Appendix A: Frame Class Models

This appendix lists the constraint-understanding system's frame class models, written in EXPRESS [34]:

- Section A.1 contains the frame class model for EXPRESS models (see Section 6.1);
- Section A.2 contains the frame class model for higher-level constraints (see Section 6.2);
- Section A.3 contains the frame class model for cases (see Section 6.3).

A.1 Frame Class Model of EXPRESS Models

```
(*******************************************************)
(** SCHEMA express_frame_class_model                 **)
(*******************************************************)
(** Frame class model of EXPRESS (an EXPRESS metamodel). **)
(*******************************************************)

SCHEMA express_frame_class_model;
USE FROM hlc_frame_class_model;

  (****************************************)
  (* ENTITY frame                         *)
  (****************************************)
  (* The superclass of all frame classes. *)
  (****************************************)

 ENTITY frame ABSTRACT SUPERTYPE;
 END_ENTITY;

  (*******************************************************)
  (* ENTITY named_type                                   *)
  (*******************************************************)
  (* Superclass of named types, namely ENTITYs and TYPEs.  *)
  (*******************************************************)

 ENTITY named_type ABSTRACT SUPERTYPE OF
                   (ONEOF (entity_type, defined_type))
                   SUBTYPE OF (frame);
   name : STRING;
   domain_rules : OPTIONAL SET [1 : ?] OF domain_rule;
 END_ENTITY;
```

```
(*****************************************************************)
(* ENTITY entity_type                                            *)
(*****************************************************************)
(* Represents an entity, which can be used anywhere a named type is *)
(* required. The attributes capture the information from the     *)
(* corresponding ENTITY definition.                              *)
(*****************************************************************)

ENTITY entity_type SUBTYPE OF (named_type);
  is_abstract : BOOLEAN;
  supertype_of : OPTIONAL supertype_expression;
  subtype_of : OPTIONAL SET [1 : ?] OF entity_type;
  explicit_attributes : SET [0 : ?] OF explicit_attribute;
  derived_attributes : OPTIONAL SET [1 : ?] OF derived_attribute;
  inverse_attributes : OPTIONAL SET [1 : ?] OF inverse_attribute;
  unique_rules : OPTIONAL SET [1 : ?] OF unique_rule;

  higher_level_constraints : OPTIONAL conjunction;
END_ENTITY;

(*******************************************************************)
(* ENTITY defined_type                                             *)
(*******************************************************************)
(* Represents a defined type, which can be used anywhere a named type *)
(* is required. The attributes capture the information from the    *)
(* corresponding TYPE definition.                                  *)
(*******************************************************************)

ENTITY defined_type SUBTYPE OF (named_type);
  tipe : underlying_type; -- "type" is a reserved word
END_ENTITY;

(******************************************************************)
(* TYPE underlying_type                                           *)
(******************************************************************)
(* Represents the possible underlying types for a type definition: *)
(* simple types such as INTEGER, aggregations such as SET, entities, *)
(* defined types, plus selects and enumerations.                  *)
(******************************************************************)

TYPE underlying_type = SELECT (defined_type,
                               simple_type,aggregation_type,
                               select_type,enumeration_type);
END_TYPE;
```

```
(******************************************************************)
(* TYPE parameter_type                                            *)
(******************************************************************)
(* Represents the possible types of an attribute: simple types such *)
(* as INTEGER, aggregation types such as SET, named types (entities *)
(* and defined types) but not direct selects or enumerations.      *)
(******************************************************************)

TYPE parameter_type = SELECT (simple_type,aggregation_type,named_type);
END_TYPE;

(******************************************************************)
(* TYPE simple_type                                               *)
(******************************************************************)
(* A simple type is either BOOLEAN, LOGICAL, NUMBER, INTEGER, REAL, *)
(* STRING or BINARY.                                              *)
(******************************************************************)

TYPE simple_type = ENUMERATION OF
   (boolean_type, logical_type,
  number_type, integer_type, real_type,
  string_type, binary_type);
END_TYPE;

(********************************************************************)
(* ENTITY aggregation_type                                          *)
(********************************************************************)
(* Represents an aggregation type: which kind of aggregate it is, the *)
(* type of its elements and its lower and upper bounds.             *)
(********************************************************************)

ENTITY aggregation_type SUBTYPE OF (frame);
  kind : aggregate_kind;
  element_type : parameter_type;
  lower_bound : expression;
  upper_bound : expression;
  is_unique : BOOLEAN;
  is_optional : BOOLEAN;
END_ENTITY;

(**************************************************)
(* TYPE aggregate_kind                            *)
(**************************************************)
(* An aggregate is either a SET, BAG, LIST or ARRAY. *)
(**************************************************)

TYPE aggregate_kind = ENUMERATION OF (set_type,bag_type,list_type,
                                      array_type);
END_TYPE;
```

```
(****************************************************************)
(* TYPE select_type                                             *)
(****************************************************************)
(* A select is a set of named types (entities or defined types). *)
(****************************************************************)

TYPE select_type = SET [1 : ?] OF named_type;
END_TYPE;

(********************************************************************)
(* TYPE enumeration_type                                            *)
(********************************************************************)
(* An enumeration is an ordered set of symbols (enumeration types do *)
(* have an order relation).                                         *)
(********************************************************************)

TYPE enumeration_type = LIST [1 : ?] OF UNIQUE symbol;
END_TYPE;

(*************************************)
(* TYPE symbol                       *)
(*************************************)
(* An enumeration symbol is a string. *)
(*************************************)

TYPE symbol = STRING;
END_TYPE;

(*******************************************************************)
(* TYPE supertype_expression                                       *)
(*******************************************************************)
(* A supertype expression is either just an entity reference, or an *)
(* intermediate ONEOF, AND or ANDOR operation.                     *)
(*******************************************************************)

TYPE supertype_expression = SELECT (entity_type, oneof_expression,
                                    and_expression, andor_expression);
END_TYPE;

(******************************************************)
(* ENTITY oneof_expression                            *)
(******************************************************)
(* A ONEOF expression is a set of supertype expressions. *)
(******************************************************)

ENTITY oneof_expression;
  expressions : SET [1 : ?] OF supertype_expression;
END_ENTITY;
```

```
(*****************************************************)
(* ENTITY and_expression                             *)
(*****************************************************)
(* An AND expression is a pair of supertype expressions. *)
(*****************************************************)

ENTITY and_expression;
  first_expression : supertype_expression;
  second_expression : supertype_expression;
END_ENTITY;

(*******************************************************)
(* ENTITY andor_expression                             *)
(*******************************************************)
(* An ANDOR expression is a pair of supertype expressions. *)
(*******************************************************)

ENTITY andor_expression;
  first_expression : supertype_expression;
  second_expression : supertype_expression;
END_ENTITY;

(********************************************************************)
(* ENTITY attribute                                                 *)
(********************************************************************)
(* Represents an attribute of an entity, which can either be        *)
(* explicit, INVERSE or DERIVEd. All attributes have a name and a   *)
(* type; "owner" refers to the supertype of the entity this attribute *)
(* belongs to, just in case it overrides it. `attribute' is a subtype *)
(* of `expression' so that anywhere an expression is expected, it can *)
(* be a reference to an attribute.                                  *)
(********************************************************************)

ENTITY attribute ABSTRACT SUPERTYPE OF (ONEOF (explicit_attribute,
                                               derived_attribute,
                                               inverse_attribute))
                 SUBTYPE OF (expression);
  name : STRING;
  tipe : parameter_type;        -- "type" is a reserved word
  owner : OPTIONAL entity_type;
END_ENTITY;

(*************************************************************)
(* ENTITY explicit_attribute                                 *)
(*************************************************************)
(* Represents an explicit attribute in an entity, which may be *)
(* OPTIONAL.                                                 *)
(*************************************************************)

ENTITY explicit_attribute SUBTYPE OF (attribute);
  is_optional : BOOLEAN;
```

```
INVERSE
  the_entity : entity_type FOR explicit_attributes;
END_ENTITY;

(*********************************************************************)
(* ENTITY inverse_attribute                                          *)
(*********************************************************************)
(* Represents an INVERSE attribute in an entity, the value of which *)
(* is an entity that has this entity as the value of its FOR        *)
(* attribute.                                                        *)
(*********************************************************************)

ENTITY inverse_attribute SUBTYPE OF (attribute);
  for_attribute : attribute;
INVERSE
  the_entity : entity_type FOR inverse_attributes;
END_ENTITY;

(***********************************************************************)
(* ENTITY derived_attribute                                            *)
(***********************************************************************)
(* Represents a DERIVEd attribute in an entity, the value of which is *)
(* calculated using a formula rather than stored explicitly.          *)
(***********************************************************************)

ENTITY derived_attribute SUBTYPE OF (attribute);
  formula : expression;
INVERSE
  the_entity : entity_type FOR derived_attributes;
END_ENTITY;

(***********************************************************************)
(* ENTITY local_rule                                                   *)
(***********************************************************************)
(* Represents constraints on an entity, which are either UNIQUE rules *)
(* or domain (WHERE) rules.                                            *)
(***********************************************************************)

ENTITY local_rule ABSTRACT SUPERTYPE OF (ONEOF (unique_rule,
                                                domain_rule))
                  SUBTYPE OF (frame);
  label : STRING;
END_ENTITY;

(**********************************************)
(* ENTITY unique_rule                         *)
(**********************************************)
(* Represents a uniqueness rule on an entity. *)
(**********************************************)

ENTITY unique_rule SUBTYPE OF (local_rule);
```

```
  attributes : SET [1 : ?] OF attribute;
INVERSE
  the_entity : entity_type FOR unique_rules;
END_ENTITY;

(*****************************************)
(* ENTITY domain_rule                    *)
(*****************************************)
(* Represents a domain rule on an entity. *)
(*****************************************)

ENTITY domain_rule SUBTYPE OF (local_rule);
  expression : expression;
INVERSE
  the_type : named_type FOR domain_rules;
END_ENTITY;

(*********************************)
(* ENTITY expression             *)
(*********************************)
(* Superclass of all expressions. *)
(*********************************)

ENTITY expression ABSTRACT SUPERTYPE OF
                  (ONEOF (attribute,
                          literal_expression,
                          built_in_constant,
                          unary_operation,
                          binary_operation,
                          query_expression,
                          query_variable,
                          function_call,
                          qualifier,
                          unresolved_reference,
                          aggregate_initialiser))
                  SUBTYPE OF (frame);
END_ENTITY;

(******************************)
(* ENTITY literal_expression   *)
(******************************)
(* Represents a literal value. *)
(******************************)

ENTITY literal_expression SUBTYPE OF (expression);
  literal_value : literal;
END_ENTITY;
```

```
(********************************************************************)
(* TYPE literal                                                     *)
(********************************************************************)
(* A literal can be either an integer, a real number, a logical,    *)
(* a string or a binary.                                            *)
(********************************************************************)

TYPE literal = SELECT (integer_literal, real_literal, logical_literal,
                       string_literal, binary_literal);
END_TYPE;

(***********************)
(* TYPE integer_literal *)
(***********************)
(* An integer literal.  *)
(***********************)

TYPE integer_literal = INTEGER;
END_TYPE;

(*************************)
(* TYPE real_literal      *)
(*************************)
(* A real number literal. *)
(*************************)

TYPE real_literal = REAL;
END_TYPE;

(*********************************************)
(* TYPE logical_literal                       *)
(*********************************************)
(* A logical literal (TRUE, FALSE or UNKNOWN). *)
(*********************************************)

TYPE logical_literal = LOGICAL;
END_TYPE;

(**********************)
(* TYPE string_literal *)
(**********************)
(* A string literal.   *)
(**********************)

TYPE string_literal = STRING;
END_TYPE;
```

```
(***********************)
(* TYPE binary_literal *)
(***********************)
(* A binary literal.   *)
(***********************)

TYPE binary_literal = BINARY;
END_TYPE;

(**************************************************************)
(* ENTITY built_in_constant                                   *)
(**************************************************************)
(* Represents an occurrence of one of the built-in constants. *)
(**************************************************************)

ENTITY built_in_constant SUBTYPE OF (expression);
  built_in_value : built_in;
END_ENTITY;

(***************************************************)
(* TYPE built_in                                   *)
(***************************************************)
(* The built in constants are e, PI, SELF and `?'. *)
(***************************************************)

TYPE built_in = ENUMERATION OF (e, pie, this, infinity);
END_TYPE;

(*********************************************)
(* ENTITY unary_operation                    *)
(*********************************************)
(* Represents an operation with one operand. *)
(*********************************************)

ENTITY unary_operation SUBTYPE OF (expression);
  operator : unary_operator;
  expression : expression;
END_ENTITY;

(**************************************)
(* TYPE unary_operator                *)
(**************************************)
(* The unary operators are NOT and -. *)
(**************************************)

TYPE unary_operator = ENUMERATION OF (not_operator, unary_minus);
END_TYPE;
```

```
(********************************************)
(* ENTITY binary_operation                  *)
(********************************************)
(* Represents an operation with two operands. *)
(********************************************)

ENTITY binary_operation SUBTYPE OF (expression);
  operator : binary_operator;
  first_expression : expression;
  second_expression : expression;
END_ENTITY;

(*******************************************************************)
(* TYPE binary_operator                                              *)
(*******************************************************************)
(* The binary operators are <, <=, >, >=, =, <>, :=:, :<>:, IN, LIKE, *)
(* +, -, OR, XOR, *, /, DIV, MOD, AND, complex constructor and **.    *)
(*******************************************************************)

TYPE binary_operator = ENUMERATION OF (
 less_than, less_than_or_equals,
 greater_than, greater_than_or_equals,
 equals, not_equals, instance_equals, instance_not_equals,
 in_operator, like_operator,
 add, subtract, or_operator, xor_operator,
 multiply, divide, div_operator, mod_operator, and_operator,
 complex_constructor, exponent);
END_TYPE;

(**************************)
(* ENTITY query_expression *)
(**************************)
(* Represents a query.     *)
(**************************)

ENTITY query_expression SUBTYPE OF (expression);
  variable : query_variable;
  aggregation_expression : expression;
  logical_expression : expression;
END_ENTITY;

(*******************************************)
(* ENTITY query_variable                   *)
(*******************************************)
(* Represents the bound variable in a query. *)
(*******************************************)

ENTITY query_variable SUBTYPE OF (expression);
  name : STRING;
INVERSE
  the_query : query_expression FOR variable;
```

```
END_ENTITY;

(**********************************)
(* ENTITY function_call           *)
(**********************************)
(* Represents a call to a function. *)
(**********************************)

ENTITY function_call ABSTRACT SUPERTYPE OF (ONEOF
                     (built_in_function_call, user_function_call))
                     SUBTYPE OF (expression);
  actual_parameters : LIST [0 : ?] OF expression;
END_ENTITY;

(*********************************************)
(* ENTITY user_function_call                 *)
(*********************************************)
(* Represents a call to a user-defined function. *)
(*********************************************)

ENTITY user_function_call SUBTYPE OF (function_call);
  function_called : STRING;
END_ENTITY;

(**************************************************)
(* ENTITY built_in_function_call                  *)
(**************************************************)
(* Represents a call to one of the built-in functions. *)
(**************************************************)

ENTITY built_in_function_call SUBTYPE OF (function_call);
  function_called : built_in_function;
END_ENTITY;

(*****************************************************************)
(* TYPE built_in_function                                        *)
(*****************************************************************)
(* The built-in functions are ABS, ACOS, ASIN, ATAN, BLENGTH, COS,  *)
(* EXISTS, EXP, FORMAT, HIBOUND, HIINDEX, LENGTH, LOBOUND, LOG,     *)
(* LOG10, LOG2, LOINDEX, NVL, ODD, ROLESOF, SIN, SIZEOF, SQRT, TAN, *)
(* TYPEOF, USEDIN, VALUE, VALUE_IN and VALUE_UNIQUE.                *)
(*****************************************************************)

TYPE built_in_function = ENUMERATION OF (
 abs_function, acos_function, asin_function, atan_function,
 blength_function, cos_function, exists_function, exp_function,
 format_function, hibound_function, hiindex_function,
 length_function, lobound_function, log_function, log10_function,
 log2_function, loindex_function, nvl_function, odd_function,
 rolesof_function, sin_function, sizeof_function, sqrt_function,
 tan_function, typeof_function, usedin_function, value_function,
```

```
 value_in_function, value_unique_function);
END_TYPE;

(*************************************)
(* ENTITY qualifier                 *)
(*************************************)
(* Represents a qualified attribute. *)
(*************************************)

ENTITY qualifier SUBTYPE OF (expression);
  kind : qualifier_kind;
  first_expression : OPTIONAL expression;
  second_expression : OPTIONAL expression;
  third_expression : OPTIONAL expression;
END_ENTITY;

(******************************************)
(* TYPE qualifier_kind                    *)
(******************************************)
(* The qualifiers are `.', `\' and `[...]'. *)
(******************************************)

TYPE qualifier_kind = ENUMERATION OF (dot, backslash, subscript);
END_TYPE;

(*******************************************************************)
(* ENTITY unresolved_reference                                    *)
(*******************************************************************)
(* Yes, I'm afraid the current frame system cannot resolve all       *)
(* references (which are represented as strings in ExJava rather than *)
(* hard references), particularly those in qualifiers. It's better to *)
(* represent this officially in this model! :-)                    *)
(*******************************************************************)

ENTITY unresolved_reference SUBTYPE OF (expression);
  id : STRING;
END_ENTITY;

(**************************************)
(* ENTITY aggregate_initialiser      *)
(**************************************)
(* Represents an aggregation literal. *)
(**************************************)

ENTITY aggregate_initialiser SUBTYPE OF (expression);
  elements : LIST [0 : ?] OF aggregate_initialiser_element;
END_ENTITY;
```

```
  (**************************************************)
  (* ENTITY aggregate_initialiser_element           *)
  (**************************************************)
  (* Represents an element in an aggregation literal. *)
  (**************************************************)

  ENTITY aggregate_initialiser_element;
    expression : expression;
    repeat_count : OPTIONAL expression;
  INVERSE
    the_aggregate_initialiser : aggregate_initialiser FOR elements;
  END_ENTITY;
END_SCHEMA;
```

A.2 Frame Class Model of Higher-Level Constraints

```
(**************************************************)
(** SCHEMA hlc_frame_class_model                 **)
(**************************************************)
(** Frame class model of higher-level constraints. **)
(**************************************************)

SCHEMA hlc_frame_class_model;
USE FROM express_frame_class_model;

  (**************************************************)
  (* ENTITY higher_level_constraint                 *)
  (**************************************************)
  (* The superclass of all higher-level constraints. *)
  (**************************************************)

  ENTITY higher_level_constraint ABSTRACT SUPERTYPE
                                 SUBTYPE OF (frame);
    extracted_from : SET [1 : ?] OF frame;
    conjunct_of : OPTIONAL conjunction;
  END_ENTITY;

  (**************************************)
  (* ENTITY conjunction                 *)
  (**************************************)
  (* Higher-level CONJUNCTION constraint. *)
  (**************************************)

  ENTITY conjunction SUBTYPE OF (higher_level_constraint);
  INVERSE
    conjuncts : SET [0 : ?] OF higher_level_constraint
                             FOR conjunct_of;
  END_ENTITY;
```

```
(*************************************)
(* ENTITY exactly_one                *)
(*************************************)
(* Higher-level EXACTLY_ONE constraint. *)
(*************************************)

ENTITY exactly_one SUBTYPE OF (higher_level_constraint);
  choice_set : SET [1 : ?] OF choice;
  constrainee : constrainee;
END_ENTITY;

(*************************************)
(* ENTITY at_most_one                *)
(*************************************)
(* Higher-level AT_MOST_ONE constraint. *)
(*************************************)

ENTITY at_most_one SUBTYPE OF (higher_level_constraint);
  choice_set : SET [1 : ?] OF choice;
  constrainee : constrainee;
END_ENTITY;

(*****************************************************************)
(* TYPE constrainee                                              *)
(*****************************************************************)
(* The constrainee of an HLC must be either an entity or attribute. *)
(*****************************************************************)

TYPE constrainee = SELECT (entity_type, attribute);
END_TYPE;

(******************************************************************)
(* ENTITY bound                                                   *)
(******************************************************************)
(* Higher-level BOUND constraint on a variable expression which must *)
(* be <, >, <= or >= some limit.                                  *)
(******************************************************************)

ENTITY bound SUBTYPE OF (higher_level_constraint);
  kind : bound_kind;
  inclusive : BOOLEAN;
  discrete : BOOLEAN;
  variable : expression;
  limit : NUMBER;
END_ENTITY;
```

```
(*****************************************)
(* TYPE bound_kind                       *)
(*****************************************)
(* A bound can either be lower or upper. *)
(*****************************************)

TYPE bound_kind = ENUMERATION OF (lower, upper);
END_TYPE;

(**********************************************************************)
(* TYPE choice                                                        *)
(**********************************************************************)
(* A choice for an exactly_one HLC can be either a frame or a symbol. *)
(**********************************************************************)

TYPE choice = SELECT (frame, symbol);
END_TYPE;

(******************************************)
(* ENTITY range                           *)
(******************************************)
(* Higher-level numeric RANGE constraint. *)
(******************************************)

ENTITY range SUBTYPE OF (higher_level_constraint);
  lower_bound : bound;
  upper_bound : bound;
  variable : expression;
END_ENTITY;

(*******************************************************************)
(* ENTITY non_null                                                 *)
(*******************************************************************)
(* Higher-level constraint that the value of an explicit attribute *)
(* must not be  null.                                              *)
(*******************************************************************)

ENTITY non_null SUBTYPE OF (higher_level_constraint);
  attribute : explicit_attribute;
END_ENTITY;

(**************************************************************)
(* ENTITY zero_or_more                                        *)
(**************************************************************)
(* Higher-level constraint that captures the common semantics *)
(* of OPTIONAL SET [1 : ?] and SET [0 : ?].                   *)
(**************************************************************)

ENTITY zero_or_more SUBTYPE OF (higher_level_constraint);
  attribute : explicit_attribute;
END_ENTITY;
```

```
 (****************************************)
 (* ENTITY equation                      *)
 (****************************************)
 (* Higher-level constraint for equations. *)
 (****************************************)

 ENTITY equation SUBTYPE OF (higher_level_constraint);
   equals : SET [2 : ?] OF expression;
 END_ENTITY;

 (********************************)
 (* ENTITY adder                 *)
 (********************************)
 (* Higher-level adder constraint. *)
 (********************************)

 ENTITY adder SUBTYPE OF (higher_level_constraint);
   addends : BAG [2 : ?] OF expression;
   sum : expression;
 END_ENTITY;

 (*************************************)
 (* ENTITY multiplier                 *)
 (*************************************)
 (* Higher-level multiplier constraint. *)
 (*************************************)

 ENTITY multiplier SUBTYPE OF (higher_level_constraint);
   multiplicands : BAG [2 : ?] OF expression;
   product : expression;
 END_ENTITY;
END_SCHEMA;
```

A.3 Frame Class Model of Cases

```
(*********************************************)
(** SCHEMA cbr_frame_class_model            **)
(*********************************************)
(** Frame class model of case-based reasoning. **)
(*********************************************)

SCHEMA cbr_frame_class_model;
USE FROM express_frame_class_model;
USE FROM hlc_frame_class_model;
```

```
 (*****************************************************************)
 (* ENTITY case_frame                                             *)
 (*****************************************************************)
 (* Represents a case, which has a problem and a solution (a higher- *)
 (* level constraint to extract from an EXPRESS model).           *)
 (*****************************************************************)

 ENTITY case_frame SUBTYPE OF (frame);
   problem : frame;
   solution : higher_level_constraint;
 END_ENTITY;
END_SCHEMA;
```