AD-A267 725

CDRL No. 0002AB-6

Quarterly Technical Report - Report No. 6

April 1, 1993 - June 30, 1993

DARPA DICE Manufacturing Optimization

DTIC
ELECTE
AUG 10 1993
S A D

Linda J. Lapointe
Thomas J. Laliberty
Robert V.E. Bryant

Raytheon Company

1993

This document has been approved for public release and sale; its distribution is unlimited.

DARPA
Defense Advanced Research
Projects Agency

93-17681

93 8 3 248

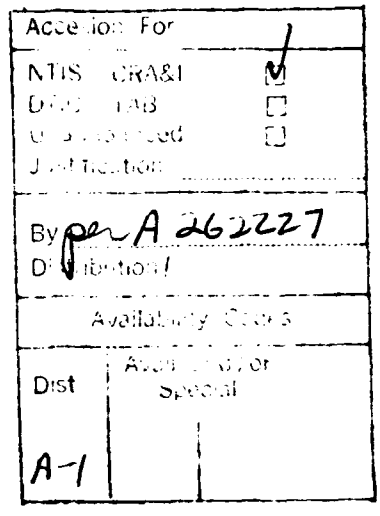
CDRL No. 0002AB-6

Quarterly Technical Report - Report No. 6

April 1, 1993 - June 30, 1993

DARPA DICE Manufacturing Optimization

Prepared by
Linda J. Lapointe
Thomas J. Laliberty
Robert V.E. Bryant



Raytheon Company
Missile Systems Laboratories
Tewksbury, MA 01876

July 1993

DTIC QUALITY INSPECTED 3

ARPA Order No. 8363/02
Contract MDA972-92-C-0020

Prepared for

DARPA

Defense Advanced Research
Projects Agency

Contracts Management Office
Arlington, VA 22203-1714

Contents

Figures

1. Summary

This is the Quarterly Technical Report for the DARPA DICE Manufacturing Optimization. The goal of the Manufacturing Optimization (MO) system is to facilitate a two tiered team approach to the product/process development cycle where a product design is analyzed by multiple manufacturing engineers, and product/process changes are traded concurrently in the product and process domains. The system will support Design for Manufacturing and Assembly (DFMA) with a set of tools to model manufacturing processes, and manage tradeoffs across multiple processes. The subject of this report is the technical work accomplished during the sixth quarter of the contract.

The main thrust of the sixth quarter was MO software development. Highlighted in this report are the manufacturing process model EXPRESS schemas and the C++ class header files.

During the next quarter, Raytheon's activities will include development, integration, and test of the MO System.

2. Introduction

This is the Quarterly Technical Report for the DARPA DICE Manufacturing Optimization. The concept behind the Manufacturing Optimization (MO) system is to facilitate a two tiered team approach to the product/process development cycle where a product design is analyzed by multiple manufacturing engineers, and product/process changes are traded concurrently in the product and process domains. The system will support DFMA with a set of tools to model manufacturing processes, and manage tradeoffs across multiple processes. The subject of this report is the technical work accomplished during the sixth quarter of the contract.

Raytheon spent the sixth quarter on MO software development. To demonstrate some of the work that was accomplished, we have included technical highlights of the manufacturing process model EXPRESS schemas and the C++ class header files developed for the MO system.

3. Process Model Schema Specification

In order to perform cost and yield analysis on a design, the manufacturing process must be modeled. The MO process model supports a hierarchical tree based model of a manufacturing enterprise. Processes, operations and steps are defined for a manufacturing activity. Rules are defined which tie the product data to the processes, operations and steps. The selection rules, if satisfied, will trigger the selection of that process, operation or step.

An object-oriented methodology has been employed to implement the model. To represent processes, operations, and steps in the tree structure, a generic Manufacturing Activity class named "MfgSpec" was defined. The MfgSpec objects contain information that is common to processes, operations, and steps. Within each MfgSpec is a reference to an "info" object. This info object contains the information specific to the type of manufacturing activity being modeled (i.e. process, operation, or step).

The Manufacturing Analyzer's selection methodology is done by traversing the process model in depth-first fashion. The logic at each manufacturing activity node will be evaluated to see if this is an applicable path to follow. The selected nodes are added to an analysis tree which is also modeled as a general purpose tree structure. After the entire process model has been evaluated and the applicable nodes identified, the analysis tree created during process selection is traversed in a post-order fashion so that the time and cost can be calculated.

The EXPRESS model specified in this section was created for process model representation. Figure 3.1-1 is an EXPRESS-G representation of the same model.

3.1 EXPRESS Schema for Process Model

This EXPRESS schema listing defines the process model. The process model schema references the resource_schema, as well as some predefined constants and types. The specification of the additional schema will follow.

EXPRESS Specification :

```
*)
INCLUDE 'resource.exp';

SCHEMA process_model;

REFERENCE FROM resource_schema;
```

```
CONSTANT

 -- Constants to Aid in Part Entity Status Markings
 AVAILABLE : INTEGER := 0;
 TESTING   : INTEGER := 1;
 TESTED    : INTEGER := 2;
 PROCESSED : INTEGER := 3;
 COMPLETED : INTEGER := 4;

 -- Ordering Constants
 SEQUENTIAL : INTEGER := 0;
 CONCURRENT : INTEGER := 1;

 -- Step Type Constants
 SETUP      : INTEGER := 0;
 RUNTIME    : INTEGER := 1;

END_CONSTANT;

TYPE MfgSpecOrder = ENUMERATION OF
  (SEQUENTIAL, CONCURRENT);
END_TYPE;

TYPE StepTypes = ENUMERATION OF
  (SETUP, RUNTIME);
END_TYPE;

TYPE PartEntityStatus = ENUMERATION OF
  (AVAILABLE, TESTING, TESTED, PROCESSED, COMPLETED);
END_TYPE;
( *
```

3.1.1 ProcessModel Entity

A ProcessModel entity is the specification of a manufacturing process model that contains a hierarchical tree structure of Manufacturing Activity entities (i.e. MfgSpec objects). Additional data about the model is also stored including its name, author, creation date, and last modification date.

EXPRESS Specification :

```
ENTITY ProcessModel;
    name : STRING;                  -- Process Model name
    creationDate : DateRec;         -- Model creation date
    modifyDate   : DateRec;         -- Model last modify date
    author : STRING;                -- Model author
    topProcess : MfgSpec;           -- Top MfgSpec in
END_ENTITY;                         -- hierarchical tree structure
```

Attribute definitions:

name: Name of the manufacturing process model.

creationDate: The date that the model was created.

modifyDate: The date that the model was last modified.

author: The author of the model.

topProcess: The root or top most process in the process model tree structure.

3.1.2 MfgSpec Entity

A MfgSpec entity is the definition of a manufacturing activity which contains manufacturing process information and its corresponding reasoning logic. If the reasoning logic is satisfied, then the MfgSpec node is included in the overall analysis results. MfgSpec's are organized as a hierarchical planning system. The hierarchical planning system takes the form of a tree where each node can have one parent and an optional list of ordered (i.e. sequential or concurrent) children. Each MfgSpec will also have a reference to its right sibling.

EXPRESS Specification :

```
ENTITY MfgSpec;
    id: STRING;                          -- Unitque MfgSpec Identifier
    info: Process;                       -- Manufacturing Process Information
    logic: ReasoningLogic;               -- Manufacturing Spec Reasoning Logic
    ordering: MfgSpecOrder;              -- Sequential or Concurrent Ordering
    parent : MfgSpec;                    -- Parents Spec
    children : LIST [0:?] OF MfgSpec;    -- List of Children (Descendants)
    rsibling : MfgSpec;                  -- Right Sibling
    entities : LIST [0:?] OF ROSEOBJECT; -- Spec Produced Entities
    specCost: Cost;                      -- Spec Cost
END_ENTITY;
```

Attribute definitions:

id: Unique Identifier of the manufacturing specification.

info: Pointer to the Manufacturing Process Information associated with this manufacturing specification node.

logic: Reasoning Logic associated with the Manufacturing Process information. The logic is comprised of design feature entity and attributes being present or of specific values.

ordering: Ordering associated with the children of this specification. The order can be Sequential or Concurrent.

parent: Parent MfgSpec node associated with this specification.

children: List of MfgSpec children associated with this specification.

rsibling: The right sibling associated with this MfgSpec tree node.

entities: List of entities produced by this specification for a particular part under analysis.

specCost: Total Cost of the manufacturing specification.

3.1.3 Process Entity

A Process Entity is the definition to support modeling of processes and sub-processes. A process is an organized sequence of events, either discrete or continuous, that transform raw materials into a finished product. A sub-process is an organized sequence of events, either discrete or continuous, that result in a transformation of the product.

EXPRESS Specification :

```
*)
ENTITY Process;
   name: STRING;                                  -- Process Name
   desc: STRING;                                  -- Description
   resources: LIST [0:?] OF ResourceUtilization;  -- Resources (i.e workcenter/workstation)
   qualResults : Quality;                         -- Process Quality
   indivRate : Cost;                              -- Individual Process time and cost
 END_ENTITY;
(*
```

Attribute definitions:

name: Manufacturing Process Name.

desc: Description of the Manufacturing Process.

resources: List of resources used by the process node as an entity. This list of resources are associated with the process node.

qualResults: The resulting Process Quality associated with this Process.

indivRate: The individual Time and Cost of the Process.

3.1.4 Operation Entity

An Operation Entity is the definition to support modeling of operations. An operation is a logical grouping of work, confined to one workcenter, and often one machine or machining cell where a discrete unit of work is performed.

EXPRESS Specification :

```
*)
```

```
        ENTITY Operation
          SUBTYPE OF (Process);
             optype: LaborClass;                    -- F, A, I, T
             scrap_rate : LIST [0:?] OF Scrap;      -- Scrap rates
             rework_rate : LIST [0:?] OF Rework;    -- Rework rates
        END_ENTITY;
(*
```

Attribute definitions:

optype: Type of Operation (i.e. fabrication, assembly, inspection, or test).

scrap_rate: A list of table entries providing an indexed lookup of scrap rates based on values of entities and their attributes or an equation that when evaluated will provide the scrap rate for the operation.

rework_rate: A list of table entries providing an indexed lookup of rework rates based on values of entities and their attributes or an equation that when evaluated will provide the rework rate for the operation.

3.1.5 Step Entity

A Step Entity is the definition to support modeling of steps. A step is an element of work inside an operation, analogous to specific actions.

EXPRESS Specification :

```
*)
        ENTITY Step
          SUBTYPE OF (Process);
             stepType: StepTypes;                   -- Setup or Run Time
        END_ENTITY;
(*
```

Attribute definitions:

stepType: Type of Step (i.e. setup or run time).

3.1.6 Scrap Entity

The scrap entity is used to represent scrap rate (i.e. scrap = 1-yield) data. Scrap is the percentage of parts that are lost or rejected at this operation. Scrap data is maintained in a list of scrap entities. In each entity there is a scrap rule and a corresponding scrap rate. If the scrap rule is satisfied, then the corresponding scrap rate is computed.

EXPRESS Specification :

```
*)
```

```
ENTITY Scrap;
    scrapRule : Rules;                    -- Rule to be evaluated
    scrapRate : Equation;                 -- Scrap that applies if rule is satisfied
    scrapPercentage: REAL;                -- Actual Calculated operational scrap rate
END_ENTITY;
```

(*

Attribute definitions:

scrapRule: The scrap rule to be evaluated.

scrapRate: The scrap rate equation to apply if the scrapRule is satisfied.

scrapPercentage: Scrap percentage associated with an operation in a particular part.

3.1.7 Rework Entity

The rework entity is used to represent rework rate data. Rework is the percentage of parts that must be reworked due to this operation. Rework data is maintained in a list of rework entities. In each entity there is a rework rule and a corresponding rework rate. If the rework rule is satisfied, then the corresponding rework rate is computed. There is a list of resources associated with the rework which is used to calculate the cost of performing the rework operation.

EXPRESS Specification :

*)

```
ENTITY Rework;
    reworkRule : Rules;                          -- Rule to be evaluated
    reworkRate : Equation;                       -- Rework that applies if rule is satisfied
    resources : LIST [0:?] OF ResourceUtilization; -- Rework resources
    reworkPercentage: REAL;                      -- Calculated operational rework rate
    reworkCost: REAL;                            -- Calculated Rework Cost
END_ENTITY;
```

(*

Attribute definitions:

reworkRule: The rework rule to be evaluated.

reworkRate: The rework rate equation to apply if the reworkRule is satisfied.

resources: The resources associated with the rework.

reworkPercentage: Rework percentage associated with an operation in a particular part.

reworkCost: Rework cost associated with an operation in a particular part.

3.1.8 Cost Data

The Cost data types and entities are used to represent calculated analyzer time and cost data.

EXPRESS Specification :
*)

```
ENTITY Cost;
   setupTime: REAL;          -- Operation Setup Time
   runTime:  REAL;           -- Operation Run Time
   idealTime: REAL;          -- Calculated Ideal Time
   idealCost: REAL;          -- Calculated Ideal Cost
   actualTime: REAL;         -- Calculated Actual Estimated Time
   actualCost: REAL:         -- Calculated Actual Estimated Cost
END_ENTITY;
```

(*

Attribute definitions:

setupTime: Operation calculated setup time.

runTime: Operation calculated run time.

IdealFait: Operation Fabrication, Assembly, Inspection, and Test Cost where no scrap and rework are included.

ActualFait: Actual Estimated Operation Fabrication, Assembly, Inspection, and Test Cost

3.1.9 Quality Data

The Quality data types and entities are used to represent calculated scrap, rework, and production quantity.

EXPRESS Specification :
*)

```
ENTITY Quality;
   scrapPercent: REAL;       -- Scrap Percentage
   prodQty:  INTEGER;        -- Production QTY
   reworkPercent: REAL;      -- Rework Percentage
   reworkCost: REAL;         -- Rework Cost
END_ENTITY;
```

(*

Attribute definitions:

scrapPercent: Calculated scrap percentage.

prodQty: Required production quantity.

reworkPercent: Calculated rework percentage.

reworkCost: Calculated rework cost.

3.1.10 ReasoningLogic Entity

The ReasoningLogic entity is used to hold the selection rules for the manufacturing activity node. The rules define the reasons behind why a node should or should not be selected as part of the process to manufacture a part.

EXPRESS Specification :
*)

```
ENTITY ReasoningLogic;
   rules: LIST [0:?] OF Rules;                    -- List of selection rules
END_ENTITY;
```

(*

Attribute definitions:

rules: List of manufacturing activity selection rules.

3.2 EXPRESS-G Schema for Process Model

The following EXPRESS-G model (figure 3.1-1) represents the Process Model schema:

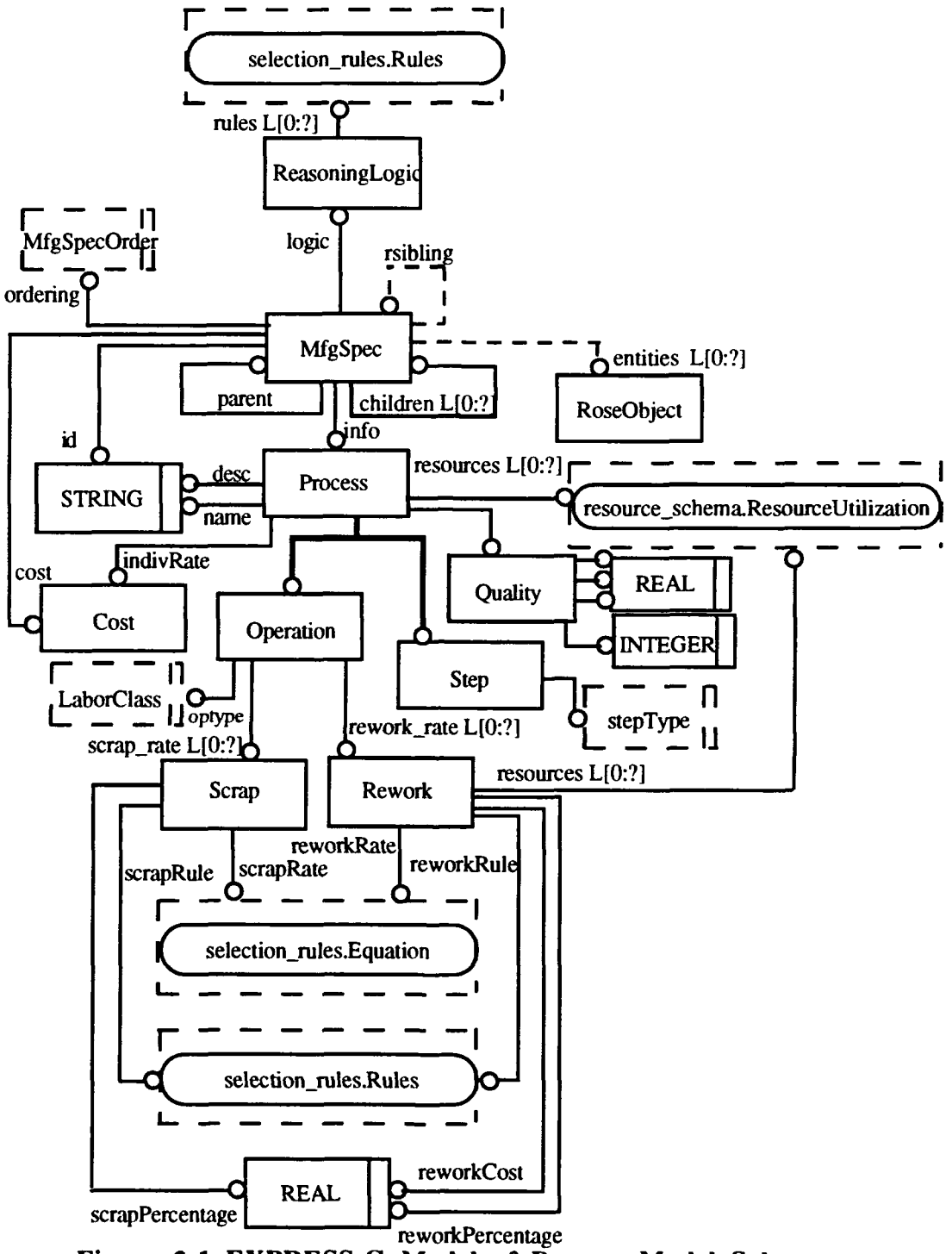


Figure 3-1 EXPRESS-G Model of Process Model Schema

3.3 EXPRESS Schema for Resource

The resource schema defines a collection of entities that are used to specify resources. A resource is any facility, labor, equipment, or consumable material used in the manufacturing

process. A consumable material is a material that is used to aid the manufacturing process and is not considered raw material of the product. As defined in the schema a resource is a generic entity. Specific subtypes of the resource entity are defined to represent facilities, people, equipment, and consumable materials. The resource schema includes the selection_rules schema, as well as some predefined constants and types. The specification of the additional schema will follow.

EXPRESS Specification :

```
*)
INCLUDE 'rules.exp';

SCHEMA resource_schema;

REFERENCE FROM selection_rules;

 CONSTANT

   -- Labor Classification Types
   FABRICATION : INTEGER := 0;
   ASSEMBLY    : INTEGER := 1;
   INSPECTION  : INTEGER := 2;
   TEST        : INTEGER := 3;

 END_CONSTANT;

 TYPE LaborClass = ENUMERATION OF
    (FABRICATION, ASSEMBLY, INSPECTION, TEST);
 END_TYPE;
( *
```

3.3.1 ResourceUtilization Entity

The ResourceUtilization Entity is used to store which resource(s) are utilized by a process or operation.

EXPRESS Specification :

```
* )
     ENTITY ResourceUtilization;
        resource : Resource;                  -- Resource utilized
        setupTime: Equation;                  -- Setup Equation
        runTime:  Equation;                   -- RunTime Equation
        effRate : OPTIONAL REAL;              -- Efficiency Rate
        rate: ResourceRates;                  -- Calculated Resource Rates
     END_ENTITY;

( *
```

Attribute definitions:

resource: The resource being utilized.

setupTime: The amount of setup time required for the resource.

runTime: The amount of time that the resource is being used while running the operation.

effRate: This optional attribute provides an efficiency rate factor that when applied to a labor standard associated with an operation will provide the actual time for the operation.

rate: Calculated Resource Time and Cost Rates.

3.3.2 Resource Entity

This is the generic resource entity. Each resource is named and can be coded of a certain type. A list of generic attributes can be attached to each resource using the parameter entity.

EXPRESS Specification :

```
* )

ENTITY Resource;
   resource_name : STRING;                    -- Resource Name
   resource_code : STRING;                    -- Resource Code
   parameters : LIST [0:?] of Parameter;      -- Resource Parameters
END_ENTITY;

( *
```

Attribute definitions:

resource_name: The name string associated with the resource.

resource_code: A string used to assign a code to the resource.

parameters: A list of generic attributes that can be attached to this resource.

3.3.3 Parameter Entity

The parameter entity is used to define a generic attribute.

EXPRESS Specification :

```
* )

ENTITY Parameter;
   p_name : STRING;                           -- Parameter Name
   p_value : STRING;                          -- Parameter Value
END_ENTITY;
( *
```

Attribute definitions:

p_name: The name of the parameter.

p_value: The value of the parameter.

3.3.4 Labor Entity

The entities in this section define the labor resource. The labor entity is a subtype of the generic resource entity.

EXPRESS Specification :

```
*)

ENTITY Labor SUBTYPE OF (Resource);
   job_code : STRING;                    -- Labor Job Code
   l_type: LaborClass;                   -- Labor Type
   rate : REAL;                          -- Labor Rate
END_ENTITY;

(*
```

Attribute definitions:

job_code : A unique identifier associated with the labor.

l_type: Labor Type (i.e. Fabrication, Assembly, Inspection, Test)

rate : The labor rate.

3.3.5 Equipment Entity

The equipment entity is a subtype of the generic resource entity. It is used to specify the cost of operating the equipment resource during an operation or process.

EXPRESS Specification :

```
*)

ENTITY Equipment SUBTYPE OF (Resource);
   equipment_category : STRING;          -- Equipment Category
   cost_per_time_unit : REAL;            -- Cost Per Time Unit
END_ENTITY;

(*
```

Attribute definitions:

equipment_category: The equipment code or category.

cost_per_time_unit: The cost of operating the equipment resource per unit of time.

3.3.6 Facility Entity

The facility entity is a subtype of the generic resource entity. It is used to specify the cost of using the facility resource during an operation or process.

EXPRESS Specification :

```
* )
ENTITY facility SUBTYPE OF (Resource);
   square_feet_allocated : REAL;                  -- Square Feet Allocated
   cost_per_sq_ft_per_time_unit : REAL;           -- Cost Per Sq Foot Per Time Unit
END_ENTITY;
( *
```

Attribute definitions:

square_feet_allocated: The square feet allocated to this particular operation or process.

cost_per_sq_ft_per_time_unit: The cost per square foot per time unit.

3.3.7 ConsumableMaterial Entity

The consumable material entity is a subtype of the generic resource entity. Consumable materials are those materials used to aid in the manufacturing of a product that are consumed by the process. These materials are not considered as part of the raw materials used in the manufacture of the product. They only aid in the production process and are consumed at some measurable rate during the process.

EXPRESS Specification :

```
* )
ENTITY ConsumableMaterial SUBTYPE OF (Resource);
   cost_per_unit : REAL;                          -- Cost Per Unit
   resourceRates: LIST [0:?] OF ResourceConsumable;   -- list of resource rates
END_ENTITY;

ENTITY ResourceConsumable;
   aresource : Resource;                          -- Associated Resource
   units_exhausted_per_time_unit : REAL;          -- Units Exhausted Per Hour
END_ENTITY;
( *
```

Attribute definitions:

cost_per_unit: The cost of one unit of the consumable material.

resourceRates: The list of resource rates.

aResource: The associated Consumable Resource.

units_exhausted_per_time_unit: Units consumed per unit of time during or by the operation or process.

3.3.8 ResourceRates Entity

The ResourceRates entity is the entity which holds the calculated time and cost data associated with the resources.

EXPRESS Specification :

```
*)
ENTITY ResourceRates;
   setupTime: REAL;                    -- setup Time
   runTime: REAL;                      -- run Time
   idealTime: REAL;                    -- ideal Time
   idealCost: REAL;                    -- ideal Cost
END_ENTITY;

(*
```

Attribute definitions:

setupTime: Setup Time associated with the Resources.

runTime: Run Time associated with the Resources.

idealTime: Ideal Time associated with the Resources.

idealCost: Ideal Cost associated with the Resources.

3.4 EXPRESS-G Schema for Resource

The following EXPRESS-G schema (figure 3.1-2) represents the Resource schema:

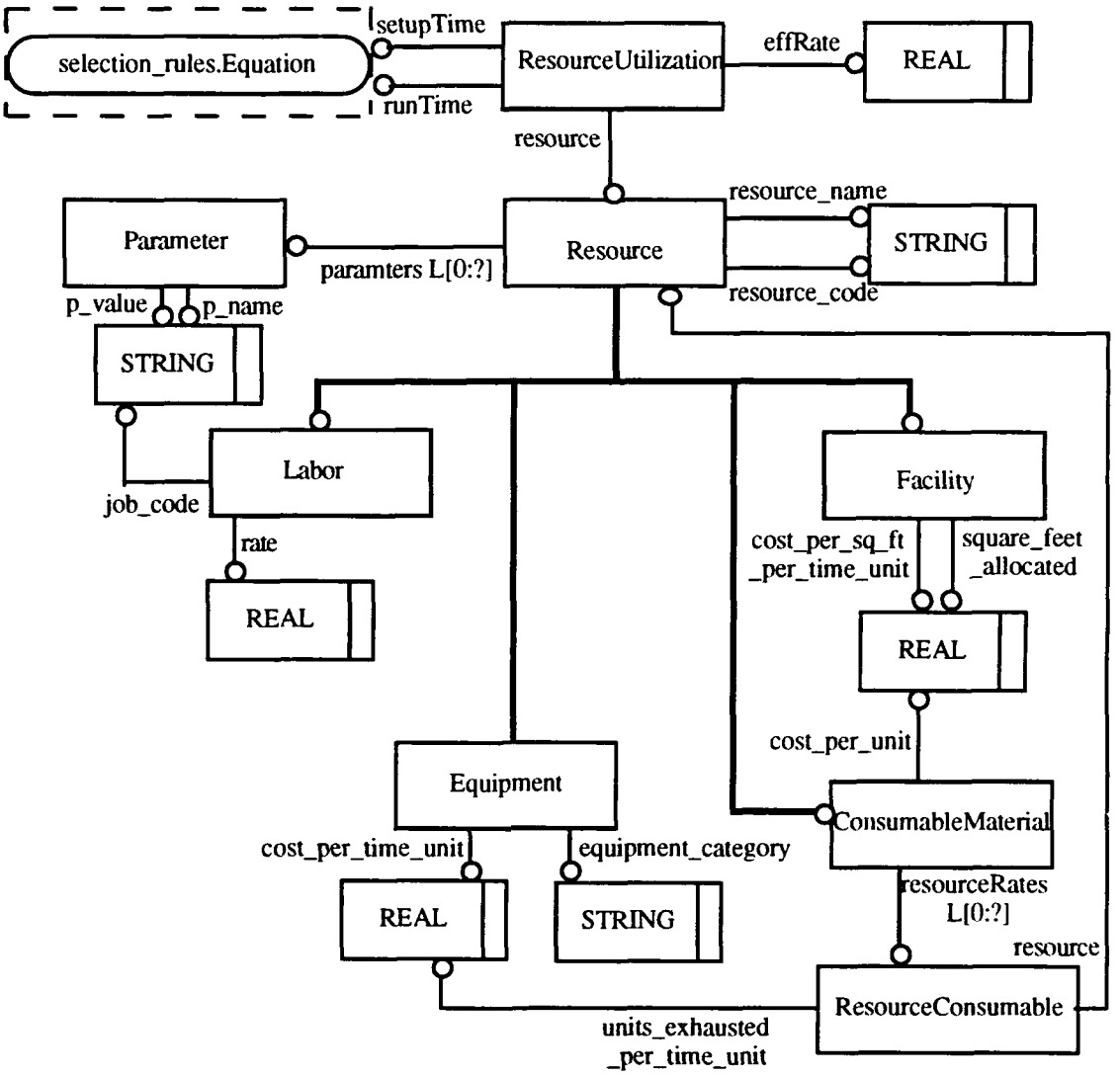


Figure 3-2 EXPRESS-G Model of Resources Schema

3.5 EXPRESS Schema for Selection Rules

This schema defines a grammar format which rules for selection and equations for evaluation are specified. Rules are tied to process nodes and equations are tied to such entities as scrap and rework formulas. Provided below is the complete BNF (Backus-Naur Form) grammar format for the selection rules and equations which the EXPRESS schema is based on.

Rule Grammar Format

<rule> := <expression>, [<rule>]

<expression> := <equation> | <complexExp> | <simpleExp> | <stringValue>

<complexExp> := <equation> <equiv_op> <expression>

<simpleExp> := <unary_op> <DataDictStr>

<stringValue> := "string"

<equation> := <term> | <complexEquation>

<complexEquation> := <term> <Add_Sub_Oper> <equation>

<Add_Sub_Oper> := + addition
- subtraction

<term> := <const> | <DataDictStr> | <parenEquation> | <complexTerm>

<const> := real numbers | integers

<DataDictStr> := <entity> | <entityAttr> | <SpecialFunct>

<parenEquation> := (<equation>)

<complexTerm> := <equation> <Mult_Div_Oper> <equation>

<Mult_Div_Oper> := * multiplication
/ division

<unary_op> := ! not

<equiv_op> := < less than
<= less than equal to
> greater than
>= greater than equal to
= equal to
!= not equal to

Operator Precedence (ordered by most --> least priority)

Priority	Operator	Description	
1	!	logical negation	
2	*	multiplication	
	/	division	(left to right)
3	+	addition	
	-	subtraction	(left to right)
4	<	less than	
	<=	less than equal to	
	>	greater than	(left to right)

```
      >= greater than equal to
      =  equal to
     != not equal to
==========================================================
```

3.5.1 Constants and Types for Rule Construction

The following is a listing of the EXPRESS source that defines symbolic constants and aggregate types that are necessary for the specification of the rule's BNF:

EXPRESS Specification :

```
* )

SCHEMA selection_rules;

    CONSTANT
       Multiply              : STRING := '*';
       Divide                : STRING := '/';
       Add                   : STRING := '+';
       Subtract              : STRING := '-';

       U_Op                  : STRING := '!';

       Less                  : STRING := '<';
       LessEqual             : STRING := '<=';
       Greater               : STRING := '>';
       GreaterEqual          : STRING := '>=';
       Equal                 : STRING := '=';
       NotEqual              : STRING := '!=';

       LP                    : STRING := '(';
       RP                    : STRING := ')';
       DQ                    : STRING := '"';
    END_CONSTANT;

    TYPE DQuote = ENUMERATION OF (DQ);
    END_TYPE;

    TYPE LParen = ENUMERATION OF (LP);
    END_TYPE;

    TYPE RParen = ENUMERATION OF (RP);
    END_TYPE;

    TYPE Unary_Op = ENUMERATION OF (U_Op);
    END_TYPE;

    TYPE Strings = STRING;
    END_TYPE;

    TYPE Real_numbers = REAL;
    END_TYPE;

    TYPE Integers = INTEGER;
    END_TYPE;
```

```
TYPE
   TokenReturnValue = SELECT (Real_numbers, Integers, Strings);
END_TYPE;

TYPE
   Const = SELECT (Real_numbers, Integers);
END_TYPE;

TYPE Add_Sub_Oper = ENUMERATION OF
   (Add, Subtract);
END_TYPE;

TYPE Mult_Div_Oper = ENUMERATION OF
   (Multiply, Divide);
END_TYPE;

TYPE Equiv_Op = ENUMERATION OF
   (Less, LessEqual, Greater, GreaterEqual, Equal, NotEqual);
END_TYPE;
```

(*

3.5.2 DataDictStr Entity

The DataDictStr entity is an abstract base class from which two subclasses have been created. The first is the EntityName class which holds the name of an entity name. The other is the EntityAttrName which is used to support the following entity attribute specification :

```
entity[.entity[.entity[... .attr]]]
```

An example of an instance of this might be :

```
line.point1.x
```

EXPRESS Specification :

*)

```
ENTITY DataDictStr; -- abstract base class
END_ENTITY;

ENTITY EntityName;
   SUBTYPE OF (DataDictStr);
      name : STRING;
END_ENTITY;
```

(*

Attribute definitions:

name: The name of the entity as it appears in the product data EXPRESS model.

*)

EXPRESS Specification :
*)

```
ENTITY EntityAttrName
   SUBTYPE OF (DataDictStr);
      entityName : LIST [1:?] OF STRING;
      attrName   : STRING;
END_ENTITY;
```

(*

Attribute definitions:

entityName: List of entity name that corresponds to the structure : .ent[.ent[... .ent]]. of the entity as it appears in the product data EXPRESS model.

attrName: The attribute name which the final value is associate with. These attribute names should be specified as they appear in the product data EXPRESS model.

3.5.3 Rules Entities

A complex rule is composed of a list of rules. A rule is an Expressions anded together. The following BNF segment defines the grammar of the EXPRESS entities :

<Rules> := <Expression> , [<Rules>]

EXPRESS Specification :
*)

```
ENTITY Rules;
   exp1 : LIST [1:?] OF Expression;
   moreRulesFiring : BOOLEAN;
END_ENTITY;
```

(*

3.5.4 Expression Entities

The Expression syntax is represented by the following BNF segment :

<Expression> := <Equation> | <ComplexExp> | <SimpleExp> | <StringValue>

EXPRESS Specification :
*)

```
TYPE
   Expression = SELECT (Equation, ComplexExp, SimpleExp, StringValue);
END_TYPE;

ENTITY StringValue;
   quote1 : DQuote;
```

```
        value1 : STRING;
        quote2 : DQuote;
    END_ENTITY;

    ENTITY ComplexExp;
        Equ1    : Equation;
        EquivOp1 : Equiv_Op;
        Exp1    : Expression;
    END_ENTITY;

    ENTITY SimpleExp;
        Not1 : Unary_Op;
        DataDictVar : DataDictStr;
    END_ENTITY;
```

(*

3.5.5 Equation Entities

The Equation syntax is represented by the following BNF segment :

<Equation> := <Term> | <ComplexEquation>

EXPRESS Specification :

*)

```
    TYPE
        Equation = SELECT (Term, ComplexEquation);
    END_TYPE;

    ENTITY ComplexEquation;
        Var1    : Term;
        Oper1   : Add_Sub_Oper;
        Value   : Equation;
    END_ENTITY;

    ENTITY ParenEquation;
        Lparenthesis    : LParen;
        Equ             : Equation;
        Rparenthesis    : RParen;
    END_ENTITY;
```

(*

3.5.6 Term Entities

The Term syntax is represented by the following BNF segment :

<Term> := <Const> | <DataDictStr> | <ParenEquation> | <ComplexTerm>

EXPRESS Specification :

*)

```
    TYPE
      Term = SELECT (Const, DataDictStr, ParenEquation, ComplexTerm);
    END_TYPE;

END_SCHEMA;

(*
```

3.5.7 ComplexTerm Entities

The Term syntax is represented by the following BNF segment :

<ComplexTerm> := <equation> <mult_div_oper> <equation>

EXPRESS Specification :

```
*)

    ENTITY ComplexTerm;
      equ1     : Equation;
      Oper1    : Mult_Div_Oper;
      equ2     : Equation;
    END_ENTITY;

END_SCHEMA;

(*
```

3.6 EXPRESS-G Schema for Selection Rules

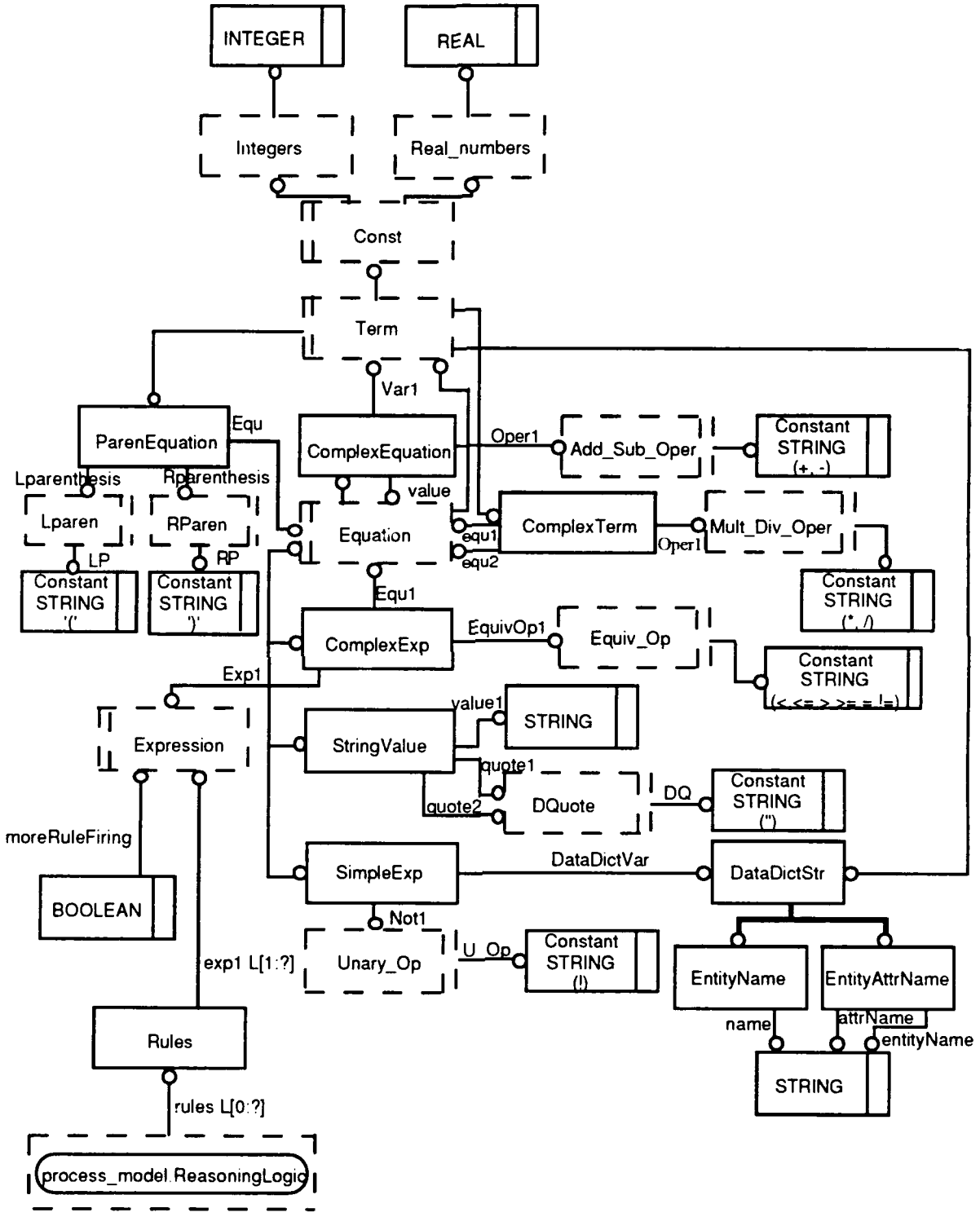


Figure 3-3 EXPRESS-G Model of Selection Rules Schema

4. C++ Header File Definitions

This section provides the C++ header files for the MO system. These files contain the definition of the pertinent classes and objects in the system.

The class specifications defined in this section were developed as follows: the EXPRESS information modeling language was used to model both the product data and the MO process data (Section 6 provides a complete EXPRESS schema specification of the product and process models). Using the express2c++ compiler which is part of the STEP Programmers Toolkit (STEP Tools Inc.), the EXPRESS entities were translated into C++ classes. The generated classes are structured such that all of the class attributes are declared as private. Public access and update methods were generated for each private attribute. Each generated class was then extended to support the additional calculation and monitoring methods required for the system.

Figure 4-1 illustrates the top-level class categories for the MO system. The sections to follow provide the details of the class specifications of each of these categories.

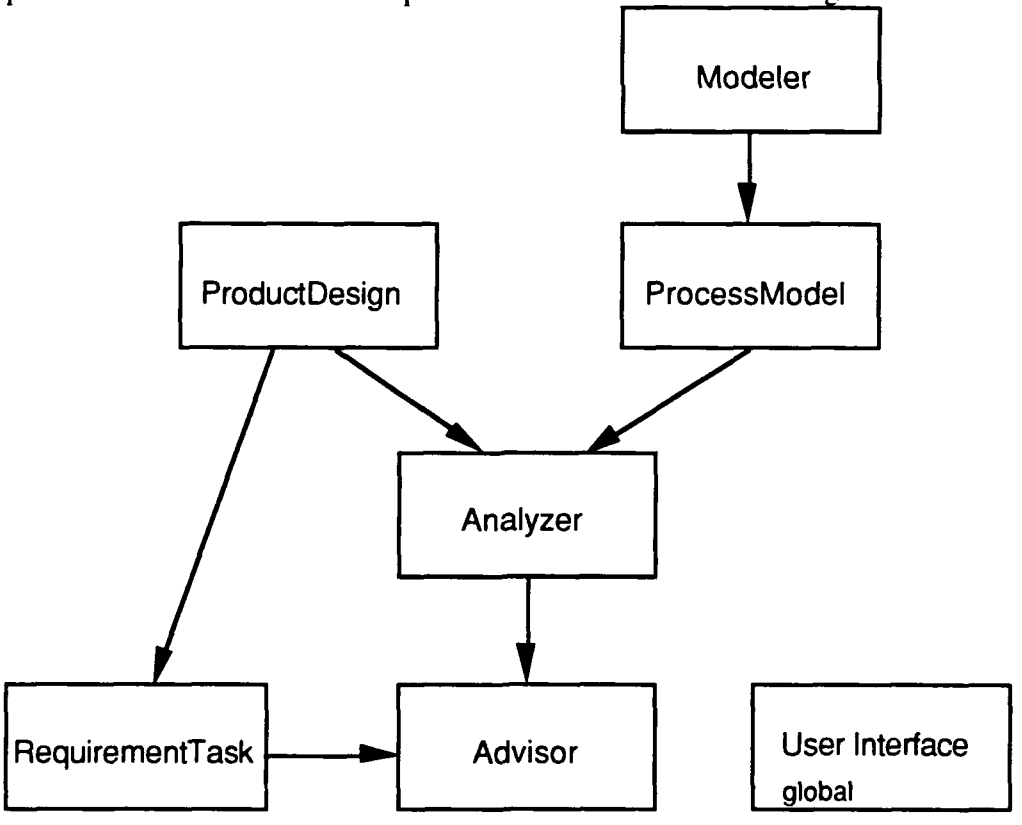


Figure 4-1 Top-Level Class (Categories) Diagram

4.1 ProductDesign

An EXPRESS product model was developed to model PWB data and electronic component library data. The model consists of approximately twenty interrelated EXPRESS schemas consisting of more than one hundred and fifty entities. C++ source code was produced by the express2c++ compiler as described above. The following specification is for the "route_rec" C++ class.

```
/* Class Declaration */
ROSE_DECLARE (route_rec) : virtual public RoseStructure {
 private:
        STR PERSISTENT_signal;
        STR PERSISTENT_route_type;
        STR PERSISTENT_status;
        pin_name_rec * PERSISTENT_target_name;
        pin_name_rec * PERSISTENT_object_name;
        pin_rec * PERSISTENT_target_pin;
        pin_rec * PERSISTENT_object_pin;
        point_rec * PERSISTENT_target_loc;
        point_rec * PERSISTENT_object_loc;
        BOOL PERSISTENT_protect;
        int PERSISTENT_target_layer;
        int PERSISTENT_object_layer;
        ListOfsegment_rec * PERSISTENT_path;
        int PERSISTENT_shield_id;
        int PERSISTENT_pin_pair_index;
        pin_pair_rec * PERSISTENT_pin_pair;
        ww_route_data_rec * PERSISTENT_ww_data; STR PERSISTENT_comment;
 public:
        ROSE_DECLARE_MEMBERS(route_rec);

/* Access and Update Methods */
/* signal Access Methods */
STR signal()
{       return ROSE_GET_PRIM (STR,PERSISTENT_signal);
}

void signal (STR asignal)
{       ROSE_PUT_PRIM (STR,PERSISTENT_signal,asignal); }

/* route_type Access Methods */
STR route_type()
{       return ROSE_GET_PRIM (STR,PERSISTENT_route_type);
}

void route_type (STR aroute_type)
{       ROSE_PUT_PRIM (STR,PERSISTENT_route_type,aroute_type); }

/* status Access Methods */
```

```
STR status()
{       return ROSE_GET_PRIM (STR,PERSISTENT_status);
}

void status (STR astatus)
{       ROSE_PUT_PRIM (STR,PERSISTENT_status,astatus); }

/* target_name Access Methods */
pin_name_rec * target_name()
{       return ROSE_GET_OBJ (pin_name_rec,PERSISTENT_target_name);
}

void target_name (pin_name_rec * atarget_name)
{       ROSE_PUT_OBJ (pin_name_rec,PERSISTENT_target_name,atarget_name); }

/* object_name Access Methods */
pin_name_rec * object_name()
{       return ROSE_GET_OBJ (pin_name_rec,PERSISTENT_object_name);
}

void object_name (pin_name_rec * aobject_name)
{       ROSE_PUT_OBJ (pin_name_rec,PERSISTENT_object_name,aobject_name); }

/* target_pin Access Methods */
pin_rec * target_pin()
{       return ROSE_GET_OBJ (pin_rec,PERSISTENT_target_pin);
}

void target_pin (pin_rec * atarget_pin)
{       ROSE_PUT_OBJ (pin_rec,PERSISTENT_target_pin,atarget_pin); }

/* object_pin Access Methods */
pin_rec * object_pin()
{       return ROSE_GET_OBJ (pin_rec,PERSISTENT_object_pin);
}

void object_pin (pin_rec * aobject_pin)
{       ROSE_PUT_OBJ (pin_rec,PERSISTENT_object_pin,aobject_pin); }

/* target_loc Access Methods */
point_rec * target_loc()
{       return ROSE_GET_OBJ (point_rec,PERSISTENT_target_loc);
}

void target_loc (point_rec * atarget_loc)
{       ROSE_PUT_OBJ (point_rec,PERSISTENT_target_loc,atarget_loc); }

/* object_loc Access Methods */
point_rec * object_loc()
{       return ROSE_GET_OBJ (point_rec,PERSISTENT_object_loc);
}

void object_loc (point_rec * aobject_loc)
```

```
{       ROSE_PUT_OBJ (point_rec,PERSISTENT_object_loc,aobject_loc); }

/* protect Access Methods */
BOOL protect()
{       return ROSE_GET_PRIM (BOOL,PERSISTENT_protect);
}

void protect (BOOL aprotect)
{       ROSE_PUT_PRIM (BOOL,PERSISTENT_protect,aprotect); }

/* target_layer Access Methods */
int target_layer()
{       return ROSE_GET_PRIM (int,PERSISTENT_target_layer);
}

void target_layer (int atarget_layer)
{       ROSE_PUT_PRIM (int,PERSISTENT_target_layer,atarget_layer); }

/* object_layer Access Methods */
int object_layer()
{       return ROSE_GET_PRIM (int,PERSISTENT_object_layer);
}

void object_layer (int aobject_layer)
{       ROSE_PUT_PRIM (int,PERSISTENT_object_layer,aobject_layer); }

/* path Access Methods */
ListOfsegment_rec * path();
void path (ListOfsegment_rec *  apath)
{       ROSE_PUT_OBJ (ListOfsegment_rec,PERSISTENT_path,apath); }

ListOfsegment_rec * route_rec :: path()
{       if( !PERSISTENT_path)
                if( this->isPersistent())
                        path (pnewIn (design()) ListOfsegment_rec);
        else    path (new ListOfsegment_rec);
        return ROSE_GET_OBJ (ListOfsegment_rec,PERSISTENT_path);
}


/* shield_id Access Methods */
int shield_id()
{       return ROSE_GET_PRIM (int,PERSISTENT_shield_id);
}

void shield_id (int ashield_id)
{       ROSE_PUT_PRIM (int,PERSISTENT_shield_id,ashield_id); }

/* pin_pair_index Access Methods */
int pin_pair_index()
{       return ROSE_GET_PRIM (int,PERSISTENT_pin_pair_index);
}
```

```
void pin_pair_index (int apin_pair_index)
{       ROSE_PUT_PRIM (int,PERSISTENT_pin_pair_index,apin_pair_index); }

/* pin_pair Access Methods */
pin_pair_rec * pin_pair()
{       return ROSE_GET_OBJ (pin_pair_rec,PERSISTENT_pin_pair);
}

void pin_pair (pin_pair_rec * apin_pair)
{       ROSE_PUT_OBJ (pin_pair_rec,PERSISTENT_ ˙ר_pa .,ɯpin_pair); }

/* ww_data Access Methods */
ww_route_data_rec * ww_data()
{       return ROSE_GET_OBJ (ww_route_data_rec,PERSISTENT_ww_data);
}

void ww_data (ww_route_data_rec * aww_data)
{       ROSE_PUT_OBJ (ww_route_data_rec,PERSISTENT_ww_data,aww_data); }

/* comment Access Methods */
STR comment()
{       return ROSE_GET_PRIM (STR,PERSISTENT_comment);
}

void comment (STR acomment)
{       ROSE_PUT_PRIM (STR,PERSISTENT_comment,acomment); }

/* Constructors */
route_rec ();
route_rec (
        STR asignal,
        STR aroute_type,
        STR astatus,
        pin_name_rec * atarget_name,
        pin_name_rec * aobject_name,
        pin_rec * atarget_pin,
        pin_rec * aobject_pin,
        point_rec * atarget_loc,
        point_rec * aobject_loc,
        BOOL aprotect,
        int atarget_layer,
        int aobject_layer,
        ListOfsegment_rec * apath,
        int ashield_id,
        int apin_pair_index,
        pin_pair_rec * apin_pair,
        ww_route_data_rec * aww_data,
        STR acomment );
};
```

4.2 ProcessModel

The ProcessModel class is used to manage the manufacturing process models. Each ProcessModel object contains a reference to the top node in the hierarchical process tree structure. Each also contains the name of the model, the dates of its creation and last modification, and the name of the author of the model. The ProcessModel objects are created by the Modeler managing object. The Analyzer traverses the ProcessModel in order to select the appropriate analysis plan for the ProductDesign under analysis, and calculate the corresponding yield, rework, and cost of each selected process and operation. The analysis plan is a subset of the original ProcessModel object. The Advisor managing object provides viewing of the resulting Analyzer process plan. The sub-sections that follow detail each of the ProcessModel, Resource, and ReasoningLogic classes/objects and their corresponding methods.

4.2.1 ProcessModel Specification

```
/* Class Declaration */
#ifndef ProcessModel_h
#define ProcessModel_h

#include "rose.h"
#include "process_model_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "ProcessModel.hi"

ROSE_DECLARE (DateRec);
ROSE_DECLARE (MfgSpec);
#define ProcessModelOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,ProcessModel)

ROSE_DECLARE (ProcessModel) : virtual public RoseStructure {
 private:
        STR PERSISTENT_name;
        DateRec * PERSISTENT_creationDate;
        DateRec * PERSISTENT_modifyDate;
        STR PERSISTENT_author;
        MfgSpec * PERSISTENT_topProcess;

 public:
        ROSE_DECLARE_MEMBERS(ProcessModel);

/* Access and Update Methods */
```

```
/* name Access Methods */
STR name()
{       return ROSE_GET_PRIM (STR,PERSISTENT_name);
}
void name (STR aname)
{       ROSE_PUT_PRIM (STR,PERSISTENT_name,aname); }

/* creationDate Access Methods */
DateRec * creationDate()
{       return ROSE_GET_OBJ (DateRec,PERSISTENT_creationDate);
}
void creationDate (DateRec * acreationDate)
{       ROSE_PUT_OBJ (DateRec,PERSISTENT_creationDate,acreationDate); }

/* modifyDate Access Methods */
DateRec * modifyDate()
{       return ROSE_GET_OBJ (DateRec,PERSISTENT_modifyDate);
}
void modifyDate (DateRec * amodifyDate)
{       ROSE_PUT_OBJ (DateRec,PERSISTENT_modifyDate,amodifyDate); }

/* author Access Methods */
STR author()
{       return ROSE_GET_PRIM (STR,PERSISTENT_author);
}
void author (STR aauthor)
{       ROSE_PUT_PRIM (STR,PERSISTENT_author,aauthor); }

/* topProcess Access Methods */
MfgSpec * topProcess()
{       return ROSE_GET_OBJ (MfgSpec,PERSISTENT_topProcess);
}
void topProcess (MfgSpec * atopProcess)
{       ROSE_PUT_OBJ (MfgSpec,PERSISTENT_topProcess,atopProcess); }

/* Constructors */
ProcessModel ();
ProcessModel (
        STR aname,
        DateRec *  acreationDate,
        DateRec *  amodifyDate,
        STR aauthor,
        MfgSpec *  atopProcess );

/* CLASS DECLARATION EXTENSIONS */
/* Process Selection Traversal Method */
ProcessModel * SelectProcessFlow(ProductEntities *);

/* PreOrder Process Model Display Method */
void PreOrderDisplay();

/* PostOrder Process Model Display Method */
void PostOrderDisplay(MfgSpec *);

/* Calculates Labor Standards Associated With Selected Processes */
```

```
void CalculateLaborStds(int);

/* Determines Total Cost of each Process/Operation/Step */
void DetermineTotalCost();

/* Advisor Display Method */
void AdvisorDisplay();
};
#endif
```

4.2.2 MfgSpec Specification

```
/* Class Declaration */
#ifndef MfgSpec_h
#define MfgSpec_h

#include "rose.h"
#include "process_model_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "MfgSpec.hi"

ROSE_DECLARE (Process);
ROSE_DECLARE (ReasoningLogic);
ROSE_DECLARE (MfgSpec);
ROSE_DECLARE (ListOfMfgSpec);
ROSE_DECLARE (Cost);
#define MfgSpecOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,MfgSpec)

ROSE_DECLARE (MfgSpec) : virtual public RoseStructure {
 private:
        STR PERSISTENT_id;
        Process * PERSISTENT_info;
        ReasoningLogic * PERSISTENT_logic;
        MfgSpecOrder PERSISTENT_ordering;
        MfgSpec * PERSISTENT_parent;
        ListOfMfgSpec * PERSISTENT_children;
        MfgSpec * PERSISTENT_rsibling;
        ListOfRoseObject * PERSISTENT_entities;
        Cost * PERSISTENT_specCost;

 public:
        ROSE_DECLARE_MEMBERS(MfgSpec);

/* Access and Update Methods */

/* id Access Methods */
STR id()
{       return ROSE_GET_PRIM (STR,PERSISTENT_id);
}
void id (STR aid)
```

```
{       ROSE_PUT_PRIM (STR,PERSISTENT_id,aid); }

/* info Access Methods */
Process * info()
{       return ROSE_GET_OBJ (Process,PERSISTENT_info);
}
void info (Process * ainfo)
{       ROSE_PUT_OBJ (Process,PERSISTENT_info,ainfo); }

/* logic Access Methods */
ReasoningLogic * logic()
{       return ROSE_GET_OBJ (ReasoningLogic,PERSISTENT_logic);
}
void logic (ReasoningLogic * alogic)
{       ROSE_PUT_OBJ (ReasoningLogic,PERSISTENT_logic,alogic); }

/* ordering Access Methods */
MfgSpecOrder ordering()
{       return ROSE_GET_PRIM (MfgSpecOrder,PERSISTENT_ordering);
}
void ordering (MfgSpecOrder aordering)
{       ROSE_PUT_PRIM (MfgSpecOrder,PERSISTENT_ordering,aordering); }

/* parent Access Methods */
MfgSpec * parent()
{       return ROSE_GET_OBJ (MfgSpec,PERSISTENT_parent);
}
void parent (MfgSpec * aparent)
{       ROSE_PUT_OBJ (MfgSpec,PERSISTENT_parent,aparent); }

/* children Access Methods */
ListOfMfgSpec * children();
void children (ListOfMfgSpec *  achildren)
{       ROSE_PUT_OBJ (ListOfMfgSpec,PERSISTENT_children,achildren); }

/* rsibling Access Methods */
MfgSpec * rsibling()
{       return ROSE_GET_OBJ (MfgSpec,PERSISTENT_rsibling);
}
void rsibling (MfgSpec * arsibling)
{       ROSE_PUT_OBJ (MfgSpec,PERSISTENT_rsibling,arsibling); }

/* entities Access Methods */
ListOfRoseObject * entities();
void entities (ListOfRoseObject *  aentities)
{       ROSE_PUT_OBJ (ListOfRoseObject,PERSISTENT_entities,aentities); }

/* specCost Access Methods */
Cost * specCost()
{       return ROSE_GET_OBJ (Cost,PERSISTENT_specCost);
}
void specCost (Cost * aspecCost)
{       ROSE_PUT_OBJ (Cost,PERSISTENT_specCost,aspecCost); }
```

```
/* Constructors */
MfgSpec ();
MfgSpec (
        STR aid,
        Process *  ainfo,
        ReasoningLogic *  alogic,
        MfgSpecOrder aordering,
        MfgSpec *  aparent,
        ListOfMfgSpec *  achildren,
        MfgSpec *  arsibling,
        ListOfRoseObject *  aentities,
        Cost *  aspecCost );

/* CLASS DECLARATION EXTENSIONS */
/* Determine Cost of Spec */
void DetermineSpecCost();

/* Calculate Spec Labor Stds */
void CalculateLaborStds(int);

/* Locate Spec parent in Results tree */
void LocateParent(MfgSpec *, ProductEntities *);

/* Deep Copy MfgSpec Node */
MfgSpec *AddMfgSpec(ProductEntities *, MfgSpec *);

/* Determine if MfgSpec is Applicable to this part */
BOOL Select(ProductEntities *);

/* Display MfgSpec */
void Display();
};
#endif
```

4.2.3 Process Specification

```
/* Class Declaration */
#ifndef Process_h
#define Process_h

#include "rose.h"
#include "process_model_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Process.hi"

ROSE_DECLARE (ListOfResourceUtilization);
ROSE_DECLARE (Quality);
ROSE_DECLARE (Cost);
#define ProcessOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Process)

ROSE_DECLARE (Process) : virtual public RoseStructure {
```

```
private:
        STR PERSISTENT_name;
        STR PERSISTENT_desc;
        ListOfResourceUtilization *  PERSISTENT_resources;
        Quality *  PERSISTENT_qualResults;
        Cost *  PERSISTENT_indivRate;

public:
        ROSE_DECLARE_MEMBERS(Process);

/* Access and Update Methods */

/* name Access Methods */
STR name()
{       return ROSE_GET_PRIM (STR,PERSISTENT_name);
}
void name (STR aname)
{       ROSE_PUT_PRIM (STR,PERSISTENT_name,aname); }

/* desc Access Methods */
STR desc()
{       return ROSE_GET_PRIM (STR,PERSISTENT_desc);
}
void desc (STR adesc)
{       ROSE_PUT_PRIM (STR,PERSISTENT_desc,adesc); }

/* resources Access Methods */
ListOfResourceUtilization * resources();
void resources (ListOfResourceUtilization *  aresources)
{       ROSE_PUT_OBJ (ListOfResourceUtilization,PERSISTENT_resources,aresources); }

/* qualResults Access Methods */
Quality * qualResults()
{       return ROSE_GET_OBJ (Quality,PERSISTENT_qualResults);
}
void qualResults (Quality * aqualResults)
{       ROSE_PUT_OBJ (Quality,PERSISTENT_qualResults,aqualResults); }

/* indivRate Access Methods */
Cost * indivRate()
{       return ROSE_GET_OBJ (Cost,PERSISTENT_indivRate);
}
void indivRate (Cost * aindivRate)
{       ROSE_PUT_OBJ (Cost,PERSISTENT_indivRate,aindivRate); }

/* Constructors */
Process ();
Process (
        STR aname,
        STR adesc,
        ListOfResourceUtilization *  aresources,
        Quality *  aqualResults,
        Cost *  aindivRate );
```

```
/* CLASS DECLARATION EXTENSIONS */
/* Determines the Scrap and Rework Rates for the Process */
virtual void DetermineScrapRework(ListOfMfgSpec *);

/* Determine Total Process Cost */
virtual Cost *TotalRate(ListOfMfgSpec *);

/* Calculate Process Quality */
virtual void CalculateQuality(int);

/* Calculate Process Time/Cost rates */
virtual void CalculateRates();

/* Specifies if Features are complete at this Process */
virtual int CompleteFeatures();

/* Performs Deep Copy of Process */
virtual Process *CopyProcess(ListOfRoseObject *);

/* Display for Process */
virtual void Display();
};
#endif
```

4.2.4 Operation Specification

```
/* Class Declaration */
#ifndef Operation_h
#define Operation_h

#include "rose.h"
#include "process_model_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Operation.hi"

#include "Process.h"
ROSE_DECLARE (ListOfScrap);
ROSE_DECLARE (ListOfRework);
#define OperationOffsets(subClass) \
        ProcessOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Operation)

ROSE_DECLARE (Operation) : virtual public Process {
 private:
        LaborClass PERSISTENT_optype;
        ListOfScrap *  PERSISTENT_scrap_rate;
        ListOfRework *  PERSISTENT_rework_rate;

 public:
        ROSE_DECLARE_MEMBERS(Operation);

/* Access and Update Methods */
```

```
/* optype Access Methods */
LaborClass optype()
{       return ROSE_GET_PRIM (LaborClass,PERSISTENT_optype);
}
void optype (LaborClass aoptype)
{       ROSE_PUT_PRIM (LaborClass,PERSISTENT_optype,aoptype); }

/* scrap_rate Access Methods */
ListOfScrap * scrap_rate();
void scrap_rate (ListOfScrap *  ascrap_rate)
{       ROSE_PUT_OBJ (ListOfScrap,PERSISTENT_scrap_rate,ascrap_rate); }

/* rework_rate Access Methods */
ListOfRework * rework_rate();
void rework_rate (ListOfRework *  arework_rate)
{       ROSE_PUT_OBJ (ListOfRework,PERSISTENT_rework_rate,arework_rate); }

/* Constructors */
Operation ();
Operation (
        STR aname,
        STR adesc,
        ListOfResourceUtilization *  aresources,
        Quality *  aqualResults,
        Cost *  aindivRate,
        LaborClass aoptype,
        ListOfScrap *  ascrap_rate,
        ListOfRework *  arework_rate );

/* CLASS DECLARATION EXTENSIONS */
/* Determine Operation Scrap and Rework Values */
void DetermineScrapRework(ListOfMfgSpec *);

/* Calculate Production Qty */
void CalculateQuality(int);

/* Return if Features are complete at this operation */
int CompleteFeatures();

/* Perform deep copy of the operation */
Process *CopyProcess(ListOfRoseObject *);

/* Display Operation data */
void Display();
};
#endif
```

4.2.5 Step Specification

```
/* Class Declaration */
#ifndef Step_h
#define Step_h

#include "rose.h"
```

```
#include "process_model_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Step.hi"

#include "Process.h"
#define StepOffsets(subClass)\
        ProcessOffsets(subClass)\
        ROSE_SUPERCLASS_OFFSET(subClass,Step)

ROSE_DECLARE (Step) : virtual public Process {
 private:
        StepTypes PERSISTENT_stepType;

 public:
        ROSE_DECLARE_MEMBERS(Step);

/* Access and Update Methods */

/* stepType Access Methods */
StepTypes stepType()
{       return ROSE_GET_PRIM (StepTypes,PERSISTENT_stepType);
}
void stepType (StepTypes astepType)
{       ROSE_PUT_PRIM (StepTypes,PERSISTENT_stepType,astepType); }

/* Constructors */
Step ();
Step (
        STR aname,
        STR adesc,
        ListOfResourceUtilization *  aresources,
        Quality *  aqualResults,
        Cost *  aindivRate,
        StepTypes astepType );

/* CLASS DECLARATION EXTENSIONS */
/* Determine Operation Scrap and Rework Values */
void DetermineScrapRework(ListOfMfgSpec *);

/* Calculate Production Qty */
void CalculateQuality(int);

/* Return if Features are complete at this operation */
int CompleteFeatures();

/* Perform deep copy of the operation */
Process *CopyProcess(ListOfRoseObject *);

/* Display Operation data */
void Display();
};
#endif
```

4.2.6 Quality Specification

```
/* Class Declaration */
#ifndef Quality_h
#define Quality_h

#include "rose.h"
#include "process_model_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Quality.hi"

#define QualityOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Quality)

ROSE_DECLARE (Quality) : virtual public RoseStructure {
 private:
        float PERSISTENT_scrapPercent;
        int PERSISTENT_prodQty;
        float PERSISTENT_reworkPercent;
        float PERSISTENT_reworkCost;

 public:
        ROSE_DECLARE_MEMBERS(Quality);

/* Access and Update Methods */

/* scrapPercent Access Methods */
float scrapPercent()
{       return ROSE_GET_PRIM (float,PERSISTENT_scrapPercent);
}
void scrapPercent (float ascrapPercent)
{       ROSE_PUT_PRIM (float,PERSISTENT_scrapPercent,ascrapPercent); }

/* prodQty Access Methods */
int prodQty()
{       return ROSE_GET_PRIM (int,PERSISTENT_prodQty);
}
void prodQty (int aprodQty)
{       ROSE_PUT_PRIM (int,PERSISTENT_prodQty,aprodQty); }

/* reworkPercent Access Methods */
float reworkPercent()
{       return ROSE_GET_PRIM (float,PERSISTENT_reworkPercent);
}
void reworkPercent (float areworkPercent)
{       ROSE_PUT_PRIM (float,PERSISTENT_reworkPercent,areworkPercent); }

/* reworkCost Access Methods */
float reworkCost()
{       return ROSE_GET_PRIM (float,PERSISTENT_reworkCost);
}
```

```
void reworkCost (float areworkCost)
{       ROSE_PUT_PRIM (float,PERSISTENT_reworkCost,areworkCost); }

/* Constructors */
Quality ();
Quality (
        float ascrapPercent,
        int aprodQty,
        float areworkPercent,
        float areworkCost );

/* CLASS DECLARATION EXTENSIONS */
Quality *AddQuality();
};
#endif
```

4.2.7 Scrap Specification

```
/* Class Declaration */
#ifndef Scrap_h
#define Scrap_h

#include "rose.h"
#include "process_model_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Scrap.hi"

ROSE_DECLARE (Rules);
ROSE_DECLARE (Equation);
#define ScrapOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Scrap)

ROSE_DECLARE (Scrap) : virtual public RoseStructure {
 private:
        Rules *  PERSISTENT_scrapRule;
        Equation *  PERSISTENT_scrapRate;
        float PERSISTENT_scrapPercentage;

 public:
        ROSE_DECLARE_MEMBERS(Scrap);

/* Access and Update Methods */

/* scrapRule Access Methods */
Rules * scrapRule()
{       return ROSE_GET_OBJ (Rules,PERSISTENT_scrapRule);
}
void scrapRule (Rules * ascrapRule)
{       ROSE_PUT_OBJ (Rules,PERSISTENT_scrapRule,ascrapRule); }
```

```
/* scrapRate Access Methods */
Equation * scrapRate()
  { return ROSE_GET_OBJ (Equation,PERSISTENT_scrapRate); }
void scrapRate (Equation * ascrapRate)
{       ROSE_PUT_OBJ(Equation,PERSISTENT_scrapRate,ascrapRate); }

/* scrapPercentage Access Methods */
float scrapPercentage()
{       return ROSE_GET_PRIM (float,PERSISTENT_scrapPercentage);
}
void scrapPercentage (float ascrapPercentage)
{       ROSE_PUT_PRIM (float,PERSISTENT_scrapPercentage,ascrapPercentage); }

/* Constructors */
Scrap ();
Scrap (
        Rules *  ascrapRule,
        Equation *  ascrapRate,
        float ascrapPercentage );

/* CLASS DECLARATION EXTENSIONS */
/* Deep Copy the Scrap Object */
Scrap *CopyScrap(ListOfRoseObject *);

/* Determine if Scrap rule should be Selected for the part under analysis */
BOOL Select(ListOfRoseObject *);
};
#endif
```

4.2.8 Rework Specification

```
/* Class Declaration */
#ifndef Rework_h
#define Rework_h

#include "rose.h"
#include "process_model_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Rework.hi"

ROSE_DECLARE (Rules);
ROSE_DECLARE (Equation);
ROSE_DECLARE (ListOfResourceUtilization);
#define ReworkOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Rework)

ROSE_DECLARE (Rework) : virtual public RoseStructure {
 private:
        Rules *  PERSISTENT_reworkRule;
        Equation *  PERSISTENT_reworkRate;
        ListOfResourceUtilization *  PERSISTENT_resources;
```

```
        float PERSISTENT_reworkPercentage;
        float PERSISTENT_reworkCost;

 public:
        ROSE_DECLARE_MEMBERS(Rework);

/* Access and Update Methods */

/* reworkRule Access Methods */
Rules * reworkRule()
{       return ROSE_GET_OBJ (Rules,PERSISTENT_reworkRule);
}
void reworkRule (Rules * areworkRule)
{       ROSE_PUT_OBJ (Rules,PERSISTENT_reworkRule,areworkRule); }

/* reworkRate Access Methods */
Equation * reworkRate()
  { return ROSE_GET_OBJ (Equation,PERSISTENT_reworkRate); }
void reworkRate (Equation * areworkRate)
{       ROSE_PUT_OBJ(Equation,PERSISTENT_reworkRate,areworkRate); }

/* resources Access Methods */
ListOfResourceUtilization * resources();
void resources (ListOfResourceUtilization *  aresources)
{       ROSE_PUT_OBJ (ListOfResourceUtilization,PERSISTENT_resources,aresources); }

/* reworkPercentage Access Methods */
float reworkPercentage()
{       return ROSE_GET_PRIM (float,PERSISTENT_reworkPercentage);
}
void reworkPercentage (float areworkPercentage)
{       ROSE_PUT_PRIM (float,PERSISTENT_reworkPercentage,areworkPercentage); }

/* reworkCost Access Methods */
float reworkCost()
{       return ROSE_GET_PRIM (float,PERSISTENT_reworkCost);
}
void reworkCost (float areworkCost)
{       ROSE_PUT_PRIM (float,PERSISTENT_reworkCost,areworkCost); }

/* Constructors */
Rework ();
Rework (
        Rules *  areworkRule,
        Equation *  areworkRate,
        ListOfResourceUtilization *  aresources,
        float areworkPercentage,
        float areworkCost );

/* CLASS DECLARATION EXTENSIONS */
/* Deep Copy Rework Object */
Rework *CopyRework(ListOfRoseObject *);

/* Determine if Rework Rule is Applicable for this part */
```

```
BOOL Select(ListOfRoseObject *partFeatures);
};
#endif
```

4.2.9 Cost Specification

```
/* Class Declaration */
#ifndef Cost_h
#define Cost_h

#include "rose.h"
#include "process_model_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Cost.hi"

#define CostOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Cost)

ROSE_DECLARE (Cost) : virtual public RoseStructure {
 private:
        float PERSISTENT_setupTime;
        float PERSISTENT_runTime;
        float PERSISTENT_idealTime;
        float PERSISTENT_idealCost;
        float PERSISTENT_actualTime;
        float PERSISTENT_actualCost;

 public:
        ROSE_DECLARE_MEMBERS(Cost);

/* Access and Update Methods */

/* setupTime Access Methods */
float setupTime()
{       return ROSE_GET_PRIM (float,PERSISTENT_setupTime);
}
void setupTime (float asetupTime)
{       ROSE_PUT_PRIM (float,PERSISTENT_setupTime,asetupTime); }

/* runTime Access Methods */
float runTime()
{       return ROSE_GET_PRIM (float,PERSISTENT_runTime);
}
void runTime (float arunTime)
{       ROSE_PUT_PRIM (float,PERSISTENT_runTime,arunTime); }

/* idealTime Access Methods */
float idealTime()
{       return ROSE_GET_PRIM (float,PERSISTENT_idealTime);
}
```

```
void idealTime (float aidealTime)
{       ROSE_PUT_PRIM (float,PERSISTENT_idealTime,aidealTime); }

/* idealCost Access Methods */
float idealCost()
{       return ROSE_GET_PRIM (float,PERSISTENT_idealCost);
}
void idealCost (float aidealCost)
{       ROSE_PUT_PRIM (float,PERSISTENT_idealCost,aidealCost); }

/* actualTime Access Methods */
float actualTime()
{       return ROSE_GET_PRIM (float,PERSISTENT_actualTime);
}
void actualTime (float aactualTime)
{       ROSE_PUT_PRIM (float,PERSISTENT_actualTime,aactualTime); }

/* actualCost Access Methods */
float actualCost()
{       return ROSE_GET_PRIM (float,PERSISTENT_actualCost);
}
void actualCost (float aactualCost)
{       ROSE_PUT_PRIM (float,PERSISTENT_actualCost,aactualCost); }

/* Constructors */
Cost ();
Cost (
        float asetupTime,
        float arunTime,
        float aidealTime,
        float aidealCost,
        float aactualTime,
        float aactualCost );

/* CLASS DECLARATION EXTENSIONS */
Cost *AddCost();
};
#endif
```

4.2.10 ResourceUtilization Specification

```
/* Class Declaration */
#ifndef ResourceUtilization_h
#define ResourceUtilization_h

#include "rose.h"
#include "resource_schema_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "ResourceUtilization.hi"

ROSE_DECLARE (Resource);
```

```
ROSE_DECLARE (Equation);
ROSE_DECLARE (ResourceRates);
#define ResourceUtilizationOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,ResourceUtilization)

ROSE_DECLARE (ResourceUtilization) : virtual public RoseStructure {
 private:
        Resource *  PERSISTENT_resource;
        Equation *  PERSISTENT_setupTime;
        Equation *  PERSISTENT_runTime;
        float PERSISTENT_effRate;  /* OPTIONAL */
        ResourceRates *  PERSISTENT_rate;

 public:
        ROSE_DECLARE_MEMBERS(ResourceUtilization);

/* Access and Update Methods */

/* resource Access Methods */
Resource * resource()
{       return ROSE_GET_OBJ (Resource,PERSISTENT_resource);
}
void resource (Resource * aresource)
{       ROSE_PUT_OBJ (Resource,PERSISTENT_resource,aresource); }

/* setupTime Access Methods */
Equation * setupTime()
  { return ROSE_GET_OBJ (Equation,PERSISTENT_setupTime); }
void setupTime (Equation * asetupTime)
{       ROSE_PUT_OBJ(Equation,PERSISTENT_setupTime,asetupTime); }

/* runTime Access Methods */
Equation * runTime()
  { return ROSE_GET_OBJ (Equation,PERSISTENT_runTime); }
void runTime (Equation * arunTime)
{       ROSE_PUT_OBJ(Equation,PERSISTENT_runTime,arunTime); }

/* effRate Access Methods */
float effRate()
{       return ROSE_GET_PRIM (float,PERSISTENT_effRate);
}
void effRate (float aeffRate)
{       ROSE_PUT_PRIM (float,PERSISTENT_effRate,aeffRate); }

/* rate Access Methods */
ResourceRates * rate()
{       return ROSE_GET_OBJ (ResourceRates,PERSISTENT_rate);
}
void rate (ResourceRates * arate)
{       ROSE_PUT_OBJ (ResourceRates,PERSISTENT_rate,arate); }

/* Constructors */
```

```
ResourceUtilization ();
ResourceUtilization (
        Resource * aresource,
        Equation * asetupTime,
        Equation * arunTime,
        float aeffRate,
        ResourceRates * arate );

/* CLASS DECLARATION EXTENSIONS */
/* Deep Copy the ResourceUtilization Object */
ResourceUtilization * AddResourceUtilization(ListOfRoseObject *);

/* Calculatge Resource Rates */
void CalculateResourceRates(ListOfRoseObject *);
};
#endif
```

4.2.11 Parameter Specification

```
/* Class Declaration */
#ifndef Parameter_h
#define Parameter_h

#include "rose.h"
#include "resource_schema_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */

#define ParameterOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Parameter)

ROSE_DECLARE (Parameter) : virtual public RoseStructure {
 private:
        STR PERSISTENT_p_name;
        STR PERSISTENT_p_value;

 public:
        ROSE_DECLARE_MEMBERS(Parameter);

/* Access and Update Methods */

/* p_name Access Methods */
STR p_name()
{       return ROSE_GET_PRIM (STR,PERSISTENT_p_name);
}
void p_name (STR ap_name)
{       ROSE_PUT_PRIM (STR,PERSISTENT_p_name,ap_name); }

/* p_value Access Methods */
STR p_value()
{       return ROSE_GET_PRIM (STR,PERSISTENT_p_value);
}
```

```
void p_value (STR ap_value)
{       ROSE_PUT_PRIM (STR,PERSISTENT_p_value,ap_value); }

/* Constructors */
Parameter ();
Parameter (
        STR ap_name,
        STR ap_value );

/* CLASS DECLARATION EXTENSIONS */
};
#endif
```

4.2.12 ResourceRates Specification

```
/* Class Declaration */
#ifndef ResourceRates_h
#define ResourceRates_h

#include "rose.h"
#include "resource_schema_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "ResourceRates.hi"

#define ResourceRatesOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,ResourceRates)

ROSE_DECLARE (ResourceRates) : virtual public RoseStructure {
 private:
        float PERSISTENT_setupTime;
        float PERSISTENT_runTime;
        float PERSISTENT_idealTime;
        float PERSISTENT_idealCost;

 public:
        ROSE_DECLARE_MEMBERS(ResourceRates);

/* Access and Update Methods */

/* setupTime Access Methods */
float setupTime()
{       return ROSE_GET_PRIM (float,PERSISTENT_setupTime);
}
void setupTime (float asetupTime)
{       ROSE_PUT_PRIM (float,PERSISTENT_setupTime,asetupTime); }

/* runTime Access Methods */
float runTime()
{       return ROSE_GET_PRIM (float,PERSISTENT_runTime);
}
```

```
void runTime (float arunTime)
{       ROSE_PUT_PRIM (float,PERSISTENT_runTime,arunTime); }

/* idealTime Access Methods */
float idealTime()
{       return ROSE_GET_PRIM (float,PERSISTENT_idealTime);
}
void idealTime (float aidealTime)
{       ROSE_PUT_PRIM (float,PERSISTENT_idealTime,aidealTime); }

/* idealCost Access Methods */
float idealCost()
{       return ROSE_GET_PRIM (float,PERSISTENT_idealCost);
}
void idealCost (float aidealCost)
{       ROSE_PUT_PRIM (float,PERSISTENT_idealCost,aidealCost); }

/* Constructors */
ResourceRates ();
ResourceRates (
        float asetupTime,
        float arunTime,
        float aidealTime,
        float aidealCost );

/* CLASS DECLARATION EXTENSIONS */
ResourceRates *AddRates();
};
#endif
```

4.2.13 Resource Specification

```
/* Class Declaration */
#ifndef Resource_h
#define Resource_h

#include "rose.h"
#include "resource_schema_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Resource.hi"

ROSE_DECLARE (ListOfParameter);
#define ResourceOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Resource)

ROSE_DECLARE (Resource) : virtual public RoseStructure {
 private:
        STR PERSISTENT_resource_name;
        STR PERSISTENT_resource_code;
        ListOfParameter *  PERSISTENT_parameters;
```

```
public:
        ROSE_DECLARE_MEMBERS(Resource);

/* Access and Update Methods */

/* resource_name Access Methods */
STR resource_name()
{       return ROSE_GET_PRIM (STR,PERSISTENT_resource_name);
}
void resource_name (STR aresource_name)
{       ROSE_PUT_PRIM (STR,PERSISTENT_resource_name,aresource_name); }

/* resource_code Access Methods */
STR resource_code()
{       return ROSE_GET_PRIM (STR,PERSISTENT_resource_code);
}
void resource_code (STR aresource_code)
{       ROSE_PUT_PRIM (STR,PERSISTENT_resource_code,aresource_code); }

/* parameters Access Methods */
ListOfParameter * parameters();
void parameters (ListOfParameter *  aparameters)
{       ROSE_PUT_OBJ (ListOfParameter,PERSISTENT_parameters,aparameters); }

/* Constructors */
Resource ();
Resource (
        STR aresource_name,
        STR aresource_code,
        ListOfParameter *  aparameters );

/* CLASS DECLARATION EXTENSIONS */
/* retrieve the resource rate */
virtual float getRate();
};
#endif
```

4.2.13.1 Equipment Specification

```
/* Class Declaration */
#ifndef Equipment_h
#define Equipment_h

#include "rose.h"
#include "resource_schema_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Equipment.hi"

#include "Resource.h"
#define EquipmentOffsets(subClass)\
```

```
        ResourceOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Equipment)

ROSE_DECLARE (Equipment) : virtual public Resource {
 private:
        STR PERSISTENT_equipmentCategory;
        float PERSISTENT_cost_per_time_unit;

 public:
        ROSE_DECLARE_MEMBERS(Equipment);

/* Access and Update Methods */

/* equipmentCategory Access Methods */
STR equipmentCategory()
{       return ROSE_GET_PRIM (STR,PERSISTENT_equipmentCategory);
}
void equipmentCategory (STR aequipmentCategory)
{       ROSE_PUT_PRIM (STR,PERSISTENT_equipmentCategory,aequipmentCategory); }

/* cost_per_time_unit Access Methods */
float cost_per_time_unit()
{       return ROSE_GET_PRIM (float,PERSISTENT_cost_per_time_unit);
}
void cost_per_time_unit (float acost_per_time_unit)
{       ROSE_PUT_PRIM (float,PERSISTENT_cost_per_time_unit,acost_per_time_unit); }

/* Constructors */
Equipment ();
Equipment (
        STR aresource_name,
        STR aresource_code,
        ListOfParameter *  aparameters,
        STR aequipmentCategory,
        float acost_per_time_unit );

/* CLASS DECLARATION EXTENSIONS */
float getRate();
};
#endif
```

4.2.13.2 ConsumableMaterial Specification

```
/* Class Declaration */
#ifndef ConsumableMaterial_h
#define ConsumableMaterial_h

#include "rose.h"
#include "resource_schema_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "ConsumableMaterial.hi"
```

```
#include "Resource.h"
ROSE_DECLARE (ListOfResourceConsumable);
#define ConsumableMaterialOffsets(subClass) \
        ResourceOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,ConsumableMaterial)

ROSE_DECLARE (ConsumableMaterial) : virtual public Resource {
 private:
        float PERSISTENT_cost_per_unit;
        ListOfResourceConsumable *  PERSISTENT_resourceRates;

 public:
        ROSE_DECLARE_MEMBERS(ConsumableMaterial);

/* Access and Update Methods */

/* cost_per_unit Access Methods */
float cost_per_unit()
{       return ROSE_GET_PRIM (float,PERSISTENT_cost_per_unit);
}
void cost_per_unit (float acost_per_unit)
{       ROSE_PUT_PRIM (float,PERSISTENT_cost_per_unit,acost_per_unit); }

/* resourceRates Access Methods */
ListOfResourceConsumable * resourceRates();
void resourceRates (ListOfResourceConsumable *  aresourceRates)
{       ROSE_PUT_OBJ
(ListOfResourceConsumable,PERSISTENT_resourceRates,aresourceRates); }

/* Constructors */
ConsumableMaterial ();
ConsumableMaterial (
        STR aresource_name,
        STR aresource_code,
        ListOfParameter *  aparameters,
        float acost_per_unit,
        ListOfResourceConsumable *  aresourceRates );

/* CLASS DECLARATION EXTENSIONS */
float getRate();
};
#endif
```

4.2.13.3 ResourceConsumable Specification

```
/* Class Declaration */
#ifndef ResourceConsumable_h
#define ResourceConsumable_h

#include "rose.h"
#include "resource_schema_types.h"
```

```
/* CLASS INCLUDE-FILE EXTENSIONS */
#include "ResourceConsumable.hi"

ROSE_DECLARE (Resource);
#define ResourceConsumableOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,ResourceConsumable)

ROSE_DECLARE (ResourceConsumable) : virtual public RoseStructure {
 private:
        Resource *  PERSISTENT_aresource;
        float PERSISTENT_units_exhausted_per_time_unit;

 public:
        ROSE_DECLARE_MEMBERS(ResourceConsumable);

/* Access and Update Methods */

/* aresource Access Methods */
Resource * aresource()
{       return ROSE_GET_OBJ (Resource,PERSISTENT_aresource);
}
void aresource (Resource * aaresource)
{       ROSE_PUT_OBJ (Resource,PERSISTENT_aresource,aaresource); }

/* units_exhausted_per_time_unit Access Methods */
float units_exhausted_per_time_unit()
{       return ROSE_GET_PRIM (float,PERSISTENT_units_exhausted_per_time_unit);
}
void units_exhausted_per_time_unit (float aunits_exhausted_per_time_unit)
{       ROSE_PUT_PRIM
(float,PERSISTENT_units_exhausted_per_time_unit,aunits_exhausted_per_time_unit); }

/* Constructors */
ResourceConsumable ();
ResourceConsumable (
        Resource *  aaresource,
        float aunits_exhausted_per_time_unit );

/* CLASS DECLARATION EXTENSIONS */
float getUnitsConsumed();
};
#endif
```

4.2.13.4 Labor Specification

```
/* Class Declaration */
#ifndef Labor_h
#define Labor_h

#include "rose.h"
```

```
#include "resource_schema_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Labor.hi"

#include "Resource.h"
#define LaborOffsets(subClass) \
        ResourceOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Labor)

ROSE_DECLARE (Labor) : virtual public Resource {
 private:
        STR PERSISTENT_job_code;
        LaborClass PERSISTENT_l_type;
        float PERSISTENT_rate;

 public:
        ROSE_DECLARE_MEMBERS(Labor);

/* Access and Update Methods */

/* job_code Access Methods */
STR job_code()
{       return ROSE_GET_PRIM (STR,PERSISTENT_job_code);
}
void job_code (STR ajob_code)
{       ROSE_PUT_PRIM (STR,PERSISTENT_job_code,ajob_code); }

/* l_type Access Methods */
LaborClass l_type()
{       return ROSE_GET_PRIM (LaborClass,PERSISTENT_l_type);
}
void l_type (LaborClass al_type)
{       ROSE_PUT_PRIM (LaborClass,PERSISTENT_l_type,al_type); }

/* rate Access Methods */
float rate()
{       return ROSE_GET_PRIM (float,PERSISTENT_rate);
}
void rate (float arate)
{       ROSE_PUT_PRIM (float,PERSISTENT_rate,arate); }

/* Constructors */
Labor ();
Labor (
        STR aresource_name,
        STR aresource_code,
        ListOfParameter *  aparameters,
        STR ajob_code,
        LaborClass al_type,
        float arate );

/* CLASS DECLARATION EXTENSIONS */
```

```
float getRate();
};
#endif
```

4.2.13.5 Facility Specification

```
/* Class Declaration */
#ifndef Facility_h
#define Facility_h

#include "rose.h"
#include "resource_schema_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Facility.hi"

#include "Resource.h"
#define FacilityOffsets(subClass) \
        ResourceOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Facility)

ROSE_DECLARE (Facility) : virtual public Resource {
 private:
        float PERSISTENT_square_feet_allocated;
        float PERSISTENT_cost_per_sq_ft_per_time_unit;

 public:
        ROSE_DECLARE_MEMBERS(Facility);

/* Access and Update Methods */

/* square_feet_allocated Access Methods */
float square_feet_allocated()
{       return ROSE_GET_PRIM (float,PERSISTENT_square_feet_allocated);
}
void square_feet_allocated (float asquare_feet_allocated)
{       ROSE_PUT_PRIM
(float,PERSISTENT_square_feet_allocated,asquare_feet_allocated); }

/* cost_per_sq_ft_per_time_unit Access Methods */
float cost_per_sq_ft_per_time_unit()
{       return ROSE_GET_PRIM (float,PERSISTENT_cost_per_sq_ft_per_time_unit);
}
void cost_per_sq_ft_per_time_unit (float acost_per_sq_ft_per_time_unit)
{       ROSE_PUT_PRIM
(float,PERSISTENT_cost_per_sq_ft_per_time_unit,acost_per_sq_ft_per_time_unit); }

/* Constructors */
Facility ();
Facility (
        STR aresource_name,
        STR aresource_code,
```

```
        ListOfParameter *  aparameters,
        float asquare_feet_allocated,
        float acost_per_sq_ft_per_time_unit );

/* CLASS DECLARATION EXTENSIONS */
float getRate();
};
#endif
```

4.2.14 ReasoningLogic Specification

```
/* Class Declaration */
#ifndef ReasoningLogic_h
#define ReasoningLogic_h

#include "rose.h"
#include "process_model_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "ReasoningLogic.hi"

ROSE_DECLARE (ListOfRules);
#define ReasoningLogicOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,ReasoningLogic)

ROSE_DECLARE (ReasoningLogic) : virtual public RoseStructure {
 private:
        ListOfRules *  PERSISTENT_rules;

 public:
        ROSE_DECLARE_MEMBERS(ReasoningLogic);

/* Access and Update Methods */

/* rules Access Methods */
ListOfRules * rules();
void rules (ListOfRules *  arules)
{       ROSE_PUT_OBJ (ListOfRules,PERSISTENT_rules,arules); }

/* Constructors */
ReasoningLogic ();
ReasoningLogic (
        ListOfRules *  arules );

/* CLASS DECLARATION EXTENSIONS */
/* Evaluate ReasoningLogic */
BOOL Evaluate(ProductEntities *);

/* Display ReasoningLogic Data */
void Display();
};
#endif
```

4.2.15 Rules Specification

```
/* Class Declaration */
#ifndef Rules_h
#define Rules_h

#include "rose.h"
#include "selection_rules_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Rules.hi"

ROSE_DECLARE (ListOfExpression);
#define RulesOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Rules)

ROSE_DECLARE (Rules) : virtual public RoseStructure {
 private:
        ListOfExpression *  PERSISTENT_exp1;
        BOOL PERSISTENT_moreRuleFiring;

 public:
        ROSE_DECLARE_MEMBERS(Rules);

/* Access and Update Methods */

/* exp1 Access Methods */
ListOfExpression * exp1();
void exp1 (ListOfExpression *  aexp1)
{       ROSE_PUT_OBJ (ListOfExpression,PERSISTENT_exp1,aexp1); }

/* moreRuleFiring Access Methods */
BOOL moreRuleFiring()
{       return ROSE_GET_PRIM (BOOL,PERSISTENT_moreRuleFiring);
}
void moreRuleFiring (BOOL amoreRuleFiring)
{       ROSE_PUT_PRIM (BOOL,PERSISTENT_moreRuleFiring,amoreRuleFiring); }

/* Constructors */
Rules ();
Rules (
        ListOfExpression *  aexp1,
        BOOL amoreRuleFiring );

/* CLASS DECLARATION EXTENSIONS */
/* Evaluate Rules */
BOOL Evaluate(ProductEntities *, ListOfRoseObject *);

/* Display Rules */
void Display();
};
```

```
#endif
```

4.2.16 Expression Specification

```
/* Class Declaration */
#ifndef Expression_h
#define Expression_h


/* Class Expression */
#include "rose.h"
#include "selection_rules_types.h"
ROSE_DECLARE (Equation);
ROSE_DECLARE (ComplexExp);
ROSE_DECLARE (SimpleExp);
ROSE_DECLARE (StringValue);

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Expression.hi"

#define ExpressionOffsets(subClass) \
        RoseUnionOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Expression)

ROSE_DECLARE (Expression) : public RoseUnion  {
 public:

ROSE_DECLARE_MEMBERS(Expression);

/* Access and Update Methods */
BOOL is_Equation()
{       return (getAttribute() == getAttribute("_Equation"));
}
Equation *  _Equation()
{       return ROSE_GET_OBJ (Equation,PERSISTENT_data.value.aPtr); }

void _Equation (Equation *  a_Equation)
{       this->putAttribute("_Equation");
        if (!ROSE.error())
                ROSE_PUT_OBJ(Equation,PERSISTENT_data.value.aPtr,a_Equation); }

BOOL is_ComplexExp()
{       return (getAttribute() == getAttribute("_ComplexExp"));
}
ComplexExp *  _ComplexExp()
{       return ROSE_GET_OBJ (ComplexExp,PERSISTENT_data.value.aPtr); }

void _ComplexExp (ComplexExp *  a_ComplexExp)
{       this->putAttribute("_ComplexExp");
        if (!ROSE.error())

        ROSE_PUT_OBJ(ComplexExp,PERSISTENT_data.value.aPtr,a_ComplexExp); }
```

```
BOOL is_SimpleExp()
{       return (getAttribute() == getAttribute("_SimpleExp"));
}
SimpleExp *  _SimpleExp()
{       return ROSE_GET_OBJ (SimpleExp,PERSISTENT_data.value.aPtr); }

void _SimpleExp (SimpleExp *  a_SimpleExp)
{       this->putAttribute("_SimpleExp");
        if (!ROSE.error())
                ROSE_PUT_OBJ(SimpleExp,PERSISTENT_data.value.aPtr,a_SimpleExp); }

BOOL is_StringValue()
{       return (getAttribute() == getAttribute("_StringValue"));
}
StringValue *  _StringValue()
{       return ROSE_GET_OBJ (StringValue,PERSISTENT_data.value.aPtr); }

void _StringValue (StringValue *  a_StringValue)
{       this->putAttribute("_StringValue");
        if (!ROSE.error())
                ROSE_PUT_OBJ(StringValue,PERSISTENT_data.value.aPtr,a_StringValue);
}

/* Constructor */

Expression ();

/* CLASS DECLARATION EXTENSIONS */
/* Evaluate Expression */
TokenReturnValue Evaluate(BOOL &, ProductEntities *, ListOfRoseObject *);
/* Display Expression */
void Display();
};
#endif
```

4.2.17 ComplexExp Specification

```
/* Class Declaration */
#ifndef ComplexExp_h
#define ComplexExp_h

#include "rose.h"
#include "selection_rules_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "ComplexExp.hi"

ROSE_DECLARE (Equation);
ROSE_DECLARE (Expression);
#define ComplexExpOffsets(subClass) \
        RoseStructureOffsets(subClass) \
```

```
        ROSE_SUPERCLASS_OFFSET(subClass,ComplexExp)

ROSE_DECLARE (ComplexExp) : virtual public RoseStructure {
 private:
        Equation *  PERSISTENT_Equ1;
        Equiv_Op PERSISTENT_EquivOp1;
        Expression *  PERSISTENT_Exp1;

 public:
        ROSE_DECLARE_MEMBERS(ComplexExp);

/* Access and Update Methods */

/* Equ1 Access Methods */
Equation * Equ1()
  { return ROSE_GET_OBJ (Equation,PERSISTENT_Equ1); }
void Equ1 (Equation * aEqu1)
{       ROSE_PUT_OBJ(Equation,PERSISTENT_Equ1,aEqu1); }

/* EquivOp1 Access Methods */
Equiv_Op EquivOp1()
{       return ROSE_GET_PRIM (Equiv_Op,PERSISTENT_EquivOp1);
}
void EquivOp1 (Equiv_Op aEquivOp1)
{       ROSE_PUT_PRIM (Equiv_Op,PERSISTENT_EquivOp1,aEquivOp1); }

/* Exp1 Access Methods */
Expression * Exp1()
  { return ROSE_GET_OBJ (Expression,PERSISTENT_Exp1); }
void Exp1 (Expression * aExp1)
{       ROSE_PUT_OBJ(Expression,PERSISTENT_Exp1,aExp1); }

/* Constructors */
ComplexExp ();
ComplexExp (
        Equation *  aEqu1,
        Equiv_Op aEquivOp1,
        Expression *  aExp1 );

/* CLASS DECLARATION EXTENSIONS */
TokenReturnValue Evaluate(BOOL &, ProductEntities *, ListOfRoseObject *);
void Display();
};
#endif
```

4.2.18 SimpleExp Specification

```
/* Class Declaration */
#ifndef SimpleExp_h
#define SimpleExp_h

#include "rose.h"
```

```
#include "selection_rules_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "SimpleExp.hi"

ROSE_DECLARE (DataDictStr);
#define SimpleExpOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,SimpleExp)

ROSE_DECLARE (SimpleExp) : virtual public RoseStructure {
 private:
        Unary_Op PERSISTENT_Not1;
        DataDictStr *  PERSISTENT_DataDictVar;

 public:
        ROSE_DECLARE_MEMBERS(SimpleExp);

/* Access and Update Methods */

/* Not1 Access Methods */
Unary_Op Not1()
{       return ROSE_GET_PRIM (Unary_Op,PERSISTENT_Not1);
}
void Not1 (Unary_Op aNot1)
{       ROSE_PUT_PRIM (Unary_Op,PERSISTENT_Not1,aNot1); }

/* DataDictVar Access Methods */
DataDictStr * DataDictVar()
{       return ROSE_GET_OBJ (DataDictStr,PERSISTENT_DataDictVar);
}
void DataDictVar (DataDictStr * aDataDictVar)
{       ROSE_PUT_OBJ (DataDictStr,PERSISTENT_DataDictVar,aDataDictVar); }

/* Constructors */
SimpleExp ();
SimpleExp (
        Unary_Op aNot1,
        DataDictStr *  aDataDictVar );

/* CLASS DECLARATION EXTENSIONS */
TokenReturnValue Evaluate(BOOL &, ProductEntities *, ListOfRoseObject *);
void Display();
};
#endif
```

4.2.19 Equation Specification

```
/* Class Declaration */
#ifndef Equation_h
#define Equation_h
```

```
/* Class Equation */
#include "rose.h"
#include "selection_rules_types.h"
ROSE_DECLARE (Term);
ROSE_DECLARE (ComplexEquation);

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Equation.hi"

#define EquationOffsets(subClass) \
        RoseUnionOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Equation)

ROSE_DECLARE (Equation) : public RoseUnion  {
 public:

ROSE_DECLARE_MEMBERS(Equation);

/* Access and Update Methods */
BOOL is_Term()
{       return (getAttribute() == getAttribute("_Term"));
}
Term *  _Term()
{       return ROSE_GET_OBJ (Term,PERSISTENT_data.value.aPtr); }

void _Term (Term *  a_Term)
{       this->putAttribute("_Term");
        if (!ROSE.error())
                ROSE_PUT_OBJ(Term,PERSISTENT_ data.value.aPtr,a_Term); }

BOOL is_ComplexEquation()
{       return (getAttribute() == getAttribute("_ComplexEquation"));
}
ComplexEquation *  _ComplexEquation()
{       return ROSE_GET_OBJ (ComplexEquation,PERSISTENT_data.value.aPtr); }

void _ComplexEquation (ComplexEquation *  a_ComplexEquation)
{       this->putAttribute("_ComplexEquation");
        if (!ROSE.error())

        ROSE_PUT_OBJ(ComplexEquation,PERSISTENT_data.value.aPtr,a_ComplexEquati
on); }

/* Constructor */

Equation ();

/* CLASS DECLARATION EXTENSIONS */
TokenReturnValue Evaluate(BOOL &, ProductEntities *, ListOfRoseObject *);
void Display();
};
#endif
```

4.2.20 ComplexTerm Specification

```
/* Class Declaration */
#ifndef ComplexTerm_h
#define ComplexTerm_h

#include "rose.h"
#include "selection_rules_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "ComplexTerm.hi"

ROSE_DECLARE (Equation);
#define ComplexTermOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,ComplexTerm)

ROSE_DECLARE (ComplexTerm) : virtual public RoseStructure {
 private:
        Equation *  PERSISTENT_equ1;
        Mult_Div_Oper PERSISTENT_Oper1;
        Equation *  PERSISTENT_equ2;

 public:
        ROSE_DECLARE_MEMBERS(ComplexTerm);

/* Access and Update Methods */

/* equ1 Access Methods */
Equation * equ1()
  { return ROSE_GET_OBJ (Equation,PERSISTENT_equ1); }
void equ1 (Equation * aequ1)
{       ROSE_PUT_OBJ(Equation,PERSISTENT_equ1,aequ1); }

/* Oper1 Access Methods */
Mult_Div_Oper Oper1()
{       return ROSE_GET_PRIM (Mult_Div_Oper,PERSISTENT_Oper1);
}
void Oper1 (Mult_Div_Oper aOper1)
{       ROSE_PUT_PRIM (Mult_Div_Oper,PERSISTENT_Oper1,aOper1); }

/* equ2 Access Methods */
Equation * equ2()
  { return ROSE_GET_OBJ (Equation,PERSISTENT_equ2); }
void equ2 (Equation * aequ2)
{       ROSE_PUT_OBJ(Equation,PERSISTENT_equ2,aequ2); }

/* Constructors */
ComplexTerm ();
ComplexTerm (
        Equation *  aequ1,
        Mult_Div_Oper aOper1,
        Equation *  aequ2 );
```

```
/* CLASS DECLARATION EXTENSIONS */
TokenReturnValue Evaluate(BOOL &, ProductEntities *, ListOfRoseObject *);
void Display();
};
#endif
```

4.2.21 ComplexEquation Specification

```
/* Class Declaration */
#ifndef ComplexEquation_h
#define ComplexEquation_h

#include "rose.h"
#include "selection_rules_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "ComplexEquation.hi"

ROSE_DECLARE (Term);
ROSE_DECLARE (Equation);
#define ComplexEquationOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,ComplexEquation)

ROSE_DECLARE (ComplexEquation) : virtual public RoseStructure {
 private:
        Term *  PERSISTENT_Var1;
        Add_Sub_Oper PERSISTENT_Oper1;
        Equation *  PERSISTENT_Value;

 public:
        ROSE_DECLARE_MEMBERS(ComplexEquation);

/* Access and Update Methods */

/* Var1 Access Methods */
Term * Var1()
  { return ROSE_GET_OBJ (Term,PERSISTENT_Var1); }
void Var1 (Term * aVar1)
{       ROSE_PUT_OBJ(Term,PERSISTENT_Var1,aVar1); }

/* Oper1 Access Methods */
Add_Sub_Oper Oper1()
{       return ROSE_GET_PRIM (Add_Sub_Oper,PERSISTENT_Oper1);
}
void Oper1 (Add_Sub_Oper aOper1)
{       ROSE_PUT_PRIM (Add_Sub_Oper,PERSISTENT_Oper1,aOper1); }

/* Value Access Methods */
Equation * Value()
  { return ROSE_GET_OBJ (Equation,PERSISTENT_Value); }
```

```
void Value (Equation * aValue)
{       ROSE_PUT_OBJ(Equation,PERSISTENT_Value,aValue); }

/* Constructors */
ComplexEquation ();
ComplexEquation (
        Term *  aVar1,
        Add_Sub_Oper aOper1,
        Equation *  aValue );

/* CLASS DECLARATION EXTENSIONS */
TokenReturnValue Evaluate(BOOL &, ProductEntities *, ListOfRoseObject *);
void Display();
};
#endif
```

4.2.22 ParenEquation Specification

```
/* Class Declaration */
#ifndef ParenEquation_h
#define ParenEquation_h

#include "rose.h"
#include "selection_rules_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "ParenEquation.hi"

ROSE_DECLARE (Equation);
#define ParenEquationOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,ParenEquation)

ROSE_DECLARE (ParenEquation) : virtual public RoseStructure {
 private:
        LParen PERSISTENT_Lparenthesis;
        Equation *  PERSISTENT_Equ;
        RParen PERSISTENT_Rparenthesis;

 public:
        ROSE_DECLARE_MEMBERS(ParenEquation);

/* Access and Update Methods */

/* Lparenthesis Access Methods */
LParen Lparenthesis()
{       return ROSE_GET_PRIM (LParen,PERSISTENT_Lparenthesis);
}
void Lparenthesis (LParen aLparenthesis)
{       ROSE_PUT_PRIM (LParen,PERSISTENT_Lparenthesis,aLparenthesis); }

/* Equ Access Methods */
```

```
Equation * Equ()
  { return ROSE_GET_OBJ (Equation,PERSISTENT_Equ); }
void Equ (Equation * aEqu)
{       ROSE_PUT_OBJ(Equation,PERSISTENT_Equ,aEqu); }

/* Rparenthesis Access Methods */
RParen Rparenthesis()
{       return ROSE_GET_PRIM (RParen,PERSISTENT_Rparenthesis);
}
void Rparenthesis (RParen aRparenthesis)
{       ROSE_PUT_PRIM (RParen,PERSISTENT_Rparenthesis,aRparenthesis); }

/* Constructors */
ParenEquation ();
ParenEquation (
        LParen aLparenthesis,
        Equation *  aEqu,
        RParen aRparenthesis );

/* CLASS DECLARATION EXTENSIONS */
TokenReturnValue Evaluate(BOOL &, ProductEntities *, ListOfRoseObject *);
void Display();
};
#endif
```

4.2.23 Term Specification

```
/* Class Declaration */
#ifndef Term_h
#define Term_h


/* Class Term */
#include "rose.h"
#include "selection_rules_types.h"
ROSE_DECLARE (Const);
ROSE_DECLARE (DataDictStr);
ROSE_DECLARE (ParenEquation);
ROSE_DECLARE (ComplexTerm);

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Term.hi"

#define TermOffsets(subClass)\
        RoseUnionOffsets(subClass)\
        ROSE_SUPERCLASS_OFFSET(subClass,Term)

ROSE_DECLARE (Term) : public RoseUnion  {
 public:

ROSE_DECLARE_MEMBERS(Term);
```

```
/* Access and Update Methods */
BOOL is_Const()
{       return (getAttribute() == getAttribute("_Const"));
}
Const *  _Const()
{       return ROSE_GET_OBJ (Const,PERSISTENT_data.value.aPtr); }

void _Const (Const *  a_Const)
{       this->putAttribute("_Const");
        if (!ROSE.error())
                ROSE_PUT_OBJ(Const,PERSISTENT_data.value.aPtr,a_Const); }

BOOL is_DataDictStr()
{       return (getAttribute() == getAttribute("_DataDictStr"));
}
DataDictStr *  _DataDictStr()
{       return ROSE_GET_OBJ (DataDictStr,PERSISTENT_data.value.aPtr); }

void _DataDictStr (DataDictStr *  a_DataDictStr)
{       this->putAttribute("_DataDictStr");
        if (!ROSE.error())
                ROSE_PUT_OBJ(DataDictStr,PERSISTENT_data.value.aPtr,a_DataDictStr);
}

BOOL is_ParenEquation()
{       return (getAttribute() == getAttribute("_ParenEquation"));
}
ParenEquation *  _ParenEquation()
{       return ROSE_GET_OBJ (ParenEquation,PERSISTENT_data.value.aPtr); }

void _ParenEquation (ParenEquation *  a_ParenEquation)
{       this->putAttribute("_ParenEquation");
        if (!ROSE.error())

        ROSE_PUT_OBJ(ParenEquation,PERSISTENT_data.value.aPtr,a_ParenEquation); }

BOOL is_ComplexTerm()
{       return (getAttribute() == getAttribute("_ComplexTerm"));
}
ComplexTerm *  _ComplexTerm()
{       return ROSE_GET_OBJ (ComplexTerm,PERSISTENT_data.value.aPtr); }

void _ComplexTerm (ComplexTerm *  a_ComplexTerm)
{       this->putAttribute("_ComplexTerm");
        if (!ROSE.error())

        ROSE_PUT_OBJ(ComplexTerm,PERSISTENT_data.value.aPtr,a_ComplexTerm); }

/* Constructor */

Term ();

/* CLASS DECLARATION EXTENSIONS */
```

```
TokenReturnValue Evaluate(BOOL &, ProductEntities *, ListOfRoseObject *);
void Display();
};
#endif
```

4.2.24 Const Specification

```
/* Class Declaration */
#ifndef Const_h
#define Const_h


/* Class Const */
#include "rose.h"
#include "selection_rules_types.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Const.hi"

#define ConstOffsets(subClass) \
        RoseUnionOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Const)

ROSE_DECLARE (Const) : public RoseUnion  {
 public:

ROSE_DECLARE_MEMBERS(Const);

/* Access and Update Methods */
BOOL is_float()
{       return (getAttribute() == getAttribute("_float"));
}
float _float()
{       return (float) ROSE_GET_PRIM (float,PERSISTENT_data.value.aFloat); }

void _float (float a_float)
{       this->putAttribute("_float");
        if (!ROSE.error())
                ROSE_PUT_PRIM(float,PERSISTENT_data.value.aFloat,a_float); }

BOOL is_int()
{       return (getAttribute() == getAttribute("_int"));
}
int _int()
{       return (int) ROSE_GET_PRIM (int,PERSISTFNT_data.value.anInt); }

void _int (int a_int)
{       this->putAttribute("_int");
        if (!ROSE.error())
                ROSE_PUT_PRIM(int,PERSISTENT_data.value.anInt,a_int); }

/* Constructor */
```

```
Const ();

/* CLASS DECLARATION EXTENSIONS */
TokenReturnValue Evaluate();
void Display();
};
#endif
```

4.2.25 Addition/Subtraction Specification

```
/* Enumerated Type */
#ifndef Add_Sub_Oper_h
#define Add_Sub_Oper_h

#include "roseHdefs.h"
enum Add_Sub_Oper {
        Add_Sub_Oper_NULL = NULL_ENUM,
        Add_Sub_Oper_Add = 0,
        Add_Sub_Oper_Subtract
};
ROSE_DECLARE_PRIM (Add_Sub_Oper);
#endif
```

4.2.26 Multiplication/Division Specification

```
/* Enumerated Type */
#ifndef Mult_Div_Oper_h
#define Mult_Div_Oper_h

#include "roseHdefs.h"
enum Mult_Div_Oper {
        Mult_Div_Oper_NULL = NULL_ENUM,
        Mult_Div_Oper_Multiply = 0,
        Mult_Div_Oper_Divide
};
ROSE_DECLARE_PRIM (Mult_Div_Oper);
#endif
```

4.2.27 Unary_Op Specification

```
/* Enumerated Type */
enum Unary_Op {
        Unary_Op_NULL = NULL_ENUM,
        Unary_Op_U_Op = 0
};
```

4.2.28 Equiv_Op Specification

```
/* Enumerated Type */
enum Equiv_Op {
```

```
        Equiv_Op_NULL = NULL_ENUM,
        Equiv_Op_Less = 0,
        Equiv_Op_LessEqual,
        Equiv_Op_Greater,
        Equiv_Op_GreaterEqual,
        Equiv_Op_Equal,
        Equiv_Op_NotEqual
};
```

4.2.29 StringValue Specification

```
/* Class Declaration */
ROSE_DECLARE (StringValue) : virtual public RoseStructure {
 private:
        DQuote PERSISTENT_quote1;
        STR PERSISTENT_value1;
        DQuote PERSISTENT_quote2;

 public:
        ROSE_DECLARE_MEMBERS(StringValue);

/* Access and Update Methods */
/* quote1 Access Methods */
DQuote quote1()
{       return ROSE_GET_PRIM (DQuote,PERSISTENT_quote1);
}
void quote1 (DQuote aquote1)
{       ROSE_PUT_PRIM (DQuote,PERSISTENT_quote1,aquote1); }

/* value1 Access Methods */
STR value1()
{       return ROSE_GET_PRIM (STR,PERSISTENT_value1);
}
void value1 (STR avalue1)
{       ROSE_PUT_PRIM (STR,PERSISTENT_value1,avalue1); }

/* quote2 Access Methods */
DQuote quote2()
{       return ROSE_GET_PRIM (DQuote,PERSISTENT_quote2);
}
void quote2 (DQuote aquote2)
{       ROSE_PUT_PRIM (DQuote,PERSISTENT_quote2,aquote2); }

/* Constructors */
StringValue ();
StringValue (
        DQuote aquote1,
        STR avalue1,
        DQuote aquote2 );
};

/* Methods Implementation */
```

```
StringValue::StringValue () {
        PERSISTENT_quote1 = (DQuote) NULL_ENUM;
        PERSISTENT_value1 = NULL;
        PERSISTENT_quote2 = (DQuote) NULL_ENUM;
        ROSE_CTOR_EXTENSIONS;
}

StringValue::StringValue (
        DQuote aquote1,
        STR avalue1,
        DQuote aquote2 )
{
        quote1 (aquote1);
        value1 (avalue1);
        quote2 (aquote2);
        ROSE_CTOR_EXTENSIONS;
}
```

4.2.30 DataDictStr Specification

```
/* Abstract Base Class Declaration */
ROSE_DECLARE (DataDictStr) : virtual public RoseStructure {
 private:

 public:
        ROSE_DECLARE_MEMBERS(DataDictStr);

/* Access and Update Methods */
/* Constructors */
DataDictStr ();
};

/* Methods Implementation */
DataDictStr::DataDictStr () {
        ROSE_CTOR_EXTENSIONS;
}

/* CLASS EXTENSIONS */
virtual TokenReturnValue Evaluate(BOOL&, ProductEntities *, ListOfRoseObject *);
virtual void Display();
```

4.2.30.1 EntityName Specification

```
/* Class Declaration */
ROSE_DECLARE (EntityName) : virtual public DataDictStr {
 private:
        STR PERSISTENT_name;

 public:
        ROSE_DECLARE_MEMBERS(EntityName);

/* Access and Update Methods */
```

```
/* name Access Methods */
STR name()
{       return ROSE_GET_PRIM (STR,PERSISTENT_name);
}
void name (STR aname)
{       ROSE_PUT_PRIM (STR,PERSISTENT_name,aname); }

/* Constructors */
EntityName ();
EntityName (
        STR aname );
};

/* Methods Implementation */
EntityName::EntityName () {
        PERSISTENT_name = NULL;
        ROSE_CTOR_EXTENSIONS;
}

EntityName::EntityName (
        STR aname )
{
        name (aname);
        ROSE_CTOR_EXTENSIONS;
}
/* CLASS EXTENSIONS */
virtual TokenReturnValue Evaluate(BOOL&, ProductEntities *, ListOfRoseObject *);
virtual void Display();
```

4.2.30.2 EntityAttrName Specification

```
/* Class Declaration */
ROSE_DECLARE (EntityAttrName) : virtual public DataDictStr {
 private:
        ListOfString *  PERSISTENT_entityName;
        STR PERSISTENT_attrName;

 public:
        ROSE_DECLARE_MEMBERS(EntityAttrName);

/* Access and Update Methods */
/* entityName Access Methods */
ListOfString * entityName();
void entityName (ListOfString *  aentityName)
{       ROSE_PUT_OBJ (ListOfString,PERSISTENT_entityName,aentityName); }

/* attrName Access Methods */
STR attrName()
{       return ROSE_GET_PRIM (STR,PERSISTENT_attrName);
}
void attrName (STR aattrName)
{       ROSE_PUT_PRIM (STR,PERSISTENT_attrName,aattrName); }
```

```
/* Constructors */
EntityAttrName ();
EntityAttrName (
        ListOfString *  aentityName,
        STR aattrName );
};

/* Methods Implementation */
EntityAttrName::EntityAttrName () {
        PERSISTENT_entityName = NULL;
        PERSISTENT_attrName = NULL;
        ROSE_CTOR_EXTENSIONS;
}

EntityAttrName::EntityAttrName (
        ListOfString *  aentityName,
        STR aattrName )
{
        entityName (aentityName);
        attrName (aattrName);
        ROSE_CTOR_EXTENSIONS;
}

ListOfString * EntityAttrName :: entityName()
{       if( !PERSISTENT_entityName)
        if( this->isPersistent())
                entityName (pnewIn (design()) ListOfString);
        else    entityName (new ListOfString);
        return ROSE_GET_OBJ (ListOfString,PERSISTENT_entityName);
}
/* CLASS EXTENSIONS */
virtual TokenReturnValue Evaluate(BOOL&, ProductEntities *, ListOfRoseObject *);
virtual void Display();
```

4.3 Analyzer

The manufacturing Analyzer is a subsystem of MO which is responsible for performing the manufacturability analysis on a product database based on the selected process model. The Analyzer provides the user with the ability to perform a process selection, calculate yield and rework, and calculate time and cost. The Advisor uses the output of the Analyzer runs which it then displays to the user. Following is the corresponding specification and methods for the Analyzer class/object.

```
/* Class Specification */
#ifndef Analyzer_h
#define Analyzer_h
```

```
#include "rose.h"

/* CLASS INCLUDE-FILE EXTENSIONS */
#include "Analyzer.hi"

ROSE_DECLARE (ProcessModel);
#define AnalyzerOffsets(subClass) \
        RoseStructureOffsets(subClass) \
        ROSE_SUPERCLASS_OFFSET(subClass,Analyzer)

ROSE_DECLARE (Analyzer) : virtual public RoseStructure {
 private:
        STR PERSISTENT_productDesignName;
        ProcessModel *  PERSISTENT_pModel;
        ProcessModel *  PERSISTENT_plan;

 public:
        ROSE_DECLARE_MEMBERS(Analyzer);

/* Access and Update Methods */

/* productDesignName Access Methods */
STR productDesignName()
{       return ROSE_GET_PRIM (STR,PERSISTENT_productDesignName);
}
void productDesignName (STR aproductDesignName)
{       ROSE_PUT_PRIM (STR,PERSISTENT_productDesignName,aproductDesignName);
}

/* pModel Access Methods */
ProcessModel * pModel()
{       return ROSE_GET_OBJ (ProcessModel,PERSISTENT_pModel);
}
void pModel (ProcessModel * apModel)
{       ROSE_PUT_OBJ (ProcessModel,PERSISTENT_pModel,apModel); }

/* plan Access Methods */
ProcessModel * plan()
{       return ROSE_GET_OBJ (ProcessModel,PERSISTENT_plan);
}
void plan (ProcessModel * aplan)
{       ROSE_PUT_OBJ (ProcessModel,PERSISTENT_plan,aplan); }

/* Constructors */
Analyzer ();
Analyzer (
        STR aproductDesignName,
        ProcessModel *  apModel,
        ProcessModel *  aplan );
/* CLASS DECLARATION EXTENSIONS */
void PerformAnalysis();
};
#endif
```

4.4 Advisor

The Advisor is responsible for displaying the results produced by each process selected during an Analyzer run. The user can select analysis runs to view. The user can display process, yield, rework, or costing results as graphs, and can also view complete analysis data on the screen or send it to a file in report format.

The Advisor graphs are implemented using XRT/Graph for Motif widget which displays data graphically in a window. The graph widget has resources which determine how the graph will look and behave. We will be writing methods that will take the output results from the Analyzer subsystem.

The graph widget has resources which allow programmatic control of the following items:

- graph type (bar, stacked bar, line, and pie).
- header and footer positioning, border style, text, font, and color.
- data styles: line colors and patterns, fill color and patterns, line thickness, point style, size and color.
- legend positioning, orientation, border style, anchor, font and color.
- graph positioning, border style, color, width, height, and 3D effect.
- point and set labels.
- axis maximum and minimum, numbering increment, tick increment, grid increment, font, origin, and precision.
- window background and foreground color.
- text areas.
- double buffering.
- axis inversion.
- data transposition.
- marker positioning.

XRT/graph also provides several procedures and methods which allocate and load data structures containing the numbers to be graphed, output a representation of the graph in Postscript format, assist the developer in dealing with user-events, and assist the developer with setting and getting indexed resources.

4.5 Modeler

The process Modeler provides the ability to capture and modify manufacturing process models. The Modeler provides a graphical user interface where the user can capture process, operation, and step activities, as well as the corresponding selection rules and resources. The output of the Modeler is a ProcessModel object which is structured as a hierarchical tree of manufacturing activities. Each activity points to either process, operation, or step data. The ProcessModel object is used by the Analyzer and the Advisor to select the manufacturing processes that are used in the cost, yield, and rework calculations. Following is the corresponding specification and methods for the Modeler class/object.

```
/* Class Specification */
ROSE_DECLARE (Modeler) : virtual public RoseStructure {
 private:
        ProcessModel *  PERSISTENT_current_model;
 public:
            ROSE_DECLARE_MEMBERS(Modeler);

/* Access and Update Methods */
/* current_model Access Methods */
ProcessModel * current_model()
{           return ROSE_GET_OBJ (ProcessModel,PERSISTENT_current_model);
}
void current_model (ProcessModel * acurrent_model)
{           ROSE_PUT_OBJ (ProcessModel,PERSISTENT_current_model,acurrent_model); }
/* Constructors */
Modeler ();
Modeler (
            ProcessModel *  acurrent_model );
/* CLASS DECLARATION EXTENSIONS */
ProcessModel *readModel();
void writeModel();
};
```

5. Conclusions

During this reporting period, the main thrust was on the implementation of the MO software system. To demonstrate some of the work that was accomplished, technical highlights of the manufacturing process model EXPRESS schemas and the C++ class header files were included in this report.

Raytheon will continue developing, integrating, and testing the MO System during the next quarter.

6. References

1. BR-20558-1, 14 June 1991, DARPA Initiative In Concurrent Engineering (DICE) Manufacturing Optimization - Volume I - Technical.

2. CDRL No. 0002AC-1, March 1992, Operational Concept Document For The Manufacturing Optimization (MO) System, Contract No. MDA972-92-C-0020.

3. CDRL No. 0002AC-2, March 1992, Description of CE Technology For The Manufacturing Optimization (MO) System, Contract No. MDA972-92-C-0020.

4. CDRL No. 0002AC-3, May 1992, Functional Requirements and Measure of Performance For The Manufacturing Optimization (MO) System, Contract No. MDA972-92-C-0020.

5. CDRL No. 0002AC-4, December 1992, Software Design Specification For The Manufacturing Optimization (MO) System, Contract No. MDA972-92-C-0020.

6. STEP Programmer's Toolkit Reference Manual, STEP Tools Inc., 1992.

7. STEP Programmer's Toolkit Tutorial Manual, STEP Tools Inc., 1992.

8. STEP Utilities Reference Manual, STEP Tools Inc., 1992.

7. Notes

7.1 Acronyms

ASEM	Application Specific Electronic Module
CAEO	Computer Aided Engineering Operations
CDRL	Contract Data Requirements List
DARPA	Defense Advanced Research Projects Agency
DFMA	Design for Manufacturing and Assembly
DICE	DARPA Initiative In Concurrent Engineering
MO	Manufacturing Optimization
MSD	Missile Systems Division
MSL	Missile Systems Laboratories
OSF	Open Software Foundation
PWA	Printed Wiring Assembly
PWB	Printed Wiring Board
PWF	Printed Wiring Fabrication
ROSE	Rensselaer Object System For Engineering
STEP	Standard for Exchange of Product Model Data

Distribution List

DPRO-Raytheon
C/O Raytheon Company
Spencer Lab., Wayside Ave.
(one copy of each report)

Defense Advanced Research Projects Agency
ATTN: Defense Sciences Office; Dr. H. Lee Buchanan
Virginia Square Plaza
3701 N. Fairfax Drive
Arlington, VA. 22203-1714
(one copy of each report)

Defense Advanced Research Projects Agency
ATTN: Electronic Systems Technology Office; Capt. Nicholas J. Naclerio, USAF
Virginia Square Plaza
3701 N. Fairfax Drive
Arlington, VA. 22203-1714
(one copy of each report)

Defense Advanced Research Projects Agency
ATTN: Contracts Management Office; Mr. Donald C. Sharkus
Virginia Square Plaza
3701 N. Fairfax Drive
Arlington, VA. 22203-1714
(one copy of each report)

Defense Technical Information Center
Building 5, Cameron Station
ATTN: Selections
Alexandria, VA 22304
(two copies of each report)