational entity and an edge represents a precedence
      — relation between two nodes.
```

```
        entry_super_edge_lis          : DIS_sw_task_edge_ptr;
        exit_super_edge_list          : DIS_sw_task_edge_ptr;
        — Two special kinds of edges, called enter_super_edge &
        — exit_super_edge, are pointers to the lists of software
        — task edges that are to be visible outside the current module.
        — A super edge to an entering to or from exiting node of
        — task_graph. Each entry_super_edge and exit_super_edge are
        — either a task edge or a entry_super_edge/exit_super_edge
        — of a submodule.

        module_sdf                    : DIS_SDF_Template_ptr;
        — Pointer to System Design Factor Template for this module

        user_extensible_var           : DIS_user_extensible_ptr;
        — Pointer to the DIS_user_extensible type

end record;

type DIS_sw_module_edge  is

—/*————————————————————————————————————————————————————————————*/
—/* The software module edge represents the link between       */
—/* modules, and supports the hierachical orders/relations of  */
—/* the software module organization as well as the list of    */
—/* super edges belonging to this software module edge.        */
—/*————————————————————————————————————————————————————————————*/

    record

        module_edge_id                : DIS_sw_module_edge_id ;
        module_edge_name              : FIELDS;
        — Identifier and name of the software module edge

        attributes                    : FIELDS;
        — Attribute of the software module edge

        parent_sw_structure_diagram   : DIS_sw_structure_diagram;
        — Pointer to the parent software structure diagram

        from_module                   : DIS_sw_module_ptr;
        to_module                     : DIS_sw_module_ptr;
        — Source and destination pointer for the software module edge

        super_edge_list               : DIS_sw_task_edge_ptr;
        — Point to the list of the super edges belonging to this
        — software module edge.
```

```
          next_module_edge            : DIS_sw_module_edge_ptr;
          previous_module_edge        : DIS_sw_module_edge_ptr;
          — Pointers to the next/previous module edges:
          — successor/predecessor

          module_edge_sdf             : DIS_SDF_Template_ptr;
          — Pointer to the system design factor template for software
          — module edge

          user_extensible_var         : DIS_user_extensible_ptr;
          — Pointer to the DIS_user_extensible type

 end record;

type DIS_data_attribute ;
 type DIS_data_attribute_ptr is access DIS_data_attribute ;
          — This is an open declaration for software data attribute type
          — to make a pointer to the dynamic storage location in memory.

 type DIS_resource ;
type DIS_resource_ptr is access DIS_resource ;
          — This is an open declaration for software resource type to
          — make a pointer to the dynamic storage location in memory.

type t_type  is (Relative,Absolute);

          —/*——————————————————————————————————————————————————————————*/
          —/* This specfies the type of DIS time, such that Absolute */
          —/* represent the clock time while Relative represents relative */
          —/* time length from some events.                            */
          —/*——————————————————————————————————————————————————————————*/

type DIS_time_type  is

          —/*——————————————————————————————————————————————————————————*/
          —/* Specifies the type of DIS time and its value              */
          —/*——————————————————————————————————————————————————————————*/

      record
          time_kind                   : t_type ;
          time_value                  : integer;
 end  record;

type DIS_log_operators is (log_and,log_or);
```

```
    —/*——————————————————————————————————————————————————————————*/
    —/*  This is a flag specifying the conditions for executing of      */
    —/*  a task: whether all conditions (or output) data are needed     */
    —/*  (or generated) by the certain task.                            */
    —/*——————————————————————————————————————————————————————————*/

type DIS_sw_task_node is

    —/*——————————————————————————————————————————————————————————*/
    —/*  The software task node class specifies DIS_sw_task_node        */
    —/*  structure. There is an input list to identify input data       */
    —/*  and an output list to identify output data generated by        */
    —/*  the task. In addition, predecessor list identifies tasks       */
    —/*  that execute before the task and successor list identifies     */
    —/*  task that execute after the task. There is an and/or flag      */
    —/*  associated with the above four task lists that specifies       */
    —/*  whether all input (or output) data are needed ( or             */
    —/*  generated) by the task. This information is required by        */
    —/*  some optimization algorithms. Each task may include timing     */
    —/*  information such as ready time, deadline and duration.         */
    —/*  In addiion, it identifies resources it needs. For resource     */
    —/*  needs, resource type identifies the resource a task needs      */
    —/*  and amount it needs.                                           */
    —/*——————————————————————————————————————————————————————————*/

    record
        task_id                     : DIS_sw_task_id ;
        task_name                   : FIELDS;
        — Identifier and name of software task node

        parent_module                           : DIS_sw_module;
        — Pointer to the parent software module

        task_structure              : TBD;
        task_description            : TBD;
        — Structure ans description of the task node will be determined
        — in later stage of development.

        task_edge_list              : DIS_sw_task_edge_ptr;
        — Task_edge's from or to this task_node

        task_input_and_or           : DIS_log_operators;
        task_input_list             : DIS_data_attribute_ptr;
        task_output_and_or          : DIS_log_operators;
```

```
          task_output_list                        : DIS_data_attribute_ptr;
          — Data dependencies

          task_before_and_or                      : DIS_log_operators;
          task_before_list                        : DIS_sw_task_node_ptr;
          task_after_and_or                       : DIS_log_operators;
          task_after_list                         : DIS_sw_task_node_ptr;
          — Task precedence relations

          task_ready_time                         : DIS_time_type ;
          task_deadline                           : DIS_time_type ;
          task_period                             : DIS_time_type ;
          — Timing information

          task_resource_needs                     : DIS_resource_ptr;
          — Resource needs

          task_buddy_task                         : DIS_sw_task_node_ptr;
          — The cooperating tasks

          task_max_replication                    : integer;
          task_importance                         : integer;
          task_execution_probability              : TBD;
          task_communication_delay_matrix         : TBD;
          — These fields will be defined in later stage.

          error_cumulation                        : integer;
          imprecise_error_convergence             : integer;
          — Univ. Illinois imprecise computation support.

          next_task                               : DIS_sw_task_node_ptr;
          previous_task                           : DIS_sw_task_node_ptr;
          — Pointer to the next/previous task edges

          task_sdf                                : DIS_SDF_Template_ptr;
          — Pointer to the Task Design Factor template

          user_extensible_var                     : DIS_user_extensible_ptr;
          — Pointer to the DIS_user_extensible type

end record;

type direction is (no_way, one_way, two_way);
```

```
    —/*————————————————————————————————————————————————————————*/
    —/* The direction of data flows in the task edge           */
    —/*————————————————————————————————————————————————————————*/

type DIS_sw_task_edge is

    —/*————————————————————————————————————————————————————————*/
    —/* The software task edge specifies the relations between */
    —/* software task nodes and software module nodes. For each */
    —/* task edge, task_data_edge identifies the data associated */
    —/* with the edge along with the duration of availability of */
    —/* the data. In addition, from_task_node and to_task_node  */
    —/* specifies the source and destination of the edge.       */
    —/*————————————————————————————————————————————————————————*/

record

task_edge_id                        : DIS_sw_task_edge_id;
    task_edge_name                  : FIELDS;
    — Identifier and name of the software task edge

    parent_module                   : DIS_sw_module_ptr;
    parent_module_edge              : DIS_sw_module_edge_ptr;
    — Pointer to ther parent software module and module edge

    task_edge_data                  : DIS_data_attribute_ptr;
    — Pointer to the data attributes associated to thisdge

    from_task_node                  : DIS_sw_task_node_ptr;
    to_task_node                    : DIS_sw_task_node_ptr;
    — Pointer to the source and destination software task node.

    flow_direction                  : direction;
    — Direction of the data flow in the task edge.

        next_task_edge              : DIS_sw_task_edge_ptr;
        previous_task_edge          : DIS_sw_task_edge_ptr;
        — Pointer to the next/previous task node in task edge list
        — where this edge belongs to.

        task_edge_sdf               : DIS_SDF_Template_ptr;
        — Pointer to the system design factor template.

        user_extensible_var         : DIS_user_extensible_ptr;
        — Pointer to the DIS_user_extensible type

end record;
```

```
type d   is (Msg, SharedMemory);

    —/*——————————————————————————————————————————————————————*/
    —/* The type of data based on the paradigm of message-passing */
    —/* or shared memory.                                         */
    —/*——————————————————————————————————————————————————————*/

type DIS_data_attribute is

    —/*——————————————————————————————————————————————————————*/
    —/* The data attribute specifies the type and size of data    */
    —/* being communicated through edges between tasks. It points */
    —/* to the list of sender/receiver tasks specified by the list */
    —/* of their respective edges between them. In addtion, it     */
    —/* lists the resource needed for this data attribute as well  */
    —/* as the timing constraint of data-deadline and data         */
    —/* frequency (to be defined in the later stage of the         */
    —/* development.                                               */
    —/*——————————————————————————————————————————————————————*/

record

    data_attribute_id              : DIS_id_type ;
    data_attribute_name            : string(1..MAX_LENGTH);
    — Identifier and name of the data attributes

    data_kind                      : d ;
    data_size                      : integer;
    — The kind and size of the data in this data attribuets.

    task_edge_list                 : DIS_sw_task_edge_ptr;
    — The list of the software task edges through which data
    — being transmitted.

    sender_kind                    : DIS_log_operators;
    data_sender_list               : DIS_sw_task_node_ptr;
    receiver_kind                  : DIS_log_operators;
    data_receiver_list             : DIS_sw_task_node_ptr;
    — Kind of log-operation ("and" or "or") for senders and
        — receivers, and lists of senders and receivers.

    data_resource_need_list        : DIS_resource_ptr;
    — The list of the resources needed for this data attribute

    data_frequency                 : TBD;
    data_eadline                   : DIS_time_type ;
    — The timing constraint of data deadline and data frequency to
    — be determined in the later stage of development.

    user_extensible_var                : DIS_user_extensible_ptr;
    — Pointer to the DIS_user_extensible type

end record;

type  DIS_resource_type       is (CPU, Memory, IO, Communication);

—/*——————————————————————————————————————————————————————————————*/
—/*                         Kind  of  resource                   */
—/*——————————————————————————————————————————————————————————————*/

type  DIS_resource_u          is (KIPS, MIPS, Bytes, KBytes,
                              MBytes, Sec, Millisec, MicroSec);

—/*——————————————————————————————————————————————————————————————*/
—/*  Unit  time  of  each  resource  type:                       */
—/*      KIPS,  MIPS,  or  spec  marks  for  CPU  specification, */
—/*      Bytes,  KBytes,  MBytes  for  memory,                   */
—/*      (Bytes,  KBytes,  MBytes)  per  (Sec,  MilliSec,  MicroSec) */
—/*      for  IO  and  communication.                            */
—/*——————————————————————————————————————————————————————————————*/

type DIS_resource_unit is

—/*——————————————————————————————————————————————————————————————*/
—/                *Resource  unit  and  its  amount              */
—/*——————————————————————————————————————————————————————————————*/

record
    Resource_unit                      : DIS_resource_u ;
    resource_amount                    : integer;
end record;

type DIS_hw_node ;
type DIS_hw_node_ptr is access DIS_hw_node ;
     — This is an open declaration for hardware node type to make a
     — pointer to the dynamic storage location in memory.

type DIS_resource   is
```

```
—/*————————————————————————————————————————————————————————————————*/
—/* Resource specifies its size and units and pointers to the      */
—/* related nodes, edges, and data attributes. It also contains    */
—/* pointers to the hardware node where it can be defined by a     */
—/* hardware configuration.                                        */
—/*————————————————————————————————————————————————————————————————*/


    record
        resource_id                 : DIS_id_type;
        resource_name               : FIELDS;
        — Resource type and unit

        resource_kind               : DIS_resource_type;
        resource_units              : DIS_resource_unit;
        task_node_list              : DIS_sw_task_node_ptr;
        task_edge_list              : DIS_sw_task_edge_ptr;
        data_attribute_list         : DIS_data_attribute_ptr;
        — Pointer to the taske node, edge, and data attributes
        — related to this resource or that need this resource.

        hw_node_list                : DIS_hw_node_ptr;
        — Hardware node being extracted from the hardware structure

        next_resource_need          : DIS_resource_ptr;
        previous_resource_need      : DIS_resource_ptr;
        — Pointer to the next/previous resource available or required
        — in the list.

        user_extensible_var         : DIS_user_extensible_ptr;
        — Pointer to the DIS_user_extensible type

end record;
```

```
—/*=================================================== */
—/*                 HARDWARE STRUCTURE                */
—/*                                                   */
—/* A hardware structure diagram defines a hardware configuration. */
—/* Each hardware structure diagram is represented by a   */
—/* list of group nodes and a list of group links with their */
—/* communication topology bewteen group nodes. Group nodes can */
—/* be nested and each group node includes its own hardware node */
—/* graph. Unlike task graph in software structure, hardware node */
—/* graph can be nested. Our view is that the hardware   */
—/* node represents a hardware component in a computer architecture, */
—/* such as a processor, CPU, memory, IO, etc.           */
—/*___________________________________________________ */

type DIS_hw_group_link ;
type DIS_hw_group_link_ptr is access DIS_hw_group_link ;
— This is an open declaration for hardware group node type to make a
— pointer to the dynamic storage location in memory.

type DIS_hw_group_node ;
type DIS_hw_group_node_ptr is access DIS_hw_group_node ;
— This is an open declaration for hardware group link class to make a
— pointer to the dynamic storage location in memory.

type DIS_hw_group_link_topology is (

—/*___________________________________________________ */
—/* The various ways of physically connecting hardware group */
—/* nodes with communications. Here are the generally known */
—/* types of communication topology existing today.      */
—/*___________________________________________________ */

        fully_connected,
        — All group nodes are directed linked with all other
        — group nodes.

        partially_connected,
        — Some group nodes are directly linked with some other
        — groups nodes, but not all.

        hierachical,
        — Group nodes are organied or linked as s tree.

        star,
        — One of the group nodes is connected to all other group
        — nodes. Node of the other nodes are connected to each other.
```

```
          ring,
          — Each group node is phisically connected to exactly two other
          — group nodes.

          multi_access_bus);
          — There is a single shared hardware group links. All group
          — node in the system are directly connected to that group link.

type DIS_hw_structure_diagram is

          —/*——————————————————————————————————————————————————————*/
          —/* Each view of DIS_hardware_structure consists of         */
          —/*     - a list of group node,                             */
          —/*     - a list of group edges and communication           */
          —/*         topology between group node.                    */
          —/* Similar to the software module in the hierachical view, */
          —/* group nodes can be nested recuesively. Each group node  */
          —/* may include its own hardware node graph with its        */
          —/* specific internal communication topology. Different    */
          —/* from software task node in hierachical perspective of   */
          —/* configuration, the hardware node can be nested          */
          —/* recursively.                                            */
          —/*——————————————————————————————————————————————————————*/

     record
          hw_structure_diagram_id          : DIS_hw_structure_diagram_id;
          hw_structure_diagram_name        : FIELDS;
          — Identifier and name of hardware structure diagram

          paretnt_implementation_view      : DIS_implementation_view_ptr;
          — Pointer to parent implementation model

          hw_group_node_list               : DIS_hw_group_node_ptr;
          — Pointer to the double-linked list of hardware group nodes
          — belonging to this hardware structure diagram

          hw_group_link_list               : DIS_hw_group_link_ptr;
          — Pointer to the double-linked list of hardware group links
          — inter connecting hardware group nodes of this hardware
          — structure diagram

          hw_group_link_topology           : DIS_hw_group_link_topology;
          — Communication topology of hardware group nodes

          next_hw_diagram                  : DIS_hw_structure_diagram_ptr;
          previous_hw_diagram              : DIS_hw_structure_diagram_ptr;
          — Pointers to the next/previous hardware structure diagram:
          — successor/predecessor

end record;

type DIS_hw_link ;
type DIS_hw_link_ptr is access DIS_hw_link ;
          — This is an open declaration for hardware link type to make a
          — pointer to the dynamic storage location in memory.

subtype DIS_hw_link_topology is DIS_hw_group_link_topology;
          — Internal hardware link topology

type DIS_hw_group_node is

          —/*————————————————————————————————————————————————— */
          —/* A hardware_group_node class contains the following   */
          —/* informations:                                        */
          —/*     1. The hierachical, sibling and nesting relations */
          —/*           between hardware group nodes               */
          —/*     2. The identity of hardware node graphs          */
          —/*           that belong tothis hardware group node.    */
          —/* For both graph, the communication topology           */
          —/* can be specified. In addition, there are two special kinds */
          —/* of links (called entry_super_link and exit_super_link). They */
          —/* are used to identify the entry and exit points of the node */
          —/* graph at the group node level.                       */
          —/*————————————————————————————————————————————————— */

     record

          hw_group_node_id              : DIS_hw_group_node_id;
          hw_group_node_name            : FIELDS;
          — Identifier and name of hardware group node

          parent_hw_structure           : DIS_hw_structure_diagram_ptr;
          — Pointer to parent hardware structure diagram

          parent_hw_group_node          : DIS_hw_group_node_ptr;
          — Pointer to parent fardware group node

          next_hw_group_node            :  DIS_hw_group_node_ptr;
          precious_hw_group_node        :  DIS_hw_group_node_ptr;
          — Pointer to next/previous hardware group node:
          — successor/predecessor

          sub_hw_group_node             :  DIS_hw_group_node_ptr;
          — Pointer to list of sub-group nodes as children of this
          — group node
```

```
            sub_hw_group_link                  :  DIS_hw_group_link_ptr;
            hw_group_link_topology             : DIS_hw_group_link_topology;
               Pointer to list of hardware group links defining physical
            — data communication link between subgroup nodes and their
            — topology

            — NODE GRAPHS belongs to this group node

            hw_node_list                       : DIS_hw_node_ptr;
            hw_link_list                       : DIS_hw_link_ptr;
            hw_node_link_topology              : DIS_hw_link_topology;
            — Hardware node graph belongs to this hardware group node:
            — nodes, links, and their link topology

            entry_super_link_list              : DIS_hw_link_ptr;
            exit_super_link_list               : DIS_hw_link_ptr;
            — There are two special kinds of links, called
            — enter_super_link and exit_super_link, that are to be
            — visible outside the current group. A super link to an
            — entering or from exiting node of the group_node.
            — Each entry_super_link is either a hw_link or a
            — entry_super_link of a sub group node. Each exit_super_link
            — is either a hw_link or a enter_super_link of a
            — sub_group_node.

            group_node_sdf                     : DIS_SDF_Template_ptr;
            — Pointer to system design factor template

            user_extensible_var                : DIS_user_extensible_ptr;
            — Pointer to the DIS_user_extensible type

end record;

type DIS_hw_group_link is

            —/*———————————————————————————————————————————————————————*/
            —/* The hardware group link represents the physical        */
            —/* communication between hardware group nodes, and support */
            —/* the hierachical orders/relations of the hardware group */
            —/* organization with its respective topology. It also points */
            —/* to the list of the super links belonging to this       */
            —/* hardware group link.                                   */
            —/*———————————————————————————————————————————————————————*/

      record
```

```
          hw_group_link_id              : DIS_hw_group_link_id;
          hw_group_link_name            : FIELDS;
          — Identifier and name of hardware group link

          parent_hw_structure           : DIS_hw_structure_diagram_ptr;
          — Pointer to parent hardware structure diagram

          from_hw_group_node            : DIS_hw_group_node_ptr;
          to_hw_group_node              : DIS_hw_group_node_ptr;
          — Pointer to the source/destination of this hardware group link

          super_link_list               : DIS_hw_link_ptr;
          — Pointer to list of super links belonging to this group node

          next_hw_group_link            : DIS_hw_group_link_ptr;
          previous_hw_group_link        : DIS_hw_group_link_ptr;
          — Pointers to nest/previous hardware group links:
          — successor/predecessor

          group_link_sdf                : DIS_SDF_Template_ptr;
          — Pointer to system design factor template for hardware
          — group link

          user_extensible_var           : DIS_user_extensible_ptr;
          — Pointer to the DIS_user_extensible type

    end record;

   type DIS_hw_link_g  is (Bus, LAN);

          —/*—————————————————————————————————————————————————*/
          —/* The genetic type of the hardware link           */
          —/*—————————————————————————————————————————————————*/

   type DIS_hw_link_spec   is (NotKnown,Ethernet,TokenRing);

          —/*—————————————————————————————————————————————————*/
          —/* The specification of the hardware link          */
          —/*—————————————————————————————————————————————————*/

   type DIS_hw_node_g  is (Processor, CPU, Memory, IOchannel, Other);

          —/*—————————————————————————————————————————————————*/
          —/* The genetic type of the hardware node           */
          —/*—————————————————————————————————————————————————*/

   type DIS_hw_node_spec   is (NotKnown,Sun,RISC,Sparc);
```

```
        —/*—————————————————————————————————————————————————————— */
        —/* The specification of the hardware node                 */
        —/*—————————————————————————————————————————————————————— */

type DIS_hw_node  is

        —/*—————————————————————————————————————————————————————— */
        —/* A hardware node graph is a directed graph: each node   */
        —/* denotes an actual hardware component in computer        */
        —/* architecture as stated and link represents the physical */
        —/* communication line or bus between hardware components.  */
        —/* Each hardware node identifies its type, specification   */
        —/* and available resources. Unlike task graph in software  */
        —/* structure, it can be nested recursively. Super links in */
        —/* this level indicate hardware link to hardware node in   */
        —/* the different hardware nodes or group nodes.            */
        —/*—————————————————————————————————————————————————————— */

    record

        hw_node_id                      : DIS_hw_node_id ;
        hw_node_name                    : FIELDS;
        — Identifier and name of hardware node

        hw_node_generic_kind            : DIS_hw_node_g ;
        — Node specific identifies which known hardware component
        — it is. It serves a key to database containing known
        — hardware information.

        hw_node_specific                : DIS_hw_node_spec;
        — What resources are provided by this hw_node.
        — resource_available  and resource_need  should be merged.

        hw_node_resource_available      : DIS_resource_ptr;
        — Pointer to list of resource are provided by this hw_node.
        — resource_available and resource_need should be merged.

        hw_link_list                    : DIS_hw_link_ptr;
        — Pointer to list of hardwarc links to which this node
        — is connected.

        next_hw_node                    : DIS_hw_node_ptr;
        previous_hw_node                : DIS_hw_node_ptr;
        — Pointer to next/previous hardware node in the current
        — hardware graph.
        hw_node_internal_node_list         : DIS_hw_node_ptr;
        hw_node_internal_edge_list         : DIS_hw_link_ptr;
        — Pointers to sub–component nodes and links with their
        — topology, if this hardware node is built from many
        — sub–components.

        parent_hw_group_node               : DIS_hw_group_node_ptr;
        parent_hw_node                     : DIS_hw_node_ptr;
        — Pointers to parent hardware group node or parent
        — hardware node.

        hw_node_sdf                        : DIS_SDF_Template_ptr;
        — Pointer to system design factor of hardware node.

        user_extensible_var                : DIS_user_extensible_ptr;
        — Pointer to the DIS_user_extensible type

end record;

type DIS_hw_link  is

    record
        hw_link_id                         : DIS_hw_link_id ;
        hw_link_name                       : FIELDS;
        — Identifier and name of hardware link.

        hw_link_generic_kind               : DIS_hw_link_g ;
        hw_link_specific                   : DIS_hw_link_spec;
        — Generic type and specification of hardware link.

        hw_link_data_rate                  : TBD;
        hw_link_data_latency               : TBD;
        hw_link_protocol                   : TBD;
        — Data rate and latency of hardware link and its protocol

        parent_hw_group_node               : DIS_hw_group_node_ptr;
        parent_hw_node                     : DIS_hw_node_ptr;
        parent_hw_group_link               : DIS_hw_group_link_ptr;
        — Pointers to parent hardware group node or parent hardware
        — node, and parent hardware group link.

        next_hw_link_next                  : DIS_hw_link_ptr;
        previous_hw_link_next              : DIS_hw_link_ptr;
        — Pointers to next/previous hardware node in current
        — hardware link

        hw_link_sdf                        : DIS_SDF_Template_ptr;
        — Pointer to system design factor for hardware link
```

```
          user_extensible_var                       : DIS_user_extensible_ptr;
          — Pointer to the DIS_user_extensible type

end record;

—/*=================================================== */
—/*                    MAPPING   ASSIGNMENT                       */
—/*                                                               */
—/* The goal of the mapping assignment is to assign each task     */
—/* in the software structure to the specific hardware node       */
—/* in the hardware structure with some constraints imposed among */
—/* tasks or tasks and hardware node. A mapping assignment        */
—/* consists of mapping constraints and task assignment . There are */
—/* two types of mapping constraints: timing constraint and       */
—/* placement constraint. Each mapping constraint includesa preference */
—/* value that specifies the importance of meeting the mapping    */
—/* constraint; the magnitude of the value.                       */
—/*_______________________________________________________________ */

type DIS_hardw_softw_pair ;
type DIS_hardw_softw_pair_ptr is access DIS_hardw_softw_pair ;
          — This is an open declaration for hardware and software pair type
          — to make a pointer to the dynamic storage location in memory.

subtype  DIS_hardw_id          is DIS_hw_node_id;
          — renaming of DIS_hardware_id type to DIS_hardw_id in
          — MAPPING VIEW

subtype  DIS_softw_id          is DIS_sw_task_id;
          — Renaming of DIS_sw_task_id type to DIS_hardw_id in
          — MAPPING VIEW

type DIS_hardw_softw_pair  is

          —/*___________________________________________________ */
          —/* A mapping pair of  a task in software structure and */
          —/* node in hardware structure is used to specify a task and */
          —/* module assignment to a hardware component, and to   */
          —/* specify assignment preferences.                     */
          —/*___________________________________________________ */

     record

          hardw_id                                  : DIS_hardw_id ;
          — Hardware node identifier
```

```
      softw_id                              : DIS_softw_id ;
      — Software task identifier

      parent_mapping_view                   : DIS_mapping_view_ptr;
      — Pointer to mapping view

      next_hardw_softw_pair                 : DIS_hardw_softw_pair_ptr;
      previous_hardw_softw_pair             : DIS_hardw_softw_pair_ptr;
      — Pointers to next/previous hardw_softw_pair

end record;

type DIS_softw_id_list ;
type DIS_softw_id_list_ptr is access DIS_softw_id_list ;
      — This is an open declaration for software list type
      — to make a pointer to the dynamic storage location in memory.

type DIS_time_constraint ;
type DIS_time_constraint_ptr is access DIS_time_constraint ;
      — This is an open declaration for time constraint type to make a
      — pointer to the dynamic storage location in memory.

type DIS_place_constraint ;
type DIS_place_constraint_ptr is access DIS_place_constraint ;
      — This is an open declaration for place constraint type to make a
      — pointer to the dynamic storage location in memory.

type DIS_softw_id_list  is

      —/*—————————————————————————————————————————————————————*/
      —/* List of software task identifiers with timing and placement */
      —/* constraints to match them to the specific hardware node. */
      —/*—————————————————————————————————————————————————————*/

      record
        softw_id                            : DIS_softw_id ;
        — Software task identifier.

        time_constraint                     : DIS_time_constraint_ptr;
        — The timing constraint

        place_constraint                    : DIS_place_constraint_ptr;
        — Placement constraint.

        next_softw_id                       : DIS_softw_id_list_ptr;
        place_softw_id                      : DIS_softw_id_list_ptr;
        — Pointer to next/previous software task in list.
```

```
end record;

type DIS_hardw_id_list ;
type DIS_hardw_id_list_ptr is access DIS_hardw_id_list;
        — This is an open declaration for hardware list type to make a
        — pointer to the dynamic storage location in memory.

type DIS_hardw_id_list  is

—/*------------------------------------------------------------------------*/
—/* List of hardware node to be matched to software tasks.                  */
—/*------------------------------------------------------------------------*/

record
    hardw_id                              : DIS_hardw_id ;
    — Hardware node identifier

    next_hardw_id                         : DIS_hardw_id_list_ptr;
    previous_hardw_id                     : DIS_hardw_id_list_ptr;
    — Pointers to next/previous hardware identifier in list.
end record;

type DIS_mapping_constraint;
type DIS_mapping_constraint_ptr is access DIS_mapping_constraint;
    — This is an open declaration for mapping constraint type to
    — make a pointer to the dynamic storage location in memory.

type DIS_time_constraint_kind    is (

    —/*--------------------------------------------------------------------*/
    —/*                      The types of timing constraints                */
    —/*--------------------------------------------------------------------*/

            complete_within,
            — Tasks A,B,...,C should complete within time_value

            start_within,
            — Tasks A,B,...,C should start within time_value

            complete_path_within,
            — Sequence of tasks, A,B,...,C should complete within
            — time_value

            complete_start_within);
            — For two tasks, A and B, B should start within time_value
            — after the completion of A.
```

```
type DIS_time_constraint  is

        —/*————————————————————————————————————————————————————————*/
        —/* The  timing  constraint  class. It  consists  of  its  constraint  */
        —/* kind, preference  value  specifying  the  importance  of  meeting */
        —/* the  mapping  constraint, and  list  of  software  tasks  in  the   */
        —/* current  timing  contraint. It  also  includes  hierachical          */
        —/* relations  with  mapping  constraint.                                */
        —/*————————————————————————————————————————————————————————*/

    record

        time_constraint_kind                : DIS_time_constraint_kind;
        — Type of timing constraint

        preference_value                    : DIS_preference_range ;
        — Preference of timing constraint

        time_value                          : DIS_time_type ;
        — Time value of constraint

        softw_id_list                       : DIS_softw_id_list_ptr;
        — Pointer to list of software task in current timing
        — constraints

        parent_mapping_constrain           t: DIS_mapping_constraint_ptr;
        — Pointer to parent mapping constraint

        next_time_constraint                : DIS_time_constraint_ptr;
        previous_time_constraint            : DIS_time_constraint_ptr;
        — Pointer to the next and previous time constraints.

        user_extensible_var                 : DIS_user_extensible_ptr;
        — Pointer to the DIS_user_extensible type

end record;

type DIS_place_constraint_kind is  (

        —/*————————————————————————————————————————————————————————*/
        —/*              The  type  of  the  placement  constraint.          */
        —/*————————————————————————————————————————————————————————*/

        place_together,
        — Tasks A,B,..,C should be assigned to the same hardware

        place_separate,
        — Tasks A,B,..,C should be assigned to different hardware
```

```
          place_at);
          — Tasks A,B,..,C should be assigned to the particular hardware

type DIS_place_constraint  is

          —/*------------------------------------------------------------*/
          —/* The placement constraint for software tasks to be placed */
          —/* at certain hardware node. This consists types of         */
          —/* placement constraint, preference value, list of software */
          —/* tasks and hardware node identifier. It also includes     */
          —/* pointer to parent mapping constraint.                    */
          —/*------------------------------------------------------------*/

     record
          place_constraint_kind                : DIS_place_constraint_kind;
          preference_value                     : DIS_preference_range ;
          — For place_at constraint, we need to specify hardw_id

          hardw_id                              : DIS_hardw_id ;
          — Identifier of hardware to which some tasks are assigned.

          softw_id_list                        : DIS_softw_id_list;
          — Pointer to the list of task identifiers which are assigned
          — to the above hardware component.

          parent_mapping_constrain             t: DIS_mapping_constraint_ptr;
          — Pointer to the parent mapping constraint.

          next_place_constraint                : DIS_place_constraint_ptr;
          previous_place_constraint            : DIS_place_constraint_ptr;
          — Pointers to the next and previous placement constraint.

          user_extensible_var                  : DIS_user_extensible_ptr;
          — Pointer to the DIS_user_extensible type

end record;

type DIS_mapping_constraint  is

          —/*------------------------------------------------------------*/
          —/* The mapping constraint consists of timing and placement */
          —/* constraints, including pointer to parent mapping view.  */
          —/*------------------------------------------------------------*/

    record
          timing_constraint                    : DIS_time_constraint_ptr;
          — Pointer to the timing constraint.
```

```
        placement_contraint             : DIS_place_constraint_ptr;
        — Pointer to the placement constraint.

        parent_mapping_view             : DIS_mapping_view_ptr;
        — Pointer to the parent mapping view.

end record;

type DIS_user_extensible is

        —/*--------------------------------------------------------*/
        —/* This will be the extensible types or variables defined by */
        —/* user  beside the predefined fields in each type, such as  */
        —/* software module, software module edge, task node, task   */
        —/* edge, hardware node, hardware link, etc. User can define */
        —/* the unique id, name, type, and value of this variable for */
        —/* his/her own specification on those types. And these will */
        —/* be linked to define multiple types or variables.         */
        —/*--------------------------------------------------------*/

    record
        id                              : DIS_id_type;
        name                            : FIELDS;
        — Identifier and name of this user extensible type.

        ext_type                        : DIS_user_extensible_type;
        — Type of this user extensible type.

        value                           : DIS_user_extensible_value;
        — Value of this user extensible type.

        next_user_extensible            : DIS_user_extensible_ptr;
        — Pointer to the next user expensible type.
    end record;

type DIS_SDF_Quantification;
type DIS_SDF_Quantification_ptr is access DIS_SDF_Quantification;
        — This is an open declaration for System Design Factor
        — Quantification type to make a pointer to the dynamic
        — storage location in memory.

type DIS_SDF_Consistency_Rule;
type DIS_SDF_Consistency_Rule_ptr is access DIS_SDF_Consistency_Rule;
        — This is an open declaration for System Design Factor
        — Consistency Rule type to make a pointer to the dynamic
        - storage location in memory.
```

```
type DIS_SDF_Template is

        —/*———————————————————————————————————————————————————————— */
        —/*  This System Design Factor(SDF) is to optimize the design */
        —/*  to meet the requiremments and desired measure of         */
        —/*  effectiveness. The design goals and criteria in this SDF */
        —/*  are specified by the system designers and analysists to  */
        —/*  qualify the various aspects of the design and to perform */
        —/*  the trade–offs among different design goals. Respect to  */
        —/*  the type of the system, it describes the properties,     */
        —/*  attributes and characteristics of the system.  Each SDF  */
        —/*  must have its own merit to gauge every detail of the     */
        —/*  system. This merit describes the weakness and strengths  */
        —/*  of a specific area in the design. In turn, the           */
        —/*  correlation of the SDF characterizes the completeness and */
        —/*  robustness.                                              */
        —/*———————————————————————————————————————————————————————— */

    record
        Temp_id                              : DIS_id_type;
        Temp_name                            : STRING(1..128);
        Temp_type                            : STRING(1..20);
        Temp_range                           : STRING(1..50);
        Temp_units                           : STRING(1..20);
        Temp_method_or_principle             : STRING(1..240);
        Temp_priority                        : STRING(1..20);
        Temp_accuracy                        : STRING(1..20);
        Temp_rational                        : STRING(1..80);
        Temp_relationship                    : STRING(1..60);
        Temp_quantification                  : DIS_SDF_Quantification_ptr;
        Temp_consistency_rule                : DIS_SDF_Consistency_Rule_ptr;
        Temp_reference                       : STRING(1..240);
        Temp_definition                      : STRING(1..240);
        Temp_annotation                      : STRING(1..240);
        next_SDF_template                    : DIS_SDF_Template_ptr;
    end record;

type DIS_SDF_Qnty_Formula;
type DIS_SDF_Qnty_Formula_ptr is access DIS_SDF_Qnty_Formula;

type DIS_SDF_Quantification is
     record
          Qnty_type                          : integer;
          Qnty_formula                       : DIS_SDF_Qnty_Formula_ptr;
        end record;

    type DIS_SDF_Qnty_Formula is
        record
            Fm_aggregate                        : STRING(1..10);
            Fm_var_list                         : STRING(1..20);
            next_formula                        : DIS_SDF_Qnty_Formula_ptr;
        end record;

    type DIS_SDF_Consistency_Rule is
        record
            Con_aggregate                       : STRING(1..10);
            Con_type                            : STRING(1..200);
            Con_design_factor                   : STRING(1..20);
            Con_view                            : STRING(1..20);
            Con_component                       : STRING(1..20);
        end record;

    — THIS IS THE RESULT OF AN ALLOCATION ALGORITHM.

    type DIS_mapping_view  is

        —/*___________________________________________________________ */
        —/* Mapping view mainly consists of mapping constraint and     */
        —/* assignment of hardware task to the hardware nodes. It      */
        —/* also includes allocation tool to be determined in later    */
        —/* stage of development, as well as pointer to the            */
        —/* parent implementation view and to sibling mapping views    */
        —/* in mapping list.                                           */
        —/*___________________________________________________________ */

        record
            mapping_view_id                     : DIS_mapping_view_id;
            — Identifier of this mapping view.

            parent_implementation_view          : DIS_implementation_view;
            — Pointer to the parent implementation view of this mapping
            — view.

            allocation_tool_id                  : DIS_allocation_tool_id;
            — Identifier of allocation tool to be defined in the later
            — stage of developement.

            constraints                         : DIS_mapping_constraint_ptr;
            — Pointer to the mapping costraint which contains the timing
            — and placement constraints.
```

```
                    assignments                         : DIS_hardw_softw_pair_ptr;
                    — Pointer to the hardware node and software task mapping
                    — pair.

                    next_mapping_view                   : DIS_mapping_view_ptr;
                    previous_mapping_view               : DIS_mapping_view_ptr;
                    — Pointers to the next and previous mapping views.

          end record;

end DIS_implementation_model;
```

FINAL REPORT

APPENDIX D

C++ SPECIFICATIONS FOR REPRESENTING IMPLEMENTATION MODEL

SYSTEM ENGINEERING AUTOMATION (SEA) FOR DISTRIBUTED SYSTEMS

CONTRACT NO. N00014–91–C–0183
CDRL SEQUENCE NO. A002

AUGUST 1992

```
#include <stdio.h>
#include <string.h>

// SYSTEM DESIGN FACTOR FILE

#include "sdf_spec.cc"

/*==================================================================
                    DIS DECLARATIONS-IMPLEMENTATION MODEL
==================================================================*/

// SHOULD BE SET TO THE DESIRED NUMBER.
// int MAX_LENGTH = 100;                   // (To be determined) and assigned
                                           // appropriate type and value

typedef int TBD;                           // To be determined.

typedef int id_type;                       // Is used for type of identification.
                                           // If it is necessary to modify
                                           // identifier of type, this type can
                                           // be changed as to the desired type.

typedef id_type DIS_sw_structure_diagram_id; // Is used to identify the
                                           // software structure diagram.

typedef id_type DIS_sw_module_id;          // Is used to identify the software
                                           // module.

typedef id_type DIS_sw_module_edge_id;     // Is used to identify the software
                                           //module edge.

typedef id_type DIS_sw_task_id;            // Is used to identify the software
                                           // module task.

typedef id_type DIS_sw_task_edge_id;       // Is used to identify the software
                                           // task edge.

typedef id_type DIS_hw_structure_diagram_id; // Is used to identify the
                                           // hardware structure diagram.

typedef id_type DIS_hw_group_node_id;      // Is used to identify the hardware
                                           //group node.

typedef id_type DIS_hw_group_link_id;      // Is used to identify the hardware
                                           // group link.

typedef id_type DIS_hw_node_id;            // Is used to identify the hardware node.
typedef id_type DIS_hw_link_id;            // Is used to identify the hardware
                                           // link.
```

```
typedef  id_type  DIS_mapping_view_id;         // Is used to identify the mapping view.
typedef  int  DIS_allocation_tool_id;          // Is used to identify the software
                                               // module diagram.

typedef  int  DIS_preference_range;            // Is used to identify the software
                                               // module diagram.

typedef  id_type  DIS_data_attribute_id;       // Is used to identify the DIS data
                                               // attribute.

typedef  int  DIS_time_value;                  // Is used to represent DIS time value.

typedef  int  DIS_data_size;                   // Is used to represent DIS data size.

typedef  int  DIS_resource_amount;             // Is used to represent DIS resource
                                               // amount.

typedef  int  DIS_user_extensible_type         // Is used to represent the type of
                                               // DIS_user_extensible. This type will be
                                               // defined in the later stage of the
                                               // implementation specification development.

typedef  int  DIS_user_extensible_value        // Is used to represent the value of
                                               // DIS_user_extensible type. This type will
                                               // be defined in the later stage of the
                                               // implementation specification development.
/*
class  DIS_name  {  public:  DIS_name(int);    // String constructor
                    private: int len;char *str;
};
*/

/*=====================================================================
                         IMPLEMENTATION  VIEW
=====================================================================*/

class  DIS_sw_structure_diagram;               // This is an open declaration for
                                               // software structure diagram class
                                               // to make a pointer to the dynamic
                                               // storage location in memory.

class  DIS_hw_structure_diagram;               // This is an open declaration for
                                               // hardware structure diagram class
                                               // to make a pointer to the dynamic
                                               // storage location in memory.

class  DIS_mapping_view;                       // This is an open declaration for
                                               // mapping view class to make a pointer
```

```
                                              // to the dynamic storage location in
                                              // memory.

class DIS_implementation_view {

        // Class for the implementation model declaration

        DIS_sw_structure_diagram       *sw_structure_diagram;
                                              // Pointer to the software structure
                                              //diagram

        DIS_hw_structure_diagram       *hw_structure_diagram;
                                              // Pointer to the hardware structure
                                              // diagram

        DIS_mapping_view               *imp_mapping_list;
                                              // Pointer to the mapping view

        DIS_implementation_view        *next_implementation_view;
                                              // Pointer to the next implementation
                                              // view: successor view

        DIS_implementation_view        *previous_implementation_view;
                                              // Pointer to the previous
                                              // implementation view: predecessor view
};

/*==================================================================*/
/*                                                                  */
/*                      SOFTWARE  STRUCTURE                         */
/*                                                                  */
/*  Each software structure diagram is represented by a list of modules    */
/*  and a list of edges between modules. Modules can be nested and each    */
/*  module includes its own task graph. The task graph cannot be nested    */
/*  since the node of a task graph cannot be a module; however, nested     */
/*  relations between tasks can be captured using nested modules.          */
/*  The task represents a computational entity.                            */
/*------------------------------------------------------------------*/

class  DIS_sw_module;                         // This is an open declaration for
                                              // software module class to make
                                              // a pointer to the dynamic storage
                                              // location in memory.

class  DIS_sw_module_edge;                    // This is an open declaration for
                                              // software module edge class to make
```

```
                                        // a pointer to the dynamic storage
                                        // location in memory.

class DIS_sw_structure_diagram {

    /*-------------------------------------------------------------*/
    /* The software structure diagram is used to reference a       */
    /* collection of directed graphs, drawn with respect to a      */
    /* selected methodology, that captures information about a set */
    /* of components and their relations along with any hierarchical */
    /* decomposition. For example, a tree of data flow diagrams    */
    /* may be considered as one type of structure diagram.         */
    /*-------------------------------------------------------------*/

    DIS_sw_structure_diagram_id     sw_structure_diagram_id;
    DIS_name                        sw_structure_diagram_name;
                                        // Identifier and name of software
                                        // structure diagram

      DIS_sw_module                      *sw_module_list;
                                        // Pointer to the double-linked list
                                        // of the software modules
                                        // as children of this software
                                        // structure diagram

      DIS_sw_module_edge               *sw_module_edge_list;
                                        // Pointer to the double-linked list
                                        // of the software module edges
                                        // interconnecting the children
                                        // modules of this software structure
                                        // diagram

      DIS_implementation_view         *parent_implementation_view;
                                        // Pointer to the parent implementation
                                        // view of this software
                                        // structure diagram.

      DIS_sw_structure_diagram        *next_sw_diagram;
                                        // Pointer to the next software
                                        // structure diagram: successor

      DIS_sw_structure_diagram        *previous_sw_diagram;
                                        // pointer to the previous software
                                        // structure diagram:predecessor
};
```

```
class  DIS_sw_task_node;                      // This is an open declaration for
                                              // software task node class to make
                                              // a pointer to the dynamic storage
                                              // location in memory.

class  DIS_sw_task_edge;                      // This is an open declaration for
                                              // software task edge class to make
                                              // a pointer to the dynamic storage
                                              // location in memory.

class  type  DIS_user_extensible_ptr;         // This is an open declaration for user
                                              // extensible variable type to make a
                                              // pointer to the dynamic storage
                                              // location in memory.

class DIS_sw_module {

        /*______________________________________________________________________*/
        /*  A software module class contains the following information:         */
        /*                                                                      */
        /*  1. The hierarchical, sibling and nesting relations between          */
        /*        modules.                                                      */
        /*  2. The identity of task graphs that belong to the module.           */
        /*                                                                      */
        /*  In addition, there are two special kind of edges (called            */
        /*  entry_super_edge and exit_super_edge). They are used to             */
        /*  identify the entry and exit points of the task graph at the         */
        /*  module level.                                                       */
        /*______________________________________________________________________*/

        DIS_sw_module_id                      module_id;
        DIS_name                              module_name;
                                              // Identifier and name of software
                                              // module

        DIS_sw_structure_diagram              *parent_sw_structure;
                                              // Pointer to parent software
                                              // structure diagram

        DIS_sw_module                         *parent_module;
                                              // Pointer to the parent software
                                              // module if any

        DIS_sw_module                         *next_module;
        DIS_sw_module                         *previous_module;
                                              // Pointer to previous/next software
                                              // modules as successor/predecessor
```

```
DIS_sw_module                   *submodule_list;
                                // Pointer to the list of the children
                                // submodules

DIS_sw_module_edge              *module_edges;
                                // Pointer to the list of software
                                // module edge defining links
                                // between super edges of the submodules

DIS_sw_task_node                *task_node_list;
DIS_sw_task_edge                *task_edge_list;
                                // Pointers to the lists of the
                                // software task nodes belongs to this
                                // module and of software task edges
                                // defining links between tasks:
                                // that is, the task graph. A task
                                // graph is a directed graph: each
                                // node denotes a schedulable
                                // computational entity and an edge
                                // represents a precedence relation
                                // between two nodes.

DIS_sw_task_edge                *entry_super_edge_list;
DIS_sw_task_edge                *exit_super_edge_list;
                                // Two special kinds of edges, called
                                // enter_super_edge & exit_super_edge,
                                // are pointers to the lists of
                                // software task edges that are to be
                                // visible outside the current module.
                                // A super edge to an entering to or
                                // from exiting node of task_graph.
                                // Each entry_super_edge and exit_super
                                // _edge are either a task edge
                                // or a entry_super_edge/exit_super
                                // _edge of a submodule.

DIS_SDF_Template                *module_sdf;
                                // Pointer to System Design Factor
                                // Template for this module

DIS_user_extensible_ptr         *user_extensible_var;
                                // Pointer to the DIS_user_extensible type
```

```
};

class DIS_sw_module_edge {

        /*------------------------------------------------------------------*/
        /* The software module edge represents the link between              */
        /* modules, and supports the hierarchical orders/relations of        */
        /* the software module organization as well as the list of super     */
        /* edges belonging to this software module edge.                     */
        /*------------------------------------------------------------------*/

        DIS_sw_module_edge_id           module_edge_id;
        DIS_name                        module_edge_name;
                                        // Identifier and name of the software
                                        // module edge

        DIS_name                        attributes;
                                        // Attribute of the software module
                                        // edge

        DIS_sw_structure_diagram        *parent_sw_structure;
                                        // Pointer to the parent software
                                        // structure diagram

        DIS_sw_module                   *from_module;
        DIS_sw_module                   *to_module;
                                        // Source and destination pointer for
                                        // the software module edge

        DIS_sw_task_edge                *super_edge_list;
                                        // Point to the list of the super
                                        // edges belonging to this
                                        // software module edge.

        DIS_sw_module_edge              *next__module_edge;
        DIS_sw_module_edge              *previous__module_edge;
                                        // Pointers to the next/previous
                                        // module edges: successor/predecessor

        DIS_SDF_Template                *module_edge_sdf;
                                        // pointer to the system design factor
                                        // template for software module edge

        DIS_user_extensible_ptr         *user_extensible_var;
                                        // Pointer to the DIS_user_extensible type
};
```

```
class  DIS_data_attribute;                  // This is an open declaration for
                                            // software data attribute class to
                                            // make a pointer to the dynamic
                                            // storage location in memory.

class  DIS_resource;                        // This is an open declaration for
                                            // software resource class to make
                                            // a pointer to the dynamic storage
                                            // location in memory.

enum DIS_time {Relative, Absolute};

     /*-------------------------------------------------------------------*/
     /* This specifies the type of DIS time, such that Absolute          */
     /* represent the clock time while Relative represents relative      */
     /* time length from some events.                                     */
     /*-------------------------------------------------------------------*/

class DIS_time_kind {

     /*-------------------------------------------------------------------*/
     /* This specifies the type of DIS time and its value                 */
     /*-------------------------------------------------------------------*/

     DIS_time             time_kind;
     DIS_time_value               time_value;
};

enum DIS_log_operators {log_and, log_or};

     /*-------------------------------------------------------------------*/
     /* This is a flag specifying the conditions for executing a         */
     /* task: whether all conditions (or output) data are needed         */
     /* (or generated) by the certain task.                               */
     /*-------------------------------------------------------------------*/

class DIS_sw_task_node {

     /*-------------------------------------------------------------------*/
     /* The software task node class specifies DIS_sw_task_node          */
     /* structure. There is input list to identify input data and an     */
     /* output list to identify output data generated by the task.       */
     /* In addition, predecessor list identifies tasks that execute      */
     /* before the task and successor list identifies task that          */
     /* execute after the task. There is an and/or flag associated       */
     /* with the above four task lists that specifies whether all        */
/*  input  (or  output)  data  are  needed  (  or  generated)  by  the           */
/*  task.  This  information  is  required  by  some  optimization               */
/*  algorithms.  Each  task  may  include  timing  information  such  as         */
/*  ready  time,  deadline  and  duration.  In  addition,  it  identifies        */
/*  resources  it  needs.  For  resource  needs,  resource  type                 */
/*  identifies  the  resource  a  task  needs  and  amount  it  needs.           */
/*___________________________________________________________________________*/

    DIS_sw_task_id                    task_id;
    DIS_name                          task_name;
                                      // Identifier and name of
                                      // software task node

    DIS_sw_module                    *parent_module;
                                      // Pointer to the parent software module

    TBD                              task_structure;
    TBD                               task_description;
                                      // Structure and description of the
                                      // task node( will be determined
                                      // in later stage of development.

    DIS_sw_task_edge                  *task_edge_list;
                                      // Task_edge's from or to this task_node

    DIS_log_operators                 task_input_and_or;
    DIS_data_attribute                *task_input_list;
    DIS_log_operators                 task_out_and_or;
    DIS_data_attribute                *task_output_list;
                                      // Task data dependencies

    DIS_log_operators                 task_before_and_or;
    DIS_sw_task_node                  *task_before_list;
    DIS_log_operators                 task_after_and_or;
    DIS_sw_task_node                  *task_after_list;
                                      // Task precedence relations

    DIS_time                          task_ready_time;
    DIS_time                          task_deadline;
    DIS_time                          task_duration;
                                      // Timing information

    DIS_resource                     *task_resource_needs;
                                      // Resource needs

    DIS_sw_task_node                  *task_buddy_task;
                                      // The cooperating tasks
```

```
        TBD                                 task_max_replication;
        TBD                                 task_priority;
        TBD                                 task_execution_probability;
        TBD                                 task_communication_delay_matrix;
                                            // the fields will be defined in
                                            // later stage.

        DIS_sw_task_node                    *next_task;
        DIS_sw_task_node                    *previous_task;
                                            // Next/previous task in the linked list

        TBD                                 task_error_cumulation;
        TBD                                 task_imprecise_error_convergence;
                                            // Univ. Illinois imprecise
                                            // computation support

        DIS_SDF_Template                    *task_sdf;
                                            // Pointer to the Task Design Factor
                                            // template

        DIS_user_extensible_ptr             *user_extensible_var;
                                            // Pointer to the DIS_user_extensible type
};

class DIS_sw_task_edge {

        /*------------------------------------------------------------------*/
        /*  The software task edge specifies the relations between          */
        /*  software task nodes and software module nodes. For each task    */
        /*  edge, task_data_edge identifies the data associated with the    */
        /*  edge along with the duration of availability of the data.       */
        /*  In addition, from_task_node and to_task_node specifies the      */
        /*  source and  destination of the edge.                            */
        /*------------------------------------------------------------------*/

        DIS_sw_task_edge_id                 task_edge_id;
        DIS_name                            task_edge_name;
                                            // Identifier and name of the software
                                            // task edge

        DIS_sw_module                       *parent_module;
        DIS_sw_module_edge                  *parent_module_edge;
                                            // Pointer to the parent software
                                            // module and module edge
        DIS_data_attribute              *task_edge_data;
                                        // Pointer to the data attributes
                                        // associated to this edge

        DIS_sw_task_node                *from_task_node;
        DIS_sw_task_node                *to_task_node;
                                        // Pointer to the source and
                                        // destination software task node.

        DIS_sw_task_edge                *next_task_edge;
        DIS_sw_task_edge                *previous_task_edge;
                                        // Pointer to the next/previous task
                                        // node in task edge list
                                        // where this edge belongs to.

        DIS_SDF_Template                *task_edge_sdf;
                                        // Pointer to the system design factor
                                        // template.
};
enum DIS_d {Msg, SharedMemory};
        /*______________________________________________________________*/
        /* The type of data based on the paradigm of message-passing    */
        /* or shared memory.                                             */
        /*______________________________________________________________*/

class DIS_data_attribute {
        /*______________________________________________________________*/
        /* The data attribute specifies the type and size of data being */
        /* communicated through edges between tasks. This points to the */
        /* list of sender/receiver tasks specified by the list of their */
        /* respective edges between them. In addition, it lists the     */
        /* resource needed for this data attribute as well as timing    */
        /* constraint of data-deadline and data frequency (to be        */
        /* defined in the later stage of the development.               */
        /*______________________________________________________________*/

        DIS_data_attribute_id           data_attribute_id;
        DIS_name                        data_attribute_name;
                                        // Identifier and name of the data
                                        // attributes

        DIS_d                           data_kind;
        DIS_data_size                   data_size;
```

```
                                                // The kind and size of the data in
                                                // this data attributes.

        DIS_sw_task_edge                        *task_edge_list;
                                                // The list of the software task
                                                // edges through which data
                                                // being transmitted.

        DIS_log_operators                       sender_kind;
        DIS_sw_task_node                        *data_sender_list;
        DIS_log_operators                       receiver_kind;
        DIS_sw_task_node                        *data_receiver_list;
                                                // Kind of log operation (and/or) for
                                                // receiver/sender, and lists of
                                                // senders/receivers.

        DIS_resource                            *data_resource_need_list;
                                                // List of the resources needed for
                                                // this data attribute

        DIS_time                                data_deadline;
        TBD                                     data_frequency;
                                                // The timing constraint of data
                                                // deadline and data frequency to be
                                                // determined in the later stage of
                                                // development.

        DIS_user_extensible_ptr                 *user_extensible_var;
                                                // Pointer to the DIS_user_extensible type
};

enum DIS_resource_type {CPU_r, Memory_r, IO, Communication};

        /*-----------------------------------------------------------------*/
        /*                      Kind of resource                           */
        /*-----------------------------------------------------------------*/

enum DIS_resource_u {KIPS, MIPS, Bytes, Kbytes, Mbytes, Sec, MilliSec,
                     MicroSec};

        /*-----------------------------------------------------------------*/
        /* Unit time of each resource type:                                */
        /*     KIPS, MIPS, or spec mark for CPU specification.             */
        /*     Bytes, KBytes, MBytes for memory,                           */
        /*     (Bytes, KBytes, MBytes) per (Sec, MilliSec, MicroSec)       */
        /*     for IO and communication.                                   */
        /*-----------------------------------------------------------------*/
```

```
class DIS_resource_unit {
        /*---------------------------------------------------------------------*/
        /*              Resource   unit   and   its   amount                    */
        /*---------------------------------------------------------------------*/

        DIS_resource_u                  resource_unit;
        DIS_resource_amount             resource_amount;
};

class  DIS_hw_node;                     // This is an open declaration for
                                        // hardware node class to make a
                                        // pointer to the dynamic storage
                                        // location in memory.

class DIS_resource {
        /*---------------------------------------------------------------------*/
        /*  Resource specifies its size and units and pointers to the          */
        /*  related nodes, edges, and data attributes. It also contains        */
        /*  pointer to the hardware node where it can be defined by a          */
        /*  hardware configuration.                                            */
        /*---------------------------------------------------------------------*/

        DIS_resource_type               resource_kind;
        DIS_resource_unit               resource_units;
                                        // Resource type and unit

        DIS_sw_task_node                *task_node_list;
        DIS_sw_task_edge                *task_edge_list;
        DIS_data_attribute              *resource_data_attribute;
                                        // Pointer to the task node, edge,
                                        // and data attributes related to this
                                        // resource or that need this resource.

        DIS_hw_node                     *hw_node_list;
                                        // Hardware node being extracted from
                                        // the hardware structure

        DIS_resource                    *next_resource_need;
        DIS_resource                    *previous_resource_need;
                                        // Pointer to the next/previous
                                        // resource available or required
                                        // in the list.

        DIS_user_extensible_ptr         *user_extensible_var;
                                        // Pointer to the DIS_user_extensible type
};
```

```
/*============================================================================
/*                        HARDWARE  STRUCTURE                               */
/*                                                                          */
/*  A hardware structure diagram defines a hardware configuration. Each     */
/*  hardware structure diagram is represented by a list of group nodes      */
/*  and a list of group links with their communication topology between    */
/*  group nodes. Group nodes can be nested and each group node includes    */
/*  its own hardware node graph. Unlike task graph in software structure,  */
/*  hardware node graph can be nested. Our view is that the hardware       */
/*  node represents a hardware component in a computer architecture,       */
/*  such as a processor, CPU, memory, IO, etc.                              */
/*--------------------------------------------------------------------------*/

class  DIS_hw_group_node;                // This is an open declaration for
                                         // hardware group node class to make a
                                         // pointer to the dynamic storage
                                         // location in memory.

class  DIS_hw_group_link;                // This is an open declaration for
                                         // hardware group link class to make a
                                         // pointer to the dynamic storage
                                         // location in memory.

enum DIS_hw_group_link_topology {

        /*------------------------------------------------------------------*/
        /*  The various ways of physically connecting hardware group        */
        /*  nodes with communications. Here are the generally known        */
        /*  types of communication topology existing today.                 */
        /*------------------------------------------------------------------*/

        fully_connected,                 // All group nodes are directed linked
                                         // with all other group nodes.

        partially_connected,             // Some group nodes are directly linked
                                         // with some other groups nodes, but
                                         // not all.

        hierarchical,                    // Group nodes are organized or linked
                                         // as s tree.

        star,                            // One of the group nodes is connected
                                         // to all other group nodes. Node of
                                         // the other nodes are connected to
                                         // each other.
        ring,                           // Each group node is physically
                                        // connected to exactly two other
                                        // group nodes.

        multi_access_bus                // There a single shared hardware group
                                        // links. All group nodes in the system
                                        // are directly connected to that group
                                        // link.
};

class DIS_hw_structure_diagram {

        /*------------------------------------------------------------------*/
        /*  Each view of DIS_hardware_structure consists of                 */
        /*      - a list of group node,                                     */
        /*      - a list of group edges and communication topology          */
        /*        between group node.                                       */
        /*  Similar to the software module in the hierarchical view,        */
        /*  group nodes can be nested recursively. Each group node may      */
        /*  include its own hardware node graph with its specific           */
        /*  internal communication topology. Different with the software    */
        /*  task node in hierarchical perspective of configuration, the     */
        /*  hardware node can be nested recursively.                        */
        /*------------------------------------------------------------------*/

        DIS_name                        hw_structure_diagram_name;
        DIS_hw_structure_diagram_id     hw_structure_diagram_id;
                                        // Identifier and name of hardware
                                        // structure diagram

        DIS_implementation_view         *parent_implementation_view;
                                        // Pointer to parent implementation
                                        // model

        DIS_hw_group_node               *hw_group_node_list;
                                        // Pointer to the double-linked list
                                        // of hardware group nodes belonging
                                        // to this hardware structure diagram

        DIS_hw_group_link               *hw_group_link_list;
                                        // Pointer to the double-linked list
                                        // of hardware group links inter-
                                        // connecting hardware group nodes
                                        // of this hardware structure diagram
        DIS_hw_group_link_topology   hw_group_link_topology;
                                     // Communication topology of hardware
                                     // group nodes

        DIS_hw_structure_diagram     *next_hw_diagram;
        DIS_hw_structure_diagram     *previous_hw_diagram;
                                     // Pointers to the next/previous
                                     // hardware structure diagram:
                                     // successor/predecessor

};

class  DIS_hw_link;                  // This is an open declaration for
                                     // hardware link class to make a
                                     // pointer to the dynamic storage
                                     // location in memory.

typedef  DIS_hw_group_link_topology  DIS_hw_node_link_topology;
                                     // Internal hardware link topology

class DIS_hw_group_node {

        /*-------------------------------------------------------------*/
        /*  A hardware_group_node class contains the following         */
        /*  informations:                                              */
        /*      1. The hierarchical, sibling and nesting relations     */
        /*            between hardware group nodes                     */
        /*      2. The identity of hardware node graphs that belong to */
        /*            this hardware group node.                        */
        /*  For both graph, the communication topology can be specified. */
        /*  In addition, there are two special kinds of links (called  */
        /*  entry_super_link and exit_super_link). They are used to    */
        /*  identify the entry and exit points of the node graph at the */
        /*  group node level.                                          */
        /*-------------------------------------------------------------*/

        DIS_hw_group_node_id         hw_group_node_id;
        DIS_name                     hw_group_node_name;
                                     // Identifier and name of hardware
                                     // group node

        DIS_hw_structure_diagram     *parent_hw_structure;
                                     // Pointer to parent hardware
                                     // structure diagram
```

```
DIS_hw_group_node                     *parent_hw_group_node;
                                      // Pointer to parent hardware
                                      // group node

DIS_hw_group_node                     *next_hw_group_node;
DIS_hw_group_node                     *previous_hw_group_node;
                                      // Pointer to next/previous
                                      // hardware group node:
                                      // successor/predecessor

DIS_hw_group_node                     *sub_hw_group_node;
                                      // Pointer to list of sub-group nodes
                                      // as children of this group node

DIS_hw_group_link                     *sub_hw_group_link;
DIS_hw_group_link_topology   hw_group_link_topology;
                                      // Pointer to list of hardware group
                                      // links defining physical data
                                      // communication link between sub-
                                      // group nodes and their topology

// NODE GRAPHS belongs to this group node
DIS_hw_node                           *hw_node_list;
DIS_hw_link                           *hw_link_list;
DIS_hw_node_link_topology             *hw_node_link_topology;
                                      // Hardware node graph belongs to
                                      // this hardware group node: nodes,
                                      // links, and their link topology

DIS_hw_link                           *entry_super_link_list;
DIS_hw_link                           *exit_super_link_list;
                                      // There are two special kinds of
                                      // links, called enter_super_link and
                                      // exit_super_link, that are to be
                                      // visible outside the current group.
                                      // A super link to an entering or from
                                      // exiting node of the group_node.
                                      // Each entry_super_link is either a
                                      // hw_link or a entry_super_link
                                      // of a sub group node.
                                      // Each exit_super_link is either a
                                      // hw_link or a enter_super_link
                                      // of a sub_group_node.
```

```
        DIS_SDF_Template                        *group_node_sdf;
                                                // Pointer to system design factor

        DIS_user_extensible_ptr                 *user_extensible_var;
                                                // Pointer to the DIS_user_extensible type
                                                // template
};

class DIS_hw_group_link {

        /*----------------------------------------------------------------*/
        /* The hardware group link represents the physical                 */
        /* communication between hardware group nodes, and support the     */
        /* hierarchical orders/relations of the hardware group             */
        /* organization with its respective topology. It also points to    */
        /* the list of the super links belonging to this hardware          */
        /* group link.                                                     */
        /*----------------------------------------------------------------*/

        DIS_hw_group_link_id                    hw_group_link_id;
        DIS_name                                hw_group_link_name;
                                                // Identifier and name of hardware
                                                // group link

        DIS_hw_structure_diagram                *parent_hw_structure;
                                                // Pointer to parent hardware
                                                // structure diagram

        DIS_hw_group_node                       *from_hw_group_node;
        DIS_hw_group_node                       *to_hw_group_node;
                                                // Pointer to the source/destination
                                                // of this hardware group link

        DIS_hw_link                             *super_link_list;
                                                // Pointer to list of super links
                                                // belonging to this group node


        DIS_hw_group_link                       *next_hw_group_link;
        DIS_hw_group_link                       *previous__hw_group_link;
                                                // Pointers to nest/previous hardware
                                                // group links:successor/predecessor

        DIS_SDF_Template                        *group_edge_sdf;
                                                // Pointer to system design factor
                                                // template for hardware group link
        DIS_user_extensible_ptr              *user_extensible_var;
                                             // Pointer to the DIS_user_extensible type
};

enum hw_link_g {Bus, LAN};

        /*-----------------------------------------------------*/
        /* The genetic type of the hardware link                */
        /*-----------------------------------------------------*/

enum hw_link_spec {NotKnown_l, Ethernet, TokenRing};

        /*-----------------------------------------------------*/
        /*   The specification of the hardware link             */
        /*-----------------------------------------------------*/

enum hw_node_g {Processor, CPU_hw, Memory_hw, IOchannel, Other};

        /*-----------------------------------------------------*/
        /* The genetic type of the hardware node                */
        /*-----------------------------------------------------*/

enum hw_node_spec {NotKnown_n, Sun, RISC, Sparc};

        /*-----------------------------------------------------*/
        /* The specification of the hardware node               */
        /*-----------------------------------------------------*/

class DIS_hw_node {

        /*-----------------------------------------------------*/
        /* A hardware node graph is a directed graph: each node denotes*/
        /* an actual hardware component in computer architecture as */
        /* stated and link represents the physical communication line */
        /* or bus between hardware components. Each hardware node */
        /* identifies its type, specification and available resources. */
        /* Unlike task graph in software structure, it can be nested */
        /* recursively. Super links in this level indicate hardware link*/
        /* to hardware node in the different hardware nodes or group */
        /* nodes.                                                */
        /*-----------------------------------------------------*/

        DIS_name                             hw_node_name;
        DIS_hw_node_id                       hw_node_id;
                                             // Identifier and name of hardware
                                             // node
```

```
hw_node_g                 hw_node_generic;
hw_node_spec              hw_node_specific;
                          // Specific identification of node
                          // as known hardware component.
                          // This serves a key to database
                          // containing known hardware
                          // information.

DIS_resource              *hw_node_resource_available;
                          // Pointer to list of resource are
                          // provided by this hw_node.
                          // resource_available and
                          // resource_need should be merged.

DIS_hw_link               *hw_link;
                          // Pointer to list of hardware links
                          // to which this node is connected.

DIS_hw_node               *next_hw_node;
DIS_hw_node               *previous_hw_node;
                          // Pointer to next/previous hardware
                          // node in the current hardware graph.

DIS_hw_node               *hw_node_internal_node_list;
DIS_hw_link               *hw_node_internal_link_list;
                          // Pointers to sub–component nodes and
                          // links with their topology, if this
                          // hardware node is built from many
                          // sub–components.

DIS_hw_group_node         *parent_hw_group_node;
DIS_hw_node               *parent_hw_node;
                          // Pointers to parent hardware group
                          // node or parent hardware node.
```

```
        DIS_SDF_Template                    *hw_node_sdf;
                                            // Pointer to system design factor
                                            // of hardware node.

        DIS_user_extensible_ptr             *user_extensible_var;
                                            // Pointer to the DIS_user_extensible type
};

class DIS_hw_link {
        DIS_hw_link_id                      hw_link_id;
        DIS_name                            hw_link_name;
                                            // Identifier and name of hardware
                                            // link.

        hw_link_g                           hw_link_generic_kind;
        hw_link_spec                        hw_link_specific;
                                            // Generic type and specification
                                            // of hardware link.

        TBD                                 hw_link_data_rate;
        TBD                                 hw_link_data_latency;
        TBD                                 hw_link_protocol;
                                            // Data rate and latency of hardware
                                            // link and its protocol

        DIS_hw_group_node                   *parent_hw_group_node;
        DIS_hw_node                         *parent_hw_node;
        DIS_hw_group_link                   *parent_hw_group_link;
                                            // Pointers to
                                            // parent hardware group node or
                                            // parent hardware node, and parent
                                            // hardware group link.

        DIS_hw_link                         *next_hw_link;
        DIS_hw_link                         *previous_hw_link;
                                            // Pointers to next/previous hardware
                                            // node in current hardware link

        DIS_SDF_Template                    *hw_link_sdf;
                                            // Pointer to system design factor
                                            // for hardware link
};
```

```
/*================================================================*/
/*                    MAPPING ASSIGNMENT                           */
/*                                                                 */
/* The goal of the mapping assignment is to assign each task in the */
/* software structure to the specific hardware node in the hardware */
/* structure with some constraints imposed among tasks or tasks and */
/* and hardware node. A mapping assignment consists of mapping     */
/* constraints and task assignment . There are two types of mapping */
/* constraints: timing constraint and placement constraint. Each mapping */
/* constraint includes a preference value that specifies the importance */
/* of meeting the mapping constraint; the magnitude of the value   */
/* reflects its importance. And a task assignment is result of running */
/* an allocation algorithm on a pair of software structure and hardware */
/* structure within a set of timing and placement constraints. Task */
/* assignment are stored in DISmapping_result_type.                */
/*-----------------------------------------------------------------*/

typedef  DIS_hw_node_id                DIS_hardw_id;
                                       // Renaming of DIS_hardware id type
                                       // to DIS_hardw_id in MAPPING VIEW

typedef  DIS_sw_task_id                DIS_softw_id;
                                       // Renaming of DIS_sw_task_id type
                                       // to DIS_hardw_id in MAPPING VIEW

class DIS_hardw_softw_pair {

        /*---------------------------------------------------------*/
        /* a pair of mapping between a task in software structure and */
        /* node in hardware structure.                              */
        /*---------------------------------------------------------*/

        DIS_hardw_id                   hardw_id;
                                       // Hardware node identifier

        DIS_softw_id                   softw_id;
                                       // Software task identifier

        DIS_mapping_view               *parent_mapping_view;
                                       // pointer to mapping view

        DIS_hardw_softw_pair           *next_hardw_softw_pair;
        DIS_hardw_softw_pair           *previous_hardw_softw_pair;
                                       // Pointers to next/previous
                                       // hardw_softw_pair
};
```

```
class  DIS_time_constraint;                 // This is an open declaration for
                                            // time constraint class to make a
                                            // pointer to the dynamic storage
                                            // location in memory.

class      DIS_place_constraint;            // This is an open declaration for
                                            // place constraint class to make a
                                            // pointer to the dynamic storage
                                            // location in memory.

class DIS_softw_id_list {

       /*----------------------------------------------------------------*/
       /* List of software task identifiers with timing and placement    */
       /* constraints to match them to the specific hardware node.       */
       /*----------------------------------------------------------------*/

       DIS_softw_id                         softw_id;
                                            // Software task identifier.

       DIS_time_constraint                  *time_constraint;
                                            // The timing constraint

       DIS_place_constraint                 *place_constraint;
                                            // The placement constraint.

       DIS_softw_id_list                    *next_softw_id;
       DIS_softw_id_list                    *previous_softw_id;
                                            // Pointer to next/previous software
                                            // task in list.
};

class DIS_hardw_id_list {

       /*----------------------------------------------------------------*/
       /* List of hardware node to be matched to software tasks.         */
       /*----------------------------------------------------------------*/

       DIS_hardw_id                         hardw_id;
                                            // Hardware node identifier

       DIS_hardw_id_list                    *next_hardw_id;
       DIS_hardw_id_list                    *previous_hardw_id;
                                            // Pointers to next/previous
                                            // hardware identifier in list.
};
```

```
enum  DIS_time_constraint_kind {

        /*---------------------------------------------------------------*/
        /*  The  types  of  timing  constraints                          */
        /*---------------------------------------------------------------*/

        complete_within,                    // Tasks A, B,...,C should complete
                                            // within time_value

              start_within,                 // Tasks A, B,...,C should start
                                            // within time_value

              complete_path_within,         // Sequence of tasks, A,B,...,C
                                            // should complete within time_value

              complete_start_within         // For two tasks, A and B, B should
                                            // start within time_value after
                                            // the completion of A.
};

class  DIS_mapping_constraint;              // This is an open declaration for
                                            // mapping constraint class to make a
                                            // pointer to the dynamic storage
                                            // location in memory.

class DIS_time_constraint {

        /*---------------------------------------------------------------*/
        /*  The timing constraint class. This consists of its constraint */
        /*  kind, preference value specifying the importance of meeting  */
        /*  the mapping constraint, and list of software tasks in the    */
        /*  current timing constraint. This also includes hierarchical   */
        /*  relations with mapping constraint.                           */
        /*---------------------------------------------------------------*/

        DIS_time_constraint_kind            time_constraint_kind;
                                            //  Type of timing constraint

        DIS_preference_range                preference_value;
                                            // Preference of timing constraint

        DIS_time                            *time_value;
                                            // Time value of constraint

        DIS_softw_id_list                   *softw_id_list;
                                            // Pointer to list of software task
                                            // in current timing constraints
        DIS_mapping_constraint          *parent_mapping_constraint;
                                        // Pointer to parent mapping constraint

        DIS_time_constraint             *next_time_constraint;
        DIS_time_constraint             *previous_time_constraint;
                                        // Pointer to next/previous timing
                                        // constraint.

        DIS_user_extensible_ptr         *user_extensible_var;
                                        // Pointer to the DIS_user_extensible type
};
enum   DIS_place_constraint_kind {
        /*-------------------------------------------------------------*/
        /*  The type of the placement constraint.                      */
        /*-------------------------------------------------------------*/

        place_together,                 // Tasks A,B,..,C should be assigned
                                        // to the same hardware

                place_separate,         // Tasks A,B,...,C should be assigned
                                        // to different hardware

                place_at                // Tasks A,B,..,C should be assigned
                                        // to the particular hardware
};
class DIS_place_constraint {
        /*-------------------------------------------------------------*/
        /*  The placement constraint for software tasks to be placed at */
        /*  certain hardware node. This consists types of placement    */
        /*  constraint, preference value, list of software tasks, and  */
        /*  hardware node identifier. This also includes pointer to parent */
        /*  mapping constraint.                                        */
        /*-------------------------------------------------------------*/

        DIS_place_constraint_kind       place_constraint;
        DIS_preference_range            preference_value;
        DIS_hardw_id                    hardw_id;
                                        // For place_at constraint, we need
                                        // to specify DIS_hardw_id
        DIS_softw_id_list               *softw_id_list;
        DIS_mapping_constraint          *parent_mapping_constraint;
        DIS_place_constraint            *next_place_constraint;
        DIS_place_constraint            *previous_place_constraint;
        DIS_user_extensible_ptr             *user_extensible_var;
                                            // Pointer to the DIS_user_extensible type
};

class DIS_mapping_constraint {

        /*---------------------------------------------------------------------------*/
        /*  The mapping constraint consists of timing and placement                  */
        /*  constraints, including pointer to parent mapping view.                   */
        /*---------------------------------------------------------------------------*/

        DIS_time_constraint                 *timing_constraint;
        DIS_place_constraint                *placement_constraint;
        DIS_mapping_view                    *parent_mapping_view;
};

class DIS_user_extensible {

        /*---------------------------------------------------------------------------*/
        /*  This will be the extensible types or variables defined by                */
        /*  user  beside the predefined fields in each type, such as                 */
        /*  software module, software module edge, task node, task                   */
        /*  edge, hardware node, hardware link, etc. User can define                 */
        /*  the unique id, name, type, and value of this variable for                */
        /*  his/her own specification on those types. And these will                 */
        /*  be linked to define multiple types or variables.                         */
        /*---------------------------------------------------------------------------*/

              DIS_id_type                   id;
              FIELDS                        name;
                                            // Identifier and name of this
                                            // user extensible type.

              DIS_user_extensible_typext_type;
                                            // Type of this user extensible type.

             DIS_user_extensible_value      value;
                                            // Value of this user extensible type.

              DIS_user_extensible_ptr       next_user_extensible;
                                            // Pointer to the next user extensible type.
};
```

```
// THIS IS THE RESULT OF AN ALLOCATION ALGORITHM.

class DIS_mapping_view {

        /*--------------------------------------------------------------*/
            /* Mapping view mainly consists of mapping constraint and assign-  */
        /*  ment of hardware task to the hardware nodes. This also includes   */
        /*  allocation tool to be determined in later stage of development,   */
        /*  as well as pointer to the parent implementation view and to       */
        /*  sibling mapping views in mapping list.                             */
        /*--------------------------------------------------------------*/

        DIS_mapping_view_id             mapping_view_id;
        DIS_implementation_view         *parent_implementation_view;
        DIS_allocation_tool_id          allocation_tool_id;
        DIS_mapping_constraint          *constraint;
        DIS_hardw_softw_pair            *assignment;
        DIS_mapping_view                *next_mapping_view;
        DIS_mapping_view                *previous_mapping_view;
};

DIS_implementation_model () { }
```

FINAL REPORT

APPENDIX E

ADA SPECIFICATIONS FOR REPRESENTING SYSTEM DESIGN FACTOR

SYSTEM ENGINEERING AUTOMATION (SEA)
FOR DISTRIBUTED SYSTEMS

CONTRACT NO. N00014-91-C-0183
CDRL SEQUENCE NO. A002

AUGUST 1992

```
with Text_io;
use  Text_io;

package DIS_system_design_factor is

        —============================================================—
         —        DIS DECLARATIONS - SYSTEM DESIGN FACTOR
        —============================================================—

        MAX_LENGTH : INTEGER := 100;
        — Should be set to the desirable number.

         — To be determined and assigned appropriate type.
         — This is only the compilation purpose.
         subtype    TBD                                   is  INTEGER;  ;
         subtype    DIS_Rational                          is TBD;
         subtype    DIS_MorP                              is TBD;
         subtype    DIS_Reference                         is TBD;
         subtype    DIS_Definition                        is TBD;
         subtype    DIS_Comments                          is TBD;
         subtype    DIS_SDF                               is TBD;
         subtype    DIS_Ranges                            is TBD;
         subtype    DIS_Units                             is TBD;
         subtype    DIS_Qnty                              is TBD;
         subtype    DIS_Aggregate                         is TBD;
         subtype    DIS_Variable                          is TBD;
         subtype    DIS_CAggregate                        is TBD;
         subtype    DIS_CType                             is TBD;
         subtype    DIS_CView                             is TBD;
         subtype    DIS_CDFactor                          is TBD;
         subtype    DIS_CComponent                        is TBD;
         subtype    DIS_DOF                               is TBD;
         subtype    DIS_ReliabilityScale                  is TBD;
         subtype    DIS_PerformanceScale                  is TBD;
         subtype    DIS_SecurityScale                     is TBD;
         subtype    DIS_HumanFactorScale                  is TBD;
         subtype    FIELDS                                is string(1..MAX_LENGTH);
```

```
—/*——————————————————————————————————————————————————————————————*/
—/*              SYSTEM  DEIGN  FACTOR  TEMPLATE                  */
—/*                                                              */
—/* This  System  Design  Factor(SDF)  is  to  optimize  the  design */
—/* to  meet  the  requiremments  and  desired  measure  of         */
—/* effectiveness.  The  design  goals  and  criteria  in  this  SDF */
—/* are  specified  by  the  system  designers  and  analysisists  to */
—/* qualify  the  various  aspects  of  the  design  and  to  perform */
—/* the  trade–offs  among  different  design  goals.  Respect  to  */
—/* the  type  of  the  system,  it  describes  the  properties,     */
—/* attributes  and  characteristics  of  the  system.   Each  SDF   */
—/* must  have  its  own  merit  to  gauge  every  detail  of  the   */
—/* system.  This  merit  describes  the  weakness  and  strengths   */
—/* of  a  specific  area  in  the  design.  In  turn,  the          */
—/* correlation  of  the  SDF  characterizes  the  completeness  and */
—/* robustness.                                                  */
—/*——————————————————————————————————————————————————————————————*/


    type DIS_SDF_Template;
    type DIS_SDF_Template_ptr is access DIS_SDF_Template;
         — This is an open declaration for System Design Factor
         — Template type to make a pointer to the dynamic
         — storage location in memory.

    type DIS_Attribute;
    type DIS_Attribute_ptr is access DIS_Attribute;
         — This is an open declaration for System Design Factor
         — Attribute type to make a pointer to the dynamic
         — storage location in memory.

    type DIS_Quantification;
    type DIS_Quantification_ptr is access DIS_Quantification;
         — This is an open declaration for System Design Factor
         — Quantification type to make a pointer to the dynamic
         — storage location in memory.

    type DIS_Consistency;
    type DIS_Consistency_ptr is access DIS_Consistency;
         — This is an open declaration for System Design Factor
         — Consistency type to make a pointer to the dynamic
         — storage location in memory.

    type DIS_QualityReq;
    type DIS_QualityReq_ptr is access DIS_QualityReq;
    — This is an open declaration for System Design Factor
    — QualityReq type to make a pointer to the dynamic
    — storage location in memory.

type DIS_SDF_Template is
      record
           Name(MAX_LENGTH)              : FIELDS;
           Attributes                    : DIS_Attribute_ptr;
           Rational                      : DIS_Rational;
           Method_Or_Principle           : DIS_MorP;
           Quantification                : DIS_Quantify_ptr;
           Consistency                   : DIS_Consistency_ptr;
           QualityRequirements           : DIS_QualityReq_ptr;
           ReferenceList                 : DIS_Reference;
           Definitions                   : DIS_Definition;
           Annotations                   : DIS_Comments;
           NextTemplate                  : DIS_SDF_Template_ptr;
           PreviousTemplate              : DIS_SDF_Template_ptr;
end  record;

type DIS_Properties;
type DIS_Properties_ptr is access DIS_Properties;
      — This is an open declaration for property type of
      — SDF attribute to make a pointer to the dynamic
      — storage location in memory.

type DIS_RelationShip is (Functional, Logical);
      — This is a relationship type for SDF attribute.

type DIS_Attributes is
      record
      RelationShip                       : DIS_RelationShip;
      Properties                         : DIS_Properties_ptr;
      NextAttribute                      : DIS_Attribute_ptr;
end  record;

type DIS_Properties is
      record
      type                               : DIS_SDF;
      Ranges                             : DIS_Ranges;
      Units                              : DIS_Units;
end  record;

type DIS_Formula;
type DIS_Formula_ptr is access DIS_Formula;
```

```
        — This is an open declaration for formula type of
        — SDF qualify to make a pointer to the dynamic
        — storage location in memory.

  type DIS_Quantify is
        record
        type                              : DIS_Qnty;
        Formula                           : DIS_Formula_ptr;
  end  record;

  type DIS_Formula is
       record
        Aggregates                        : DIS_Aggregate;
        VariableList                      : DIS_Variable;
        NextFormula                       : DIS_Formula_ptr;
        PreviousFormula                   : DIS_Formula_ptr;
  end  record;

  type DIS_Consistency is
       record
            Aggregates                    : DIS_CAggregate;
            ByType                        : DIS_CType;
            ByView                        : DIS_CView;
            ByDesignFactor                : DIS_CDFactor;
            ByThisComponent               : DIS_CComponent;
  end  record;

 type DIS_Usability;
 type DIS_Usability_ptr is access DIS_Usability;
      — This is an open declaration for usability type of
      — SDF quality requirement to make a pointer to the dynamic
      — storage location in memory.

  type DIS_QualityReq is
        record
        Degree_Of_Functionality           : DIS_DOF;
        Usability                         : DIS_Usability_ptr;
  end  record;
```

```
          type DIS_Usability is
                record
                Reliability                    : DIS_ReliabilityScale;
                Performance                    : DIS_PerformanceScale;
                Security                       : DIS_SecurityScale;
                HumanFactors                   : DIS_HumanFactorScale;
          end  record;
end DIS_system_design_factor;
```

FINAL REPORT

APPENDIX F

C++ SPECIFICATIONS FOR REPRESENTING SYSTEM DESIGN FACTOR

SYSTEM ENGINEERING AUTOMATION (SEA) FOR DISTRIBUTED SYSTEMS

CONTRACT NO. N00014–91–C–0183
CDRL SEQUENCE NO. A002

AUGUST 1992

```
#include <stdio.h>
#include <string.h>

/*=====================================================================
//                    DIS DECLARATIONS - SYSTEM DESIGN FACTOR
=====================================================================*/

// Should be set to the desired no.
int  MAX_LENGTH  =  100;

// To be determined and assigned appropriate type.
// This is only the compilation purpose.
typedef  int  TBD;
typedef TBD DIS_Rational;
typedef TBD DIS_MorP;
typedef TBD DIS_Reference;
typedef TBD DIS_Definition;
typedef TBD DIS_Comments;
typedef TBD DIS_SDF;
typedef TBD DIS_Ranges;
typedef TBD DIS_Units;
typedef TBD DIS_Qnty;
typedef TBD DIS_Aggregate;
typedef TBD DIS_Variable;
typedef TBD DIS_CAggregate;
typedef TBD DIS_CType;
typedef TBD DIS_CView;
typedef TBD DIS_CDFactor;
typedef TBD DIS_CComponent;
typedef TBD DIS_DOF;
typedef TBD DIS_ReliabilityScale;
typedef TBD DIS_PerformanceScale;
typedef TBD DIS_SecurityScale;
typedef TBD DIS_HumanFactorScale;

// String class for names of the structures
class  String  {public:  String()  { };    // constructor
                private: int len; char *str;
};
```

```
/*-----------------------------------------------------------------*/
/*               SYSTEM DEIGN FACTOR TEMPLATE                      */
/*                                                                 */
/* This System Design Factor(SDF) is to optimize the design        */
/* to meet the requiremments and desired measure of                */
/* effectiveness. The design goals and criteria in this SDF        */
/* are specified by the system designers and analysisists to       */
/* qualify the various aspects of the design and to perform        */
/* the trade-offs among different design goals. Respect to         */
/* the type of the system, it describes the properties,            */
/* attributes and characteristics of the system.  Each SDF         */
/* must have its own merit to gauge every detail of the            */
/* system. This merit describes the weakness and strengths         */
/* of a specific area in the design. In turn, the                  */
/* correlation of the SDF characterizes the completeness and       */
/* robustness.                                                     */
/*-----------------------------------------------------------------*/

class DIS_Attribute;
        // This is an open declaration for System Design Factor
        // Attribute type to make a pointer to the dynamic
        // storage location in memory.

class DIS_Quantify;
        // This is an open declaration for System Design Factor
        // Quantification type to make a pointer to the dynamic
        // storage location in memory.

class DIS_Consistency;
        // This is an open declaration for System Design Factor
        // Consistency type to make a pointer to the dynamic
        // storage location in memory.

class DIS_QualityReq;
        // This is an open declaration for System Design Factor
        // QualityReq type to make a pointer to the dynamic
        // storage location in memory.

class DIS_SDF_Template {
        String                          Name(MAX_LENGTH);
        DIS_Attribute                   *Attributes;
        DIS_Rational                    Rational;
        DIS_MorP                        Method_Or_Principle;
        DIS_Quantify                            *Quantification;
        DIS_Consistency                         *Consistency;
        DIS_QualityReq                          *QualityRequirements;
        DIS_Reference                           ReferenceList;
        DIS_Definition                          Definitions;
        DIS_Comments                            Annotations;
        DIS_SDF_Template                        *NextTemplate;
};

class DIS_Properties;
        // This is an open declaration for property type of
        // SDF attribute to make a pointer to the dynamic
        // storage location in memory.

enum DIS_RelationShip       {Functional, Logical};
        // This is a relationship type for SDF attribute.

class DIS_Attributes        {
        DIS_RelationShip                        RelationShip;
        DIS_Properties                          *Properties;
        DIS_Attribute                           *NextAttribute;
};

class DIS_Properties        {
        DIS_SDF                                 type;
        DIS_Ranges                              Ranges;
        DIS_Units                               Units;
};

class DIS_Formula;
        // This is an open declaration for formula type of
        // SDF qualify to make a pointer to the dynamic
        // storage location in memory.

class DIS_Quantify          {
        DIS_Qnty                                type;
        DIS_Formula                             *Formula;
};
```

```
class DIS_Formula        {
        DIS_Aggregate                   Aggregates;
        DIS_Variable                    VariableList;
        DIS_Formula                     *NextFormula;
};

class DIS_Consistency        {
        DIS_CAggregate                  Aggregates;
        DIS_CType                       ByType;
        DIS_CView                       ByView;
        DIS_CDFactor                    ByDesignFactor;
        DIS_CComponent                  ByThisComponent;
};

class DIS_Usability        ;
        // This is an open declaration for usability type of
        // SDF quality requirement to make a pointer to the dynamic
        // storage location in memory.

class DIS_QualityReq        {
        DIS_DOF                         Degree_Of_Functionality;
        DIS_Usability                   Usability;
};

class DIS_Usability        {
        DIS_ReliabilityScale            Reliability;
        DIS_PertormanceScale            Performance;
        DIS_SecurityScale               Security;
        DIS_HumanFactorScale            HumanFactors;
};
```

FINAL REPORT

APPENDIX G

ROUTINES SUPPORTING DESTINATION INTERFACE SPECIFICATION

SYSTEM ENGINEERING AUTOMATION (SEA)
FOR DISTRIBUTED SYSTEMS

CONTRACT NO. N00014–91–C–0183
CDRL SEQUENCE NO. A002

AUGUST 1992

Appendix G

This Appendix contains the following Ada packages:

1. DIS_twk_mapper
2. DIS_twk_ispec
3. Write_DIS_file
4. DIS_resource_actions
5. DIS_timing_constraint_actions
6. DIS_placement_constraint_actions
7. DIS_sdf_actions
8. DIS_resource_support
9. DIS_placement_constraint_support
10. DIS_timing_constraint_support
11. DIS_sdf_template_support
12. DIS_files
13. DIS_tae_patch
14. DIS_twk_etd
15. DIS_twk_intf
16. DIS_twk_constraint

G1. DIS_twk_mapper

```
with   text_io;                                   use  text_io;
with   DIS_twk_Intf;                              use  DIS_twk_Intf;
with   DIS_imp_model;                             use  DIS_imp_model;

package DIS_twk_mapper is

   MAX_NO_EDGES                                   : INTEGER := 200;
   DIS_SW_Root                                    : DIS_sw_structure_diagram_ptr;
   DIS_SW_SDiagram                                : DIS_sw_structure_diagram_ptr;
   DIS_SW_Modules                                 : DIS_sw_module_ptr;
   DIS_RPtr                                       : DIS_sw_structure_diagram_ptr;

   type EdgeArray is array(1..MAX_NO_EDGES) of DIS_twk_flows;

   procedure  MapSWSDiagram;
   procedure  MapSWModules;
   procedure  LinkSubModules;
   procedure  MapModuleEdges;
   procedure  LinkTasks;
   procedure  LinkTaskEdges;

   procedure   LinkModuleEdges(PSWPtr             : in  DIS_sw_structure_diagram_ptr;
                               NPtr               : in  out DIS_sw_module_ptr;
                               DfdId              : in  INTEGER;
                               Nedges             : in  EdgeArray);

   procedure   GetModuleEdges(DfdId               : in  INTEGER;
                              NId                 : in  INTEGER;
                              Nedges              : in  out EdgeArray);

   procedure   Twk_GetParent(Index                : in  INTEGER;
                             DfdIndex,   Nid      : in  out INTEGER);

   procedure   AddToTask(MdlPtr                   : in  out DIS_sw_module_ptr);

   procedure   MapTaskEdges(EArray                : in  EdgeArray;
                            PPtr                  : in  out DIS_sw_module_ptr);

   procedure   MapEntrySEdges(EntrySEdges         : in  EdgeArray;
                              PPtr                : in  out DIS_sw_module_ptr);

   procedure   MapExitSEdges(ExitSEdges           : in  EdgeArray;
                                PPtr              : in  out DIS_sw_module_ptr);

   procedure   GetSWDMPtr(DfdId                   : in  INTEGER;
                          RetMPtr                 : out DIS_sw_module_ptr);

   procedure   GetSWDPtr(DfdId                    : in  INTEGER;
                         RetSDPtr                 : out DIS_sw_structure_diagram_ptr);
```

```
  procedure  GetMEPtr(NId                    : in INTEGER;
                           DfdId             : in INTEGER;
                           RetMEPtr          : out DIS_sw_module_ptr);

   procedure  GetParentNode(p_nid, dfd_id    : in INTEGER;
                           RetPNPtr          : out DIS_sw_module_ptr);

   procedure  GetNodePtr(EId                 : in INTEGER;
                           NPtr              : in DIS_sw_module_ptr;
                           RetNPtr           : out DIS_sw_task_node_ptr);

   function  EntrySuperEdge(TWKEdges         : in DIS_twk_flow_ptr;
                           DfdPtr            : in DIS_sw_module_ptr)
                           return BOOLEAN;

   function  ExitSuperEdge(TWKEdges          : in DIS_twk_flow_ptr;
                           DfdPtr            : in DIS_sw_module_ptr)
                           return BOOLEAN;

   function  TaskEdges(TWKEdge               : in DIS_twk_flow_ptr;
                           PNPtr             : in DIS_sw_module_ptr)
                           return BOOLEAN;

end DIS_twk_mapper;
```

G2. DIS_twk_ispec

```
with  io_exceptions;                                  use  io_exceptions;
with  Twk_ada_dba;                                    use  Twk_ada_dba;
with  Math;                                           use  Math;
with  DIS_twk_const;                                  use  DIS_twk_const;
with  DIS_twk_intf;                                   use DIS_twk_intf;
with Text_io; use text_io;

package DIS_twk_ispec is

      Status                                          :  Twk_status_t ;
      Pi                                              :  twk_object_ptr_t (Twk_process_index_type);
                                                      — /* process index ptr for the model */
      DDi                                             : twk_object_ptr_t(twk_dd_index_type);
                                                      — /* dd index ptr for the model */
      procedure   Initialize   (Path                  : String);

      function    Read_process_index(model_name             : String)
                  return Twk_process_index_t_ptr;

      function    Read_latest_dfd(model_name,  Dfd_name : String)
                  return Twk_dfd_t_ptr;

      procedure   Get_bubbles_in_one_dfd(Dfd                : in Twk_dfd_t_ptr;
                                        bubble_count        : in out Integer);

      procedure   Get_bubbles_in_all_dfds(model_name        : in STRING;
                                        Pi                  : Twk_process_index_t_ptr);

      procedure   Get_flow_in_one_dfd(Dfd : in Twk_dfd_t_ptr;
                                    data_flow_count         : in out Integer;
                                    df_Index                : in out Integer);

      procedure   Get_flow_in_all_dfd(model_name            : in STRING;
                                    Pi                      : in Twk_process_index_t_ptr);

      procedure   get_twk_nodes(model_name                  : in String;
                              config_name                   : in String);

      procedure   get_twk_flows(model_name                  : in String;
                              config_name                              : in String);

      procedure   clean_name(instr                          : in out string);

      Exit_failure                                          : exception;

 end DIS_twk_ispec;
```

G3. Write_DIS_file

```
with  text_io;                  use  text_io;
with  DIS_files;                use  DIS_files;
with  DIS_imp_model;            use  DIS_imp_model;
with  DIS_twk_const;            use  DIS_twk_const;

package write_DIS_files is

        procedure write_DIS_resource_file
          (resource : in out DIS_resource_ptr);

        procedure write_DIS_data_attribute_file
           (data_attribute : in out DIS_data_attribute_ptr);

        procedure write_DIS_sw_task_edge_file
           (sw_task_edge : in out DIS_sw_task_edge_ptr);

        procedure write_DIS_sw_task_node_file
           (sw_task_node : in out DIS_sw_task_node_ptr);

        procedure write_DIS_sw_module_edge_file
           (sw_module_edge : in out DIS_sw_module_edge_ptr);

        procedure write_DIS_sw_module_file
           (sw_module : in out DIS_sw_module_ptr);

        procedure write_DIS_sw_structure_diagram_file
           (DIS_SW_Root: in out DIS_sw_structure_diagram_ptr);

        procedure write_DIS_hw_link_file
           (hw_link : in out  DIS_hw_link_ptr);

        procedure write_DIS_hw_node_file
           (hw_node : in out DIS_hw_node_ptr);

        procedure write_DIS_hw_group_link_file
           (hw_group_link : in out DIS_hw_group_link_ptr);

        procedure write_DIS_hw_group_node_file
           (hw_group_node : in out DIS_hw_group_node_ptr);

        procedure write_DIS_hw_structure_diagram_file
           (hw_struc_diag : in out DIS_hw_structure_diagram_ptr);

        procedure set_DIS_file_pointers;

        procedure close_DIS_files;

end write_DIS_files;
```

G4. DIS_resource_actions

```
with  text_io;                                   use  text_io;
with  DIS_files;                                 use  DIS_files;
with  unix;                                      use  unix;
with  DIS_imp_model;                             use  DIS_imp_model;

package  DIS_resource_actions  is

          MAX_RECORD_LENGTH                      : INTEGER := 1635;
          Resource                               : DIS_resource_ptr;
          FoundFlag                              : Boolean;

          procedure  write_RES_file(N_Id         : in  STRING);
          procedure  read_RES_file(N_Id          : in  STRING);
          procedure  read_write_RES_files(task_id : in  STRING);

end  DIS_resource_actions;
```

G5. DIS_timing_constraint_actions

```
with  text_io;                              use  text_io;
with  DIS_files;                            use  DIS_files;
with  unix;                                 use  unix;
with  DIS_imp_model;                        use  DIS_imp_model;

package DIS_timing_constraint_actions is

                    Time_Constraint         : DIS_time_constraint_ptr;
                    FoundFlag               : BOOLEAN;

          procedure   write_TC_file(N_Id : in STRING);
          procedure   read_TC_file   (N_Id  in STRING);

end DIS_timing_constraint_actions;
```

G6. DIS_placement_constraint_actions

```
with  text_io;‘                               use  text_io;
with  DIS_files;                              use  DIS_files;
with  unix;                                   use  unix;
with  DIS_imp_model;                          use  DIS_imp_model;

package  DIS_placement_constraint_actions  is

          Place_Constraint                    : DIS_place_constraint_ptr;
          FoundFlag                           : BOOLEAN;

          procedure   write_PC_file(N_Id      : in  STRING);
          procedure   read_PC_file   (N_Id    : in  STRING);

end  DIS_placement_constraint_actions;
```

G7. DIS_sdf_actions

```
with  text_io;                                use  text_io;
with  DIS_files;                              use  DIS_files;
with  DIS_twk_const;                          use  DIS_twk_const;
with  unix;                                   use  unix;

package DIS_sdf_actions is

          MAX_RECORD_LENGTH                   : INTEGER := 1268;

          type DIS_SDF_Template is
               record
                    Node_id                   : INTEGER;
                    Factor                    : INTEGER;
                    sUBfACTor                 : INTEGER;
                    Name                      : STRING(1..80);
                    Type_T                    : STRING(1..20);
                    Range_T                   : STRING(1..20);
                    Units                     : STRING(1..20);
                    Priority                  : STRING(1..20);
                    Accuracy                  : STRING(1..20);
                    MethPrin                  : STRING(1..100);
                    Defn                      : STRING(1..100);
                    Rational                  : STRING(1..50);
                    Relationship              : STRING(1..50);
                    QuantType1                : STRING(1..10);
                    QuantValue1               : STRING(1..10);
                    QuantFormula1             : STRING(1..50);
                    QuantType2                : STRING(1..10);
                    QuantValue2               : STRING(1..10);
                    QuantFormula2             : STRING(1..50);
                    QuantType3                : STRING(1..10);
                    QuantValue3               : STRING(1..10);
                    QuantFormula3             : STRING(1..50);
                    QuantType4                : STRING(1..10);
                    QuantValue4               : STRING(1..10);
                    QuantFormula4             : STRING(1..50);
                    QuantType5                : STRING(1..10);
                    QuantValue5               : STRING(1..10);
                    QuantFormula5             : STRING(1..50);
                    Con_Type1                 : STRING(1..15);
                    Con_valu1                 : STRING(1..10);
                    Con_form1                 : STRING(1..50);
                    Con_Type2                 : STRING(1..15);
                    Con_valu2                 : STRING(1..10);
                    Con_form2                 : STRING(1..50);
                    Con_Type3                 : STRING(1..15);
```

```
                Con_valu3                     : STRING(1..10);
                Con_form3                     : STRING(1..50);
                Con_Type4                     : STRING(1..15);
                Con_valu4                     : STRING(1..10);
                Con_form4                     : STRING(1..50);
                Con_Type5                     : STRING(1..15);
                Con_valu5                     : STRING(1..10);
                Con_form5                     : STRING(1..50);
                Annotation                    : STRING(1..50);
        end record;

        OK_Button                             : BOOLEAN := FALSE;
        NodeName_I                            : INTEGER := 10;

        DISSDF_Template                       : DIS_SDF_Template;
        FoundFlag                             : BOOLEAN;

        procedure  Read_SDF_File(Node_No      : in STRING;
                                 Fid          : in INTEGER;
                                 SFid         : in INTEGER);

        procedure  Write_SDF_File(Node_No     : in STRING;
                                  Fid         : in INTEGER;
                                  SFid        : in INTEGER);
end DIS_sdf_actions;
```

G8. DIS_resource_support

```
with  tae;                                use  tae;
with X_Windows;
with text_io;
with  DIS_reource_actions;                use  DIS_resource_actions;
with  DIS_imp_model;                      use  DIS_imp_model;
with  DIS_files;                          use  DIS_files;
with  DIS_twk_const;                      use  DIS_twk_const;
with  COMMAND_LINE;                       use  COMMAND_LINE;

package DIS_resource_support is

    package taefloat_io is new text_io.float_io (taefloat);
    procedure initializePanels (file : in string);  — NOTE: params changed

      — BEGIN EVENT_HANDLERs
      procedure  DIS_RES_RES_CANCEL       (info   : in tae_wpt.event_context_p:r);
      procedure  DIS_RES_RES_CLOSE        (info   : in tae_wpt.event_context_ptr);
      procedure  DIS_RES_RES_OK           (info   : in tae_wpt.event_context_ptr);
      procedure  DIS_RES_RES_PREV         (info   : in tae_wpt.event_context_ptr);
      procedure  DIS_RES_RES_NEXT         (info   : in tae_wpt.event_context_ptr);
      procedure  DIS_RES_RES_NAME         (info   : in tae_wpt.event_context_ptr);
      procedure  DIS_RES_RES_UNIT_TYPE    (info   : in tae_wpt.event_context_ptr);
      procedure  DIS_RES_RES_UNIT_AMNT    (info   : in tae_wpt.event_context_ptr);
      procedure  DIS_RES_RES_TYPE         (info   : in tae_wpt.event_context_ptr);
      — END EVENT_HANDLERs

end DIS_resource_support;
```

G9. DIS_placement_constraint_support

```
with  tae;                                   use tae;
with  X_Windows;
with  text_io;
with  DIS_imp_model;                         use   DIS_imp_model;
with  DIS_placement_constraint_actions;      use   DIS_placement_constraint_actions;
with  COMMAND_LINE;                          use   COMMAND_LINE;
with  DIS_files;                             use   DIS_files;
with  DIS_twk_const;                         USE   dis_twk_const;

package DIS_placement_constraint_support is

    package taefloat_io is new text_io.float_io (taefloat);
    procedure initializePanels (file : in string);  — NOTE: params changed

    — BEGIN EVENT_HANDLERs
        procedure  DIS_PC_PC_HW          (info : in tae_wpt.event_context_ptr);
        procedure  DIS_PC_PC_KIND        (info : in tae_wpt.event_context_ptr);
        procedure  DIS_PC_PC_OK          (info : in tae_wpt.event_context_ptr);
        procedure  DIS_PC_PC_CANCEL      (info : in tae_wpt.event_context_ptr);
        procedure  DIS_PC_PC_TASK_LIST   (info : in tae_wpt.event_context_ptr);
        procedure  DIS_PC_PC_PREF_VAL    (info : in tae_wpt.event_context_ptr);
        procedure  DIS_PC_PC_HELP        (info : in tae_wpt.event_context_ptr);
    — END EVENT_HANDLERs

end DIS_placement_constraint_support;
```

G10. DIS_tming_constraint_support

```
with   tae;                              use  tae;
with   X_Windows;
with   text_io;
with   DIS_imp_model;                    use   DIS_imp_model;
with   TC_actions;                       use   TC_actions;
with   DIS_files;                        use   DIS_files;
with   DIS_twk_const;                    use   DIS_twk_const;
with   COMMAND_LINE;                     use   COMMAND_LINE;

package DIS_timing_constraint_support is

    package taefloat_io is new text_io.float_io (taefloat);
    procedure initializePanels (file : in string);   — NOTE: params changed

    — BEGIN EVENT_HANDLERs
        procedure  DIS_TC_TC_OK              (info : in tae_wpt.event_context_ptr);
        procedure  DIS_TC_TC_CANCEL          (info : in tae_wpt.event_context_ptr);
        procedure  DIS_TC_TC_PREF_VAL        (info : in tae_wpt.event_context_ptr);
        procedure  DIS_TC_TC_TM_TYPE         (info : in tae_wpt.event_context_ptr);
        procedure  DIS_TC_TC_TM_UNIT_AMNT(info : in tae_wpt.event_context_ptr);
        procedure  DIS_TC_TC_TASK_LIST       (info : in tae_wpt.event_context_ptr);
        procedure  DIS_TC_TC_KIND            (info : in tae_wpt.event_context_ptr);
        procedure  DIS_TC_TC_CLOSE           (info : in tae_wpt.event_context_ptr);
    — END EVENT_HANDLERs

end DIS_timing_constraint_support;
```

G11. DIS_sdf_template_support

```
with tae; use tae;
with X_Windows;
with text_io;
with  DIS_files;                      use DIS_files;
with  DIS_tae_patch;                  use DIS_tae_patch;
with  DIS_sdf_actions;                use DIS_sdf_actions;
with  COMMAND_LINE;                   use COMMAND_LINE;
with  DIS_twk_const;                  use DIS_twk_const;

package DIS_sdf_template_support is

    package taefloat_io is new text_io.float_io (taefloat);
    procedure initializePanels (file : in string);  — NOTE: params changed

    — BEGIN EVENT_HANDLERs
        procedure  sdf_tp_sdf_pfm       (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_tp_sdf_rtm       (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_tp_sdf_dep       (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_tp_sdf_hmw       (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_tp_sdf_phq       (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_tp_sdf_sec       (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_tp_sdf_trq       (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_tp_sdf_lcy       (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_tp_sdf_frq       (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_tp_sdf_prq       (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_tp_sdf_fnc       (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_tp_tp_ok         (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_tp_tp_cl         (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_tp_tp_hp         (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_pf_pf_rst        (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_pf_pf_cpy        (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_pf_pf_spd        (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_pf_pf_tpt        (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_pf_pf_lcy        (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_pf_pf_lbc        (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_pf_pf_gdg        (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_pf_pf_pty        (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_pf_pf_ok         (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_pf_pf_cl         (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_pf_pf_hp         (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_rt_rt_hns        (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_rt_rt_hdl        (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_rt_rt_sdl        (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_rt_rt_tdt        (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_rt_rt_mdl        (info  : in tae_wpt.event_context_ptr);
        procedure  sdf_rt_rt_tns        (info  : in tae_wpt.event_context_ptr);
procedure  sdf_rt_rt_pty        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_rt_rt_gdt        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_rt_rt_ok         (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_rt_rt_cl         (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_rt_rt_hp         (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_pr_pr_imp        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_pr_pr_uns        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_pr_pr_pty        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_pr_pr_poy        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_pr_pr_irc        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_pr_pr_msp        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_pr_pr_ok         (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_pr_pr_cl         (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_pr_pr_hp         (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_dp_dp_rty        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_dp_dp_acy        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_dp_dp_ftn        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_dp_dp_gdt        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_dp_dp_rcy        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_dp_dp_aty        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_dp_dp_qty        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_dp_dp_ok         (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_dp_dp_cl         (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_dp_dp_hp         (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_st_st_clt        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_st_st_typ        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_st_st_ptm        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_st_st_ety        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_st_st_irq        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_st_st_ok         (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_st_st_cl         (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_st_st_hp         (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_tmp_tmp_cl       (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_tmp_tmp_o        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_tmp_tmp_h        (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_tmp_tmp_md       (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_tmp_tmp_nd       (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_tmp_tmp_ty       (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_tmp_tmp_rg       (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_tmp_tmp_ut       (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_tmp_tmp_rl       (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_tmp_tmp_rt       (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_tmp_tmp_at       (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_tmp_tmp_pr       (info  : in  tae_wpt.event_context_ptr);
procedure  sdf_tmp_tmp_ac           (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_qt       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_f1       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_v1       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_t1       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_cr       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_fo       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_vo       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_NoName23     (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_NoName24     (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_ct1      (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_df       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_f2       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_f3       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_f4       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_f5       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_v2       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_v3       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_v4       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_v5       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_t2       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_t3       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_t4       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_t5       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_c1       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_c2       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_c3       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmP_c4       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_c5       (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_cv1      (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_cv2      (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_cv3      (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_cv4      (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_cv5      (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_ct2      (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_ct3      (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_ct4      (info : in tae_wpt.event_context_ptr);
    procedure  sdf_tmp_tmp_ct5      (info : in tae_wpt.event_context_ptr);
    procedure  tmp_QTp_qt_ok        (info : in tae_wpt.event_context_ptr);
    procedure  tmp_QTp_NoName01     (info : in tae_wpt.event_context_ptr);
    procedure  tmp_QTp_qtp_rd       (info : in tae_wpt.event_context_ptr);
    procedure  tmp_CTp_CTp_ok       (info : in tae_wpt.event_context_ptr);
    procedure  tmp_CTp_CTp_cl       (info : in tae_wpt.event_context_ptr);
    procedure  tmp_CTp_CTp_tp       (info : in tae_wpt.event_context_ptr);
— END EVENT_HANDLERs
```

G12. DIS_files

```
with text_io;       use text_io;

package DIS_files is

          subtype FName_Type is       STRING(1..100);
          Model_Name                  : FName_Type;
          Model_NLength               : NATURAL;
          Target_Dir                  : FName_Type;
          Target_DLength              : NATURAL;
          Config_FName                : STRING(1..128);
          Config_FLength              : INTEGER;
          File_NLength                : INTEGER;
          Temp_FName                  : FName_Type;
          Temp_SDF_FName              : constant STRING := ”DIS_Temp_SDFile”;
          DIS_SDF_FName               : constant STRING := ”_sdf”;

          function  Get_File_Name(Instr : in STRING) return STRING;

          function  Open_File(File_Name: in STRING;
                              FMode    : in text_io.file_mode)
                              return text_io.file_type;

end DIS_files;
```

G13. DIS_tae_patch

```
with  tae;                                   use  tae;
with  DIS_actions;                           use  DIS_actions;
with  text_io;                               use  text_io;


package  DIS_tae_patch  is

          procedure  ReadWrite   (PnlPtr   : in  out  tae_wpt.event_context_ptr);
          procedure  InitPnl     (PnlPtr   : in  out  tae_wpt.event_context_ptr);
          procedure  DepressBtn  (PnlPtr   : in  tae_wpt.event_context_ptr;
                                  ItemId   : in  string);

end  DIS_tae_patch;
```

G14. DIS_twk_etd

```
with  text_io;                          use text_io;
with  COMMAND_LINE;                     use COMMAND_LINE;
with  DIS_twk_const;                    use DIS_twk_const;

with  DIS_twk_ispec;                    use  DIS_twk_ispec;
with  DIS_twk_mapper;                   use DIS_twk_mapper;
with  write_EDA_files;                  use write_EDA_files


package DIS_twk_etd is
begin

          model_name(1..argv(1).s'last)     := argv(1).s;
          model_nlength := argv(1).s'last;

          target_dir(1..17)                 :="../export_import/";
          target_Dlength                    := 17;
          t_length                          := 17;

          config_name   (1..argv(2).s/last) := argv  (2)  .s;
          c_last                            := argv  (2).s'last;

          get_twk_nodes(model_name, config_name);
          get_twk_flows(model_name, config_name);

          MapSWSDiagram;
          MAPSWModules'
          LinkSubModules;
          MapModuleEdges;
          LinkTasks;
          LinkTaskEdges;

          get_EDA_file_pointers;
          write_EDA_sw_structure_diagram_file  (DIS_SW_Root);
          close_EDA_files;

end DIS_twk_etd;
```

G15. DIS_twk_intf

```
package DIS_twk_intf is

    type DIS_twk_node;
    type DIS_twk_node_ptr is access DIS_twk_node;
    type DIS_twk_node is
        record
         node_id              :  INTEGER;        — Instance #.
         node_name            :  STRING(1..100); — Text bound to the node.
         node_no              :  INTEGER;        — Holds good only for bubbles (not stores).
         node_type            :  INTEGER;        — 1–process, 2–store, 3–others.
         next_node            :  DIS_twk_node_ptr;
        end record;

    type DIS_twk_flows;
    type DIS_twk_flow_ptr is access DIS_twk_flows;
    type DIS_twk_flows is
        record
         flow_id              :  INTEGER;        — Instance #.
         flow_name            :  STRING(1..100); — Text bound to the flow.
         flow_type            :  INTEGER;        — 1 – data, 2 – Control.
         flow_start           :  INTEGER;        — Instance # of the flow st pt.
         flow_start_obj       :  INTEGER;        — 1–process, 2–store, 3–others.
         flow_end             :  INTEGER;        — Instance # of the flow end pt.
         flow_end_obj         :  INTEGER;        — 1–process, 2–store, 3–others.
         next_flow            : DIS_twk_flow_ptr;
        end record;

      type twk_node_type is array(INTEGER range <>) of DIS_twk_node_ptr;
      type twk_flow_type is array(INTEGER range <>) of DIS_twk_flow_ptr;
    ———————————————— twk extract variables ————————————————

      type DIS_twk_dfd_nodes is array(INTEGER range <>) of DIS_twk_node_ptr;
      type DIS_twk_dfd_edges is array(INTEGER range <>) of DIS_twk_flow_ptr;

         DIS_MAX_LAYERS      : INTEGER := 50;
         DIS_NAME_LENGTH     : INTEGER := 100;
         DIS_twk_CLayers     : INTEGER := 0;
         DIS_twk_CELayers    : INTEGER := 0;
         DIS_twk_NLayers     : INTEGER := –1;

         twk_nodes           : DIS_twk_node_ptr;
         twk_edges           : DIS_twk_flow_ptr;
         twk_rootnodes       : DIS_twk_node_ptr;
         DIS_twk_layers_nd   : DIS_twk_dfd_nodes(0..DIS_MAX_LAYERS);
         DIS_twk_layers_ed   : DIS_twk_dfd_edges(0..DIS_MAX_LAYERS);
         DIS_twk_LNames      : array(0..DIS_MAX_LAYERS) of
                              STRING(1..DIS_NAME_LENGTH);
end DIS_twk_intf;
```

G16. DIS_twk_constraint

```
package DIS_twk_const is

  subtype FNAME_TYPE is STRING(1..100);
    model_name      : FNAME_TYPE;
    target_dir      : FNAME_TYPE;
    model_nlength   : NATURAL;
    t_length        : NATURAL;
    target_dlength  : NATURAL;
    config_name     : STRING(1..128);
    c_last          : INTEGER;
    c_default       : constant STRING := "/home/twk401/cadre/tsa/config_file";

end DIS_twk_const;
```

FINAL REPORT

APPENDIX H

DESTINATION INTERFACE SPECIFICATION FILE FORMATS

SYSTEM ENGINEERING AUTOMATION (SEA)
FOR DISTRIBUTED SYSTEMS

CONTRACT NO. N00014–91–C–0183
CDRL SEQUENCE NO. A002

AUGUST 1992

Appendix H

This Appendix contains the following packages:

1. Software_structure_diagram_file
2. Software_module_file
3. Software_module_edge_file
4. Software_task_node_file
5. Software_task_edge_file
6. Data_atrribute_file
7. Hardware_structure_diagram_file
8. Hardware_group_node_file
9. Hardware_group_link_file
10. Hardware_node_file
11. Hardware_link_file
12. Resource_file
13. Timing_constraint_file
14. Placement_constraint_file
15. System_design_factor_template
16. System_design_factor_attribute
17. System_design_factor_quantification

```
******************************************************************************************
*                    SOFTWARE  STRUCTURE  DIAGRAM  FILE                                  *
******************************************************************************************
```

FILE NAME: software_structure_diagram_file

/FIELD	#/FIELD NAME	/DATA TYPE	/FIELD SIZE
F1	id	integer	10
F2	name	char	100
F3	parent_implementation_view	pointer	16
F4	sw_module_list	pointer	16
F5	sw_module_edge_list	pointer	16
F6	next_sw_diagram	pointer	16
F7	previous_sw_diagram	pointer	16

****

*   SOFTWARE MODULE FILE   *

****

FILe NAME: software_module_file

/FIELD #	/FIELD NAME	/DATA TYPE	/FIELD SIZE
_			
F1	id	integer	10
F2	name	char	100
F3	parent_sw_structure	pointer	16
F4	parent_module	pointer	16
F5	next_module	pointer	16
F6	previous_module	pointer	16
F7	submodule_list	pointer	16
F8	module_edge_list	pointer	16
F9	task_node_list	pointer	16
F10	task_edge_list	pointer	16
F11	entry_super_edge_list	pointer	16
F12	exit_super_edge_list	pointer	16
F13	module_sdf	pointer	16

```
****************************************************************************
*                    SOFTWARE  MODULE  EDGE  FILE                          *
****************************************************************************
```

FILE NAME: software_module_edge_file

/FIELD #	/FIELD NAME	/DATA TYPE	/FIELD SIZE
F1	id	integer	10
F2	name	char	100
F3	attributes	char	100
F4	parent_sw_structure	pointer	16
F5	from_mofule	pointer	16
F6	to_module	pointer	16
F7	super_edge_list	pointer	16
F8	next_module_edge	pointer	16
F9	previous_module_edge	pointer	16
F10	module_edge_sdf	pointer	16

****

*   SOFTWARE TASK NODE FILE   *

****

FILE NAME: software_task_node_file

/FIELD #	/FIELD NAME	/DATA TYPE	/FIELD SIZE
F1	id	integeer	10
F2	name	char	100
F3	parent_module	pointer	16
F4	task_structure	pointer	16
F5	task_description	pointer	16
F6	task_edge_list	pointer	16
F7	task_input_and_and_or	integer	4
F8	task_input_list	pointer	16
F9	task_out_and_or	integer	4
F10	task_output_list	pointer	16
F11	task_before_and_or	integer	4
F12	task_before_list	pointer	16
F13	task_after_and_or	integer	4
F14	task_after_list	pointer	16
F15	task_ready_time	integer	4
F16	task_deadline	integer	4
F17	task_duration	integer	4
F18	task_resource_needs	integer	16
F19	task_buddy_list	pointer	16

F20	task_max_replication	integer	4
F21	task_priority	integer	4
F22	task_execution_probability	integer	4
F23	task_communication_delay_matrix	integer	4
F24	next_task	pointer	16
F25	previous_task	pointer	16
F26	task_error_cumulation	integer	4
F27	task_imprecise_error_convergence	integer	4
F28	task_sdf	pointer	16

```
***************************************************************************
*                         SOFTWARE TASK EDGE FILE                         *
***************************************************************************
```

FILE NAME: software_task_edge_file

/FIELD #	/FIELD NAME	/DATA TYPE	/FIELD SIZE
F1	id	integer	10
F2	name	char	100
F3	parent_module	pointer	16
F4	parent_module_edge	pointer	16
F5	task_edge_data	pointer	16
F6	from_task_node	pointer	16
F7	to_task_node	pointer	16
F8	next_task_edge	pointer	16
F9	previous_task_edge	pointer	16
F10	task_edge_sdf	pointer	16

```
*******************************************************************************************
*                                    DATA   ATTRIBUTE   FILE                               *
*******************************************************************************************
```

FILE NAME: data_attribute_file

/FIELD #/FIELD NAME		DATE TYPE	SIZE IN BYTES
F1	id	integer	10
F2	name	char	100
F3	data_kind	integer	4
F4	task_edge_list	pointer	16
F5	sender_kind	integer	4
F6	data_sender_list	pointer	16
F7	receiver_kind	integer	4
F8	data_receiver_list	pointer	16
F9	data_resource_need_list	pointer	16
F10	data_deadline	integer	4
F11	data_frequency	integer	4

```
********************************************************************************
*                   HARDWARE  STRUCTURE  DIAGRAM  FILE                         *
********************************************************************************
```

FILE NAME: hardware_structure_diagram_file

/FIELD	#/FIELD NAME	/DATA TYPE	/SIZE IN BYTES
F1	id	integer	10
F2	name	char	100
F3	parent_implementation_view	pointer	16
F4	hw_group_node_list	pointer	16
F5	hw_group_link_list	pointer	16
F6	hw_group_link_topology	pointer	16
F7	next_hw_diagram	pointer	16
F8	previous_hw_diagram	pointer	16

****

*   HARDWARE GROUP NODE FILE   *

****

FILE NAME: hardware_group_node_file

/FIELD #	/FIELD NAME	/DATA TYPE	/SIZE IN BYTES
F1	id	integer	10
F2	name	char	100
F3	parent_hw_structure	pointer	16
F4	parent_hw_group_node	pointer	16
F5	next_hw_group_node	pointer	16
F6	previous_hw_group_node	pointer	16
F7	sub_hw_group_node_list	pointer	16
F8	sub_hw_group_link_list	pointer	16
F9	hw_group_link_topology	integer	4
F10	hw_node_list	pointer	16
F11	hw_link_list	pointer	16
F12	hw_node_link_topology	integer	4
F13	entry_super_lisk_list	pointer	16
F14	exit_super_link_list	pointer	16
F15	group_node_sdf	pointer	16

```
**************************************************************************************
*                           HARDWARE  GROUP  LINK  FILE                              *
**************************************************************************************
```

FILE NAME: hardware_group_link_file

/FIELD #	/FIELD NAME	/DATA TYPE	/SIZE IN BYTES
F1	id	integer	10
F2	name	char	100
F3	parent_hw_structure	pointer	16
F4	from_hw_group_node	pointer	16
F5	to_hw_group_node	pointer	16
F6	super_link_list	pointer	16
F7	next_hw_group_link	pointer	16
F8	previous_hw_group_link	pointer	16
F9	group_link_sdf	pointer	16

HARDWARE NODE FILE

FILE NAME: hardware_node_file

/FIELD #	/FIELD NAME	/DATA TYPE	/SIZE IN BYTES
F1	id	integer	10
F2	name	char	100
F3	hw_node_generic	integer	4
F4	hw_node_specific	integer	4
F5	hw_node_resource_available	pointer	16
F6	hw_link_list	pointer	16
F7	next_hw_node	pointer	16
F8	previous_hw_node	pointer	16
F9	internal_hw_node_list	pointer	16
F10	internal_hw_link_list	pointer	16
F11	parent_hw_group_node	pointer	16
F12	parent_hw_node	pointer	16
F13	hw_node_sdf	pointer	16

```
*************************************************************************
*                          HARDWARE  LINK  FILE                         *
*************************************************************************
```

FILE NAME: hardware_link_file

/FIELD #	/FIELD NAME	/DATA TYPE	/SIZE IN BYTES
F1	id	integer	10
F2	name	char	100
F3	generic	integer	4
F4	specific	integer	4
F5	data_rate	integer	4
F6	data_latency	integer	4
F7	protocol	integer	4
F8	parent_hw_group_node	pointer	16
F9	parent_hw_node	pointer	16
F10	parent_hw_group_link	pointer	16
F11	next_hw_link	pointer	16
F12	previous_hw_link	pointer	16
F13	hw_link_sdf	pointer	16

```
**********************************************************************
*                          RESOURCE  FILE                            *
**********************************************************************
```

FILE NAME: resource_file

/FIELD	#/FIELD NAME	DATA TYPE	SIZE IN BYTES
F1	id	*integer*	10
F2	name	char	100
F3	task_node_list	pointer	16
F4	task_edge_list	pointer	16
F5	hw_node_list	pointer	16
F6	next_resource_node	pointer	16
F7	previous_resource_node	pointer	16

```
******************************************************************************
*                         TIMING  CONSTRAINT  FILE                           *
******************************************************************************
```

FILE NAME: timing_constraint_file

FIELD #	/FILED NAME	/DATA TYPE	/DATA SYZE IN BYTES
F1	id	integer	10
F2	constraint_kind	integer	4
F3	preference_value	integer	4
F4	time_type	integer	4
F5	time_value	integer	4
F6	software_id_list	char	100
F7	parent_mapping_constraint	pointer	16
F8	next_constraint	pointer	16
F9	previous_constraint	pointer	16

****

*   PLACEMENT CONSTRAINT FILE   *

****

FILE NAME: placement_constraint_file

FIELD #	/FILED NAME	/DATA TYPE	/DATA SYZE IN BYTES
F1	id	integer	10
F2	constraint_kind	integer	4
F3	preference_value	integer	1
F4	hardware_id	char	100
F5	software_id_list	char	100
F6	parent_mapping_constraint	pointer	16
F7	next_constraint	pointer	16
F8	previous_constraint	pointer	16

```
***************************************************************************************
*                     SYSTEM   DESIGN   FACTOR   TEMPLATE                             *
***************************************************************************************
```

FILE NAME: sdf_template

FIELD	#/FIELD NAME	/DATE TYPE	SIZE IN BYTES
F1	id	integer	10
F2	name	char	128
F3	Attributes	pointer	16
F4	Rational	char	80
F5	Method_or_principle	char	240
F6	Quantification_type	integer	4
F7	Quantification_formula	pointer	16
F8	Consistency_aggregat	char	10
F9	Consistency_type	char	200
F10	Consistency_design_factor	char	20
F11	Consistency_view	char	20
F12	Consistency_component	char	20
F13	QualityReq_type	char	20
F14	QualityReq_Usability_Reliability	char	20
F15	QualityReq_Usability_Performance	char	20
F16	QualityReq_Usability_Security	char	20
F17	QualityReq_Usability_HumanFactors	char	20
F18	ReferenceList	char	240
F19	Definitions	char	240

F20	Annotations	char	240
F21	next_sdf_template	pointer	16

```
*****************************************************************************
*                    SYSTEM   DESIGN   FACTOR   ATTRIBUTE                   *
*****************************************************************************
```

FILE NAME: sdf_attribute

FIELD #	/FILED NAME	/DATA TYPE	/DATA SYZE IN BYTES
F1	id	integer	10
F2	RelationShip	integer	4
F3	Properties_type	char	20
F4	Properties_range	char	50
F5	Properties_units	char	50
F6	next_attributes	pointer	16

```
***************************************************************************
*                SYSTEM  DESIGN  FACTOR  QUANTIFICATION                   *
***************************************************************************
```

FILE NAME: sdf_quantification

FIELD	#/FIELD NAME	/DATE TYPE	SIZE IN BYTES
F1	id	integer	10
F2	aggregate	char	10
F3	variable_list	CHAR	20
F4	next_formula	pointer	16

end DIS_sdf_template_support;