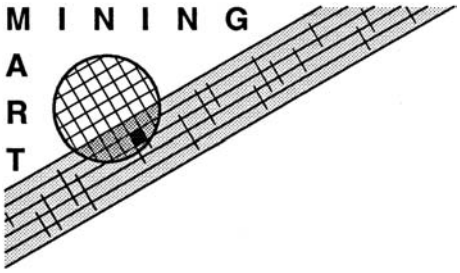

Enabling End-User Datawarehouse Mining
Contract No. IST-1999-11993
No. TR 18-01

Representing Constraints, Conditions and Assertions in M4

Martin Scholz

Dortmund, April 25, 2002

1 Where M4 should be changed

In order to directly attach constraints, conditions and assertions to the operators in the M4 model, some modifications are necessary. First of all the question arises, how operators are represented in the model. Up to now, parameters are attached to the class of operators, implying that operators in the model refer to operator applications, rather than to operators as such. In the relational representation, for each step an operator is embedded in, another instance of the operator appears in the table OPERATOR_T.

It seems more intuitive to link the arguments, including inputs, outputs and parameters[1], to the steps, rather than to single operators. Thus the first change proposed here, is to change the semantics of the class OPERATOR.

Each operator should appear exactly once in this class and thus in the table OPERATOR_T. For each step that the operator is embedded in, a foreign key reference in STEP_T is sufficient to specify the particular operator. Further foreign key references, from PARAMETER_T to STEP_T should determine the arguments and output of the operator application. No changes are necessary here, because references to STEP_T and OPERATOR_T are already foreseen in M4. As a consequence of the new structure the table OP_NAME_T will be redundant and can be removed from the M4 model, as soon as the compiler is adjusted to this change. The attribute PAR_OPID of the table PARAMETER_T will be replaced by PAR_STEPID, a foreign key reference to the step the parameters belong to. In this setting the constraints, conditions and assertions can be defined referencing the operators specified in the table OPERATOR_T. The table OPCONSTRAINT_T will be deleted. Instead four new tables will be added.

Figure one depicts the changes to the model. The constraints are represented in two tables, OP_PARAMS_T to define the inputs and outputs and OP_CONSTR_T for other constraints. Conditions and constraints will be described by two other tables. This results in the following list of new tables:

- OP_PARAMS_T defining the admissible inputs and outputs
- OP_CONSTR_T for constraints
- OP_COND_T for conditions
- OP_ASSERT_T for assertions.

The following sections will describe these tables in detail. An example scenario for using the formalism is given in section 6. Please note, that this change of the representation currently neither considers subchains nor recommendations to the case designer.

[1] Misleadingly the sum of these arguments is called *parameters* in the model so far.

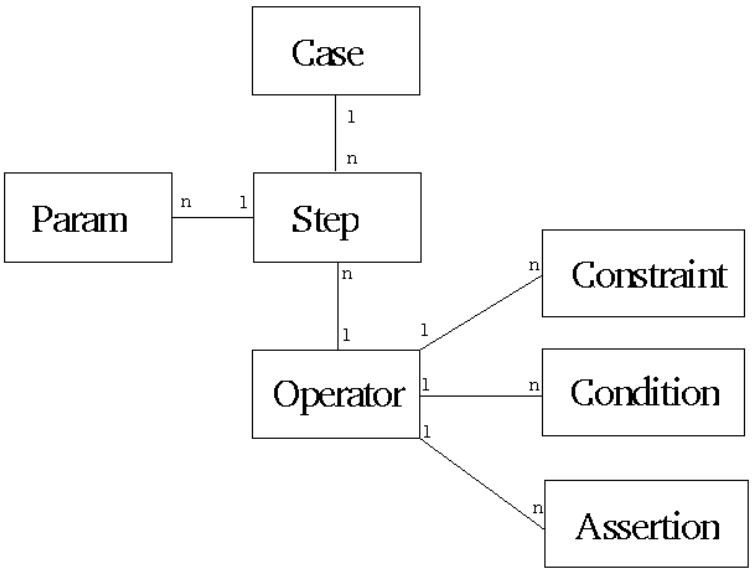

Figure 1: Modifications

2 Specifying inputs and outputs

The first of the tables is added to be able to state restrictions on the arguments used by operators. In detail:

- type and number of input arguments
- type of output

The new table OP_PARAMS_T is the first of two tables implementing the concept "Constraint" in figure 1. It has the following attributes:

- PARAM_ID: ID of the parameter tuple
- OP_ID: foreign key reference to OPERATOR_T
- MINARG: minimum number of parameters of this kind - might be "0"
- MAXARG: maximum number of parameters of this kind - NULL, if unrestricted
- NAME: The name given here has to be used in table PARAMETER_T[2] in order to be able to identify parameters. The name is interpreted as a prefix, to allow more than one parameter. The ordering over the set of parameters, matching the prefix, is determined by their number given

[2]Concept PARAM in figure 1.

within PARAMETER_T. The identification is necessary in order to formulate applicability conditions, constraints and assertions.

- IO: Is this parameter(set) an input or the output.
- TYPE: The M4 model uses different kind of object types handled by operators. The following list shows the alternatives:
  - CON (CONCEPT)
  - BA (BASE_ATTRIBUTE)
  - REL (RELATION)
  - V (VALUE)
  - MCF (MULTI COLUMN FEATURE)
  - FUNC (FUNCTION)

  The following types appear in deliverable D8/9, but seem to be deprecated, or at least not handled by software of any kind:
  - Q (QUERY)
  - TI (TIME_INTERVAL, in D8/9: just output)
  - DR (DEVIATION_RULES, in D8/9: just output)

  Unless any software module makes use of these types, they will not be supported. Time intervals can be represented by two time features and are a typical kind of multi column features, anyway. Queries do not have to be handled as an extra data type in M4 and deviation rules can be stored as database functions, similar to decision trees and SVM models.
- DOCU: Basically for the HCI, this attribute holds a brief free text description of the according parameter.

The intended usage of the above specifications of inputs and outputs is to list the admissible arguments, one after another, declaring types and names. To support arrays[3] as well, the minimum and maximum number of objects is necessary. In this case the name is interpreted as a prefix. The names of the arguments are important to identify single arguments on the instance level, namely in the table PARAMETER_T. Having these references enables a formulation of constraints, conditions and assertions.

As an example let's give the declaration of the first argument of operator no. 35, which has to be a (single) value labeled "HEAD_W":

PARAM_ID	OP_ID	MINARG	MAXARG	NAME	IO	TYPE
1001	35	1	1	"HEAD_W"	"IN"	"V"

Let us then specify an array of inputs. For arrays the following convention is

[3]By the same mechanism optional arguments could be handled, as well. However, there was no necessity, yet.

used: Given NAME="WEIGHT" all inputs with the appropriate type and name prefix "WEIGHT" will be considered part of this input array. The ordering of the array is derived from the order of the arguments in the table PARAMETER_T.

To specify a list of weights, the following statement can be used:

PARAM_ID	OP_ID	MINARG	MAXARG	NAME	IO	TYPE
1002	35	1	NULL	"WEIGHT"	"IN"	"V"

This would aggregate input arguments like "WEIGHT01" or "WEIGHTING" with data type "V" (value) for this operator. The same convention is applicable, if the output is an array.

3 Representing Operator Constraints

To formulate other operator constraint than expressable by the mechanism presented in section 2 the table OP_CONSTR_T is added to the M4 schema. It contains the attributes CONSTR_ID, CONSTR_OPID, CONSTR_TYPE, CONSTR_OBJ1, CONSTR_OBJ2, and CONSTR_SQL.
The semantics of these attributes is as follows:

- CONSTR_ID: constraint identifier
- CONSTR_OPID: foreign key reference to operator
- CONSTR_TYPE: different kinds of contraints can be used. Depending on the type, CONSTR_OBJ1 and CONSTR_OBJ2 may have different semantics
  - ISA: the concept or relation (CONSTR_)OBJ1 *isA* concept or relation (CONSTR_)OBJ2
  - SAME_FEAT: The concepts OBJ1 and OBJ2 have the same set of features, where "same" only means that feature names are equal. The assumption is made, that two different BaseAttributes may have the same name, e.g. "CUSTOMER", one time with, one time without missing values, as long as they do not share the same concept.
  - COPY_REPL: This constraint is especially meant for operators changing only one feature, leaving all the other features untouched. In this case input and output concept share the same set of features, except for one attribute, specified in OBJ2, while OBJ1 denotes the input concept. Here the assumption is made, that BaseAttributes may be part of several concepts. This implies another change of the M^4 model.
  - COMP:
    If OBJ1 is a CONCEPT, then OBJ2 is a relation. If OBJ1 is a relation, then OBJ2 can be either a concept or a relation. The constraint

states, that the relations or the concept and the relation are of compatible type.

  * If a concept C is followed by a relation R, then $C\ isA\ domain(R)$.
  * If a relation R_1 is followed by another relation R_2, then $range(R_1)\ isA\ domain(R_2)$.
  * If a relation R is followed by a concept C, then $range(R)\ isA\ C$.
    Instead of a single relation, R might also denote a set of relations, using their name prefix. The chaining of these relations is done by their ordering, using the above rules. Finally a BaseAttribute may be used, wherever a Concept is expected. In this case the BaseAttribute must be present in the according concept.

– LINK:
  This kind of constraint makes sure, that several features are deterministically and efficiently reachable ("joinable") on the level of relational tables.
  If OBJ1 and OBJ2 are BaseAttributes, then they have to refer to attributes of the same relational table, or there has to be a foreign key reference from the table containing OBJ1 to that containing OBJ2.
  If OBJ1 and OBJ2 are concepts, then all of their features have to be connected. Features are connected by sharing a relational table, by beeing in tables connected by a foreign key reference or by chaining these two alternatives. In case of chaining, references are directed and all features in OBJ2 need to be reachable from OBJ1 (not necessarily vice versa).
  If one object is a feature and one is a concept, then the ideas above will be canonically adopted, treating a feature like a single feature concept.
  Instead of single concepts, prefixes might be used, to have concept arrays. The constraint for two concepts has to hold for each concept and its successor.

– IN:
  OBJ1 holds the name of one or the prefix for more features.
  OBJ2 is the name of a concept, containing all these features. If OBJ2 denotes the (an) output concept, then this concept needs to contain a feature (or all, if a feature set is given) with the original name(s) of the object(s) referenced by OBJ1.

– TYPE:
  OBJ1 holds the name of one or the prefix for more features.
  OBJ2 is the name of a conceptual data type, allowed for the according attribute(s). If more data types are supported (not correctly subsumable by one of the M4 categories) multiple tuples for a single feature are possible.

– LT, GT, LE, GE, NE:
  OBJ1 is the name of one (or more $\rightarrow$ name prefix) input parameters.

OBJ2 is usually a number, in case of "NE" it might be a nominal constant. The conditions are "lower than", "greater than", "lower or equal", "greater or equal" and "not equal".

- ONE_OF:
  OBJ1 is the name of one (or more → name prefix) input parameters. OBJ2 is a comma separated string of possible values for this (these) parameter(s). An example for parameter "kernel type" could be: "linear,polynomial,radial". If necessary, a comma within a possible parameter value can be expressed by writing "\,".
- SUM:
  OBJ1 references a set of numerical parameters by their name prefix. OBJ2 gives the value the sum of these parameters needs to have.

- CONSTR_DOCU: This attribute holds a brief free text description of the constraint. It is intended to be used by the HCI.
- CONSTR_SQL: If possible an SQL-query implementing the test of the constraint is stored here, NULL otherwise.

The following examples demonstrate the usage[4]:

- Constraint 1: For operator no. 35 the output concept C_OUT has to be of the same type or a specialization of the input concept C_IN.

ID	OPID	TYPE	OBJ1	OBJ2
1003	35	"ISA"	"C_OUT"	"C_IN"

- Constraint 2: A given array of relationships, passed to operator no. 35, defines a domain compatible chain. The chain shall start with input concept C. The relationships should have the name prefix "REL" (e.g. $REL01, \ldots, REL20$) and have to be given in the relevant order. The concept that corresponds to the domain of the last relation should contain the BaseAttribute F.

ID	OPID	TYPE	OBJ1	OBJ2
1004	35	"COMP"	"C"	"REL"
1005	35	"COMP"	"REL"	"F"

- Constraint 3: A join on a given array of concepts, prefix C, has to be efficiently computable. Thus first of all the concepts in the array need to be stored in a single table, or to be easily joinable by exploiting foreign key references. The same holds for successive concepts. They need to be reachable via foreign key references, or might even be stored in the same table as the predecessor concept. Finally the BaseAttribute $ATTR$ should be connected equivalently easy from the last concept of the array.

ID	OPID	TYPE	OBJ1	OBJ2
1006	35	"LINK"	"C"	"ATTR"

[4]Omitting the suffix "CONSTR_" in attribute names.

- Constraint 4: All input attributes mentioned need to be part of the input concept C. This can be stated by using "IN"-constraints. E.g. for attributes Att_1, ..., Att_10 and B:

ID	OPID	TYPE	OBJ1	OBJ2
1007	35	"IN"	"Att_"	"C"
1008	35	"IN"	"B"	"C"

- Constraint 5: A BaseAttribute D has to be of type TIME or CATEGORIAL:

ID	OPID	TYPE	OBJ1	OBJ2
1009	35	"TYPE"	"D"	"TIME"
1010	35	"TYPE"	"D"	"CATEGORIAL"

- Constraint 6: The definition of a valid interval $[0, 1]$ for a numerical parameter $PROBABILITY$ of operator no. 35 can be expressed by the follwing two tuples:

ID	OPID	TYPE	OBJ1	OBJ2
1011	35	"GE"	"PROBABILITY"	0
1012	35	"LE"	"PROBABILITY"	1

- Constraint 7: The sum of an array of weights, name prefix $WEIGHT$, has to be 1:

ID	OPID	TYPE	OBJ1	OBJ2
1013	35	"SUM"	"WEIGHT"	1

4 Representing Conditions for Operator Applications

Conditions are represented using the table OP_COND_T, which contains the attributes COND_ID, COND_OPID, COND_TYPE, COND_OBJ1, COND_OBJ2 and COND_SQL:

- COND_ID: condition identifier
- COND_OPID: foreign key reference to OPERATOR_T
- COND_TYPE: different kinds of conditions can be formulated. Depending on the type, COND_OBJ1 and COND_OBJ2 may have different semantics. For conditions taking one argument only, COND_OBJ2 needs to be NULL.

  The list below describes the set of applicable conditions. If nothing else is stated, a BaseAttribute (or multiple BaseAttributes, using name prefixes) is given by (COND_)OBJ1, while (COND_)OBJ2 has to be NULL.

  - HAS_NULLS: At least one NULL value is present in the BaseAttribute.

- NOT_NULL: No values are missing. If OBJ1 specifies a concept, then in each of the concept's features no NULL entries are allowed.
- UNIQUE: The given BaseAttribute contains no duplicates.
- ORDERED: The concept values are ordered by the given feature, while OBJ2 determines if ascending ("INC"), descending ("DEC"), or if it does not matter (NULL).
- EQUIDIST: This condition is especially interesting for time series. It is checked, if the BaseAttribute (specified by OBJ1) is ordered and equidistant, which is the normal form for time series. OBJ2 might be NULL or specify a stepsize.
- LOWER_BOUND: The given BaseAttribute may not contain values below OBJ2. This condition applies to attributes of type TIME, as well.
- UPPER_BOUND: analogous for upper bounds
- AVG: The given feature has to have the average value specified by OBJ2.
- STD_DEV: The feature has a standard deviation of OBJ2.
- LE, LT, GE, GT: As for constraints, "LE" stands for "lower or equal", "GT" means "greater than" and so on. In case of such a condition it has to be checked, if the inequality between two BaseAttributes, given as arguments OBJ1 and OBJ2, holds for all instances of the according concept[5]. If the BaseAttributes belong to different concepts, the condition is not met.
  OBJ2 might also be a numerical constant, rather than a BaseAttribute.
  Further on, OBJ1 and/or OBJ2 can also refer to multiple BaseAttributes by giving their name prefixes, stating that the condition is met between all objects o_1 and o_2, with $o_1 \in OBJ1$ and $o_2 \in OBJ2$.

- COND_DOCU: This attribute holds a brief free text description of the condition.

- COND_SQL: If possible an SQL-query implementing the test of the condition is stored here, NULL otherwise.

Examples on usage[6]:

[5] Please note, that in the original version each BaseAttribute belongs to exactly one concept, so there is no need to provide the concept as an additional argument. In a later version there might be a n..m relation between Concepts and BaseAttributes. Even in this case it does not matter, which Concept containing both BaseAttributes will be chosen to check the condition. For all these concepts the same columns will be referenced, and the pairing of the columns values within each tuple will be the same.

[6] Omitting the suffix "COND_" in attribute names.

- Condition 1: A concept TC constituting a time series that is ascendingly sorted over the time feature TA[7]:

ID	OPID	TYPE	OBJ1	OBJ2
1014	35	"ORDERED"	"TA"	"INC"

- Condition 2: A numerical BaseAttribute labeled $Attr$ has to be strictly positive (e.g. for LogScaling):

ID	OPID	TYPE	OBJ1	OBJ2
1015	35	"GT"	"Attr"	0

- Condition 3: Two attributes of type TIME belong to the same concept and specify time intervals. Thus for all instances the starting time ($START$) must not be later than the end of the interval (END):

ID	OPID	TYPE	OBJ1	OBJ2
1016	35	"LE"	"START"	"END"

5 Representation of Assertions

Assertions are at the same level as conditions, giving guarantees about charateristics of the output. If an assertion is related to a feature of the output, while the output is a concept, then the following kind of reference is ment: The output needs to contain a feature with the same original name than the specified feature. The statement is about this feature.

Assertions are stated using the table OP_ASSERT_T containing the attributes ASSERT_ID, ASSERT_OPID, ASSERT_TYPE, ASSERT_OBJ1, ASSERT_OBJ2, ASSERT_SQL:

- ASSERT_ID: assertion identifier
- ASSERT_OPID: reference to OPERATOR_T
- ASSERT_TYPE: different kinds of conditions can be formulated. Depending on the type, ASSERT_OBJ1 and ASSERT_OBJ2 may have different semantics. For assertions taking one argument only, ASSERT_OBJ2 needs to be NULL.
  - SUBSET: (ASSERT_)OBJ1 is an input concept of which the output concept is a subset. This statement makes sense only, if the input and the output concept are of same type (same features).
  - PROJ: OBJ1 is an input concept of which the output concept is a projection (Complete projection of all tuples!).

[7] That TA has to be a feature of concept TC is a constraint, that should be stated in the according table, using "IN".

- NOT_NULL: OBJ1 denotes the name of the feature which is asserted not to be NULL in the output. Alternatively the name of the output concept may be entered here, stating that in all of the features no missing values are present.
- LOWER_BOUND and UPPER_BOUND: The output feature OBJ1 only contains values greater or equal / lower or equal OBJ2. This kind of assertion is applicable for TIME features, as well.
- UNIQUE, ORDERED, EQUIDIST, AVG, STD_DEV might be used as well, in the meaning described for conditions, but this is not necessary, yet.

- ASSERT_DOCU: A brief free text description of the assertion should be entered here.
- ASSERT_SQL: If possible an SQL-query implementing a test of the assertion is stored here, NULL otherwise.

Examples for Assertions[8]:

- Assertion 1: The output concept of an operator is a projection of the input concept IN_CON. This can be stated as

ID	OPID	TYPE	OBJ1	OBJ2
1015	35	"PROJ"	"IN_CON"	NULL

- Assertion 2: An attribute A which is part of the input concept IN_CON and of the output concept OUT_CON does not have missing values in the output concept. This bundles two constraints with an assertion. The constraints, A being in IN_CON and A being in OUT_CON, can be formulated using "IN", as illustrated before.

  The tuple representing the "NOT_NULL" assertion, again for example operator no. 35, looks like this:

ID	OPID	TYPE	OBJ1	OBJ2
1017	35	"NOT_NULL"	"A"	NULL

[8] Omitting the suffix "ASSERT_" in attribute names.

6 A Use-Case

This section illustrates the use of the representation framework for a specific operator. Let's specify the details for the operator FeatureSelection, having ID 43 in M^4. First of all it should be recalled, that this operator with the corresponding ID is stored in table OPERATOR_T rather than in OP_NAME_T, which was deleted from the M^4 schema. The entry in this table, omitting OP_REALIZES:

OP_ID	OP_NAME	OP_LOOP	OP_MULTI	OP_MANUAL
43	"FEATURE_SEL."	"NO"	"NO"	"YES"

This operator has the following constraints:

- There is exactly one input concept.
- A set of BaseAttributes specifies a subset of the concept's features.
- The output is a single concept.

Using the same variables as in deliverable 8/9 the input/ouput constraints (OP_PARAMS_T) would be represented as

ID	OP_ID	MINARG	MAXARG	NAME	IO	TYPE
1018	43	1	1	"TheConcept"	"IN"	"CON"
1019	43	1	NULL	"TheAttributes"	"IN"	"BA"
1020	43	1	1	"TheOutputConcept"	"OUT"	"CON"

The constraint that *TheAttributes* are all present in *TheConcept* is still missing. It is entered in table OP_CONSTR_T:

ID	OPID	TYPE	OBJ1	OBJ2
1021	43	"IN"	"TheAttributes"	"TheConcept"

To illustrate, how this is related to the corresponding instances, let's have a look at table STEP_T[9], step no. 1 of case 123, embedding this operator:

ST_ID	ST_CAID	ST_NR	ST_OPID
1022	123	1	43

The arguments for this specific step are stored in table PARAMETER_T, as shown below[10]. The attribute (PAR_)OBJID references the parameter object. For a *concept* the according concept ID of table CONCEPT_T will be given here, for relations or other types the IDs of the according M^4 tables will be referenced, instead. The type is defined in (PAR_)OBJTYPE. (PAR_)STID is a foreign key reference to the step.

[9] Loop and multistep information is omitted.

[10] The column PAR_STLOOPNR and the prefix "PAR_" of attribute names are omitted.

ID	NAME	OBJID	OBJTYPE	OPID	TYPE	NR	STID
1023	TheConcept	1029	"CON"	43	IN	1	1022
1024	TheAttributes1	1030	"BA"	43	IN	2	1022
1025	TheAttributes2	1031	"BA"	43	IN	3	1022
1026	TheAttributes3	1032	"BA"	43	IN	4	1022
1027	TheAttributes4	1033	"BA"	43	IN	5	1022
1028	TheOutputConcept	1034	"CON"	43	OUT	6	1022

The names of (PAR_)NAME have to match the specifications given in the table OP_PARAMS_T. The object IDs of the arguments (PAR_OBJID) and the internal names of these objects, e.g. the CONCEPT name, can of course not be used for the formulation of constraints, etc. Instead the arguments are referenced by the parameter names of OP_PARAMS_T. Together with the above specification in table OP_PARAMS_T the following set of arguments is found from the table PARAMETER_T:

- The_Concept points to the object with ID 1029 in table CONCEPT_T.
- *TheAttributes* is an array of four BaseAttributes, referenced by their ID:
  - TheAttributes[1] points to BA no. 1030
  - TheAttributes[2] points to BA no. 1031
  - TheAttributes[3] points to BA no. 1032
  - TheAttributes[4] points to BA no. 1033

  The ordering of these four BaseAttributes is given by their (PAR_)NR values. The elements of this array are all input arguments of type BaseAttribute, which have a name beginning with "TheAttributes".
- TheOutputConcept is the concept with ID 1034.

Now let's formulate the assertion for the FeatureSelection operator: *TheOutputConcept* is an extensionally equivalent projection of *TheConcept*. This can be formulated using an assertion of type projection ("PROJ"):

ID	OPID	TYPE	OBJ1	OBJ2
1035	43	"PROJ"	"The_Concept"	NULL

7 Embedding the knowledge

The knowledge represented by the altered part of M4, described in the previous sections, will be necessary to maintain *case consistency*. The consistency is enforced by different parts of the system. To some extent the referential integrity constraints in the database will forbid to enter problematic data into M4, for instance steps not embedding any operator.

However, due to limitations in representing constraints at this level, the main efforts for consistency checking will be elsewhere, namely in the case editor. The

constraints and conditions provided in the tables named above should be checked whenever the case designer adds a step and/or modifies a case.

Figure 1 shows the organization of the relevant parts of M^4. The concepts CASE, STEP and PARAM can be conceived as *instances*, while OPERATOR, CONSTRAINT, CONDITION and ASSERTION are comparable to *classes*[11] in the sense of object oriented modelling. Steps are instances of operators. If a case designer adds a step, then the case editor should automatically check, which input and output arguments are required and forbid to submit any combination of argument settings, not allowed due to *constraints* represented in M^4. In contrast to steps, the list of existing operators and the information about operator constraints, conditions and assertions may not be altered by the case designer, but can be considered as fixed. If a case designer wants to connect two steps, then assertions might be of relevance, as well.

Operator *conditions* can generally be checked at runtime, only, because before the relational tables behind the meta-data cannot be analyzed. In the extreme case of steps using a FeatureSelection operator, even the conceptual level cannot be specified before executing the operator, because the features of the output concept depend on the specific dataset. The compiler needs to check the conditions of an operator before executing it. This is necessary in order to generate runtime exceptions and meaningful messages to the user. Additionally to checking conditions, it would do no harm, if the compiler could check the constraints, as well, before executing an operator. However, this is redundant somehow, because it should have happened in the case editor, before. Meta-data resulting from assertions like "NOT NULL" helps to avoid unnecessary checking of data properties at runtime.

[11] Please note, that this distinction is different from conceptual and relational/executable level. One should also keep in mind that the distinction between information available early on (while designing a case) and that, available at runtime (executing a case) is still something else. All these distinctions are of importance for the given context!