Evaluation of Predicate Calculus

By Arve Meisingset, retired research scientist from Telenor Research

31.05.2017 Oslo Norway

Predicate Calculus is a calculus on the truth-values of predicates. This usage tends to split statements into fragments that do not outline the structure of the data within the statements.

In this paper, we will show example simple alphanumeric form-filling computer screens, show what specifications are needed for these, and compare Predicate Calculus with this need. We conclude with abandoning Predicate Calculus for this use and show how to replace the calculus by a structure of data.

The proposed language organizes data in a tree of lists of list in three dimensions. In this language, we neither need predicates, truth-values, logical connectives, quantifiers nor variables, and we do not use Set theory. Any item in the data tree will be identified local to its class and local to its superior item in the data tree. Hence, an entire branch of the data tree has to be provided to identify a term.

The proposed language is appropriate for descriptions, inscriptions and prescriptions. Predicate Calculus is a meta-language for reasoning, and it can describe and prescribe other predicates only.

If you do not read the whole paper, please read the Conclusion.

1 On complete statements

Predicate Calculus assumes that a simple statement like 'John loves Mary' is a complete statement. An example three-place predicate is shown in Figure 1-1.

Loves (JOHN, MARY, 2017)

Figure 1-1 Example predicate

We will discuss an alternative to this expression.

If we should create a user interface for managing the information, it might look like Figure 1-2.

Person		
Name	Loved-person	
	Name	Date
JOHN	MARY	2017
MARY		

Figure 1-2 Example simple computer screen

Here we have introduced headings as class labels. We have headings of attribute classes and of entity classes (underlined). Values are indicated with capital letters.

The reason for distinguishing entities and attributes is that the attributes are defined within the scope of their entity, eg. a Person's Name is defined for a Person only, and the Name has no meaning outside the scope of the Person-s.

- *Observation 1a: Predicate Calculus assumes that all (constant) terms are globally unique; this does not support complex naming structures both depending on the class and superior terms, as of the alternative language..* See Figure 1-3.

In Figure 1-2 we see no predicate, but we see a structure of terms. The structure of the terms in the headings may be presented as follows:

Person (Name, Loved-person (Name, Date)).

Figure 1-3 Example specification of screen

The expression in Figure 1-3 is identical to the arrangement of headings in Figure 1-2, except that line shifts are replaced by parentheses.

Figure 1-3 shows the abstract syntax of the headings; all nodes of the syntax tree appear as headings. The headings are classes. The parentheses show the containment structure. Each term is defined within the scope of its superior term. The terms are classes.

- *Observation 1b: When defining data, there is no need for predicates, as we do not map to truth-values.*

The structure of the instances (with values) within the population in Figure 1-2 may be presented as follows:

Person (Name (JOHN), Loved-person (Name (MARY), Date (2017))), Person (Name (MARY)).

Figure 1-4 Example contents of the variable fields

All these terms are instances. Person is an instance, Name is an instance, and JOHN is an instance of their respective classes. We have copied the classes in Figure 1-3 and added values as leaf nodes. Comma indicates next item.

As will soon be clear, both classes and instances are inscriptions, and they look alike. Hence, the Person-s in Figure 1-3 and Person in Figure 1-4 are three different inscriptions; the first is a class, the two last are instances. The Person-s in Figure 1-4 are copies of the inscription in Figure 1-3. The source inscription, eg. in Figure 1-3, is the class of its copies.

We need all the information in Figure 1-4 for creating the example simple computer screen in Figure 1-2. This structure may be a complete statement about JOHN and his Loved-Person.

- *Observation 1c: Predicate Calculus does not ensure that each statement contains all information of the syntax tree to produce complete alphanumeric and graphical screens.*

In Predicate Calculus, we may add more statements, but neither their syntax nor their contents will be exactly what we need for producing our screens.

Note that so far we are concerned with defining the syntax of the data. Later we will address what is described by the data. We will also add some more details before we can fully know what is stated.

Predicate Calculus presents leaf nodes of its syntax trees, similar to the leaves of syntax trees for natural language statements, using rewriting grammars. Contrary to this, we will need each term of the entire syntax tree to produce the presentation in Figure 1-2.

We need to include entities in Figures 1-3 and 1-4, because the attributes are defined within their entity, as the attribute name tags are not globally unique. Hence, the entire expression in Figure 1-5 is needed to identify JOHN. The dog JOHN is another entity, denoted by Dog (Name (JOHN)).

Person (Name (JOHN)), Person (Name (MARY))

Figure 1.5 Full identification of JOHN and MARY

Loved-person is an entity that is defined within the scope of a Person only, and its name tag is local to Person, as well. Hence, the full identification of the Loved-person MARY is:

Person (Name (JOHN), Loved-person (Name (MARY)))

Figure 1-6 Full identification of MARY

Note that we have not yet defined that the Loved-person MARY refers to a Person.

Note that even if Loved-person is subordinate to Person, we will additionally need the branch Name (JOHN) in order to identify the instance MARY.

We may add a dot notation to Predicate Calculus. Then the parentheses in Figure 1-6 may be replaced by dots, and we get two concatenated terms Person.Name.JOHN and Person.Loved-person.Name.MARY. We may put these concatenated terms into the predicate in Figure 1-1. However, Person.Loved-person.Name.MARY does not denote an entity in our Universe of Discourse, as both JOHN and BILL may have a Loved-Person MARY, we may want to distinguish them, and have different property values associated with each. The two Loved-Person MARY may or may not refer to the same Person.

Predicate Calculus is not appropriate to identify MARY; we will need the full information of the syntax tree, with the branch Name (JOHN) as shown in Figure 1-6, to identify MARY.

- *Observation 1d: Dot notation is not sufficient to state the branches of the syntax tree required to identify terms.*

Note in Figure 1-6 that Name (JOHN) and Loved-person (Name (MARY) are parallel branches.

The shown complete statement in Figure 1-6 does not include any predicate. Predicates map from an ordered set of terms to truth-values. When making statements, we normally do not need truth-values, and we do not map to truth-values. Screens of database applications is a proof of this claim.

2 Scopes within References

In Figure 2-1 we extend our example with some more information, in order to present other aspect of a complete statement.

Country					
Name	Person			Person	
	Name	Height	Loved-person	Name	Hair-colour
			Identifier		
NORWAY	JOHN	190	1	MARY	RED
			2	ANN	BLACK
	MARY	170			
	ANN	180			

Figure 2-1 Example computer screen

The line shifts in the headings of Figure 2-1 shows containment.

In column two Figure 2-1 we have assumed that a Person is defined within a Country only; therefore, we list the three Person-s within NORWAY, and we have added a Height attribute to each Person.

At the right hand side of Figure 2-1 we have shown that a Loved-person may refer to a Person. We have added new columns with a Person's Name and Hair-colour, as we want a Loved-person to be a role that refers to this other Person with these attributes.

Column two at the left side lists all Person-s. Column two at the right side refers to some Person-s.

We have introduced a new Identifier of Loved-person; this identifier needs not be the same as the Name of the other Person, in the second column from the right.

Note that both Country and Person contain Name-s; both these Name-s are defined local to their entity, and these Name-s have nothing in common, neither meaning, syntax nor value set.

The structure of the classes in the headings in Figure 2-1 may be stated as follows:

```
Country (Name, Person (Name, Height, Loved-person (<> Loved-person ‘Person ‘Country
/(Person (Name, Hair-colour)), Identifier))).
```

Figure 2-2 Specification of the screen in Figure 2-1

In Figure 2-2 we have introduced a condition (<>) on Loved-person that refers up to superior terms by hyphens (‘) to Person and Country and down to the other Person. The backslash indicates continuation from the previous line.

The shown navigation within the condition ensures that the Person MARY is found in the same Country as JOHN – see below. If the referenced Person is not found, the Loved-person will be deleted. Note that the conditions do not produce truth-values, but result in continuation or deletion.

The Identifier attribute is a subordinate attribute to the Loved-person having the condition. This will become clearer in the two-dimensional notation in Figure 2-4 and in the normalized notation in Figure 3-1.

Note that the navigation in Figure 2-2 is between entities, and does not refer to the Name-s of Person and Country. The navigation ensures that the other Person is in the same Country as of the first Person.

The structure of the instances within the population may look like this:

```
Country (Name (NORWAY), Person (Name (JOHN), Height (190), Loved-person (<> Loved-person
/‘Person ‘Country (Person (Name (MARY), Hair-colour (RED))), Identifier (1)), Loved-person (<>
/Loved-person ‘Person ‘Country (Person (Name (ANN), Hair-colour (BLACK))), Identifier (2)))),
/Person (Name (MARY), Height (170), Person (Name (ANN), Height (180).
```

Figure 2-3 Example contents of the variable fields in Figure 2-2

Figure 2-3 contains three Person instances and two Loved-person instances.

We see that the statements become cluttered when using the one-dimensional notation. Figure 2-4 shows the classes in a two-dimensional notation:

```
Schema1
        Country
        Name
        Person
                Name
                Height
                Loved-person
                        <> Loved-person ‘Person ‘Country (Person (Name, Hair-colour))
                        Identifier
```

Figure 2-4 Example Schema containing classes

Figure 2-4 shows that the two-dimensional notation can do away with both parentheses and commas.

Figure 2-5 shows the corresponding instances:

```
Population3
        Country
                Name
                        NORWAY
                Person
                        Name
                                JOHN
                        Height
                                190
                        Loved-person
                                <> Loved-person ‘Person ‘Country (Person (Name (MARY), Hair-colour (RED)))
                                Identifier
                                        1
                        Loved-person
                                <> Loved-person ‘Person ‘Country (Person (Name (ANN), Hair-colour (BLACK)))
                                Identifier
                                        2
                Person
                        Name
                                MARY
                        Height
                                170
                Person
                        Name
                                ANN
                        Height
                                180
```

Figure 2-5 Example Population containing instances

In the alphanumeric screen in Figure 2-1 we have presented the values only, ie. the leaf nodes. In a graphic screen of the same entities we may only present an icon for each Person, and no Name attribute. We may add a tag to each term to indicate which one to suppress or present in the concrete syntax.

We see that the condition with navigation from Loved-person to MARY tells that MARY is a Person within the same Country as JOHN.

In Predicate Calculus we may restrict the scope of JOHN and MARY to the same variable country.

∀ person ∃ country (person ∈ country) (Loves (person, loved-person) → (loved-person ∈ country))

Figure 2-6 Constraining the two persons to the same country

Note that the terms starting with lower case letters are variables, which will be replaced by constant terms in our alternative language. In our alternative language in Figures 2-4 and 2-5 we will do away with the distinction between constants and variables. A class in the schema acts as a 'variable' of its instances, and they all look alike. In addition, we utilize the tree structure of the data, eg. that an attribute contains the value.

The expression in Figure 2-6 is a long way from what is stated in Figure 2-4.

When comparing Figure 2-4 and 2-6, we observe another aspect of the navigation: The navigation is a means to state scopes, similar to use of bound universal and existential quantifiers in Figure 2-6. If we want to allow JOHN to love a Person in another Country, we would have to navigate to a common entity containing this other Country.

A combination of the two-dimensional and one-dimensional notation reduces the use of space in Figure 2-5. Most left-hand and right-hand parentheses will not be needed. See Figure 2-7.

```
Population3
        Country
                Name (NORWAY
                Person
                        Name (JOHN
                        Height (190
                        Loved-person
                                <> Loved-person 'Person 'Country (Person (Name (MARY), Hair-colour (RED
                                Identifier (1
                        Loved-person
                                <> Loved-person 'Person 'Country (Person (Name (ANN), Hair-colour (BLACK
                                Identifier (2
                Person
                        Name (MARY
                        Height (170
                Person
                        Name (ANN
                        Height (180
```

Figure 2-7 Example Population combining two-dimensional and one-dimensional notations

- *Observation 2a: In the alternative language, we will replace quantifiers in Predicate Calculus by conditions with navigation.*

Screens, as of Figure 2-1 are often interpreted as relations. We discourage this interpretation.

- *Observation 2b: Both schema and population data of screens form trees, ie. not flat tables as of normalized relations.*
- *Observation 2c: Property attribute values within screens are not functionally dependent on identifier attribute values; all attributes are functionally dependent on the superior node in the syntax tree, eg. on their entity.*

3 On Normal Forms

In the previous sections, we have discussed the contents of compound statements. This section will introduce normalized statements, ie. not compound statements. Here each piece of information will appear only once.

The same piece of information may appear at different places in each compound statement. Therefore, compound statements are used to express the abstract syntax of screens that have this need of presenting information. The abstract syntax of the screens is expressed in Contents schemata and Contents populations. In this paper, we will not present a language for the concrete syntax of screens other than showing the example screen pictures with headings and values.

The normal form of data is used to express the common External terminology schema and common External terminology population across all compound statements, ie. acroos all screens, of an application. The External terminology schema defines the common terminology and grammar of this application.

Since the terms are defined local to each other, the grammar and the terms cannot be defined separately. The terms are defined within the grammar, ie. within the syntax tree. Also, the use of local terms implies that we cannot distinguish the production rules from the syntax tree. We only have the syntax tree with all rules included.

Figure 3-1 shows an External terminology schema of the contents of Figures 2-2 and 2-4.

```
SCH
        Country
                Name
                Person
                        Name
                        Hair-colour
                        Height
                        Loved-person
                                <> Loved-person 'Person 'Country (Person
                                Identifier
```

Figure 3-1 Example External terminology schema

Note that both Person (Name and Person (Height are removed from the condition. They both appear under Person only.

Figure 3-2 shows an External terminology population of the contents of Figure 2-3, 2-5 and 2-7.

```
POP
        S<>S ‘ ‘ (SCH
        Country
                Name (NORWAY
                Person
                        Name (JOHN
                        Hair-colour
                        Height (190
                        Loved-person
                                <> Loved-person ‘Person ‘Country (Person (Name (MARY
                                Identifier (1
                        Loved-person
                                <> Loved-person ‘Person ‘Country (Person (Name (ANN
                                Identifier (2
                Person
                        Name (MARY
                        Hair-colour (RED
                        Height (170
                Person
                        Name (ANN
                        Hair-colour (BLACK
                        Height (180
```

Figure 3-2 Example External terminology population

In Figure 3-2, we in the second line have added a schema reference from the population POP via its subordinate S and a common superior and then down to the schema SCH shown in Figure 3-1.

From Loved-person in Figure 3-1, we make a reference to the other entity Person only. In Figure 3-2, we have added Name and value to this reference, such that we can refer to a specific individual Person. Alternatively, we could have made a reference by navigation via the Person-s to the right Person. See below in Figure 3-3. At the user interface, the references between entities may be made by pointing at icons – eg. in a map - representing the Person, without presenting any Name or sequence of Person-s. In the alternative language, we allow for this.

Finally, Hair-colour is moved from Loved-person in the Contents layer to become an attribute of Person in the External terminology layer. The Contents layer is parsing through the External terminology layer. All execution of conditions and instructions are carried out in the External terminology layer only. The contents of the Contents layer shall comply with the External terminology layer. The Contents layer shows use of the data, while the External terminology layer defines the data.

There shall be a homomorphic mapping from the contents of the Contents layer to the contents of the External terminology layer. The mappings from the populations to their corresponding schemata shall be homomorphic in both layers.

Note that in Figures 3-1 and 3-2 there is no predicate; there are only terms that exist subordinate or next to each other.

Note that in Figures 3-1 and 3-2 there are 3+1=4 Person inscriptions (with two indentations). There is no notion of similar inscriptions referring to a common string.

- *Observation 3a: Predicate Calculus assumes that similar inscription terms refer to the same string term; we assume that all inscriptions are different, even when looking identical, and they do not refer to strings.*

We will not give a full exposition of the Type-token principle of classical logic in this paper, and we will not discuss the mapping to sets.

In Predicate Calculus, we would have defined a set Persons (with the plural s) and used a variable person to range over its individual set elements. In the alternative language, we use the term Person to be the class of the individuals Person, Person and Person.

The schema in Figure 3-1 allows for creating the following population:

```
POP
        S<>S ' ' (SCH
        Country
                Name (NORWAY
                Person
                        Loved-person <> ' ' (Person, Person
                        Loved-person <> ' ' (Person, Person, Person
                Person
                Person
```

Figure 3-3 Example Population containing duplicate instances

In Figure 3-3, we in the conditions are navigating up two levels by ' ' and down to (Person, and then to the next Person-s. We do not here discuss maintenance of these references.

- *Observation 3b: In the alternative language, we are replacing references by names in Predicate Calculus by references by navigation.*
- *Observation 3c: Within screens and in the alternative language, we allow for use of significant duplicates.*

Note in Figure 3-3 that the inscription Person may be replaced by a graphical icon; they may all look alike, but are still treated as separate entities.

4 Name tags

Name tags may be separated from their data node. This is illustrated in Figures 4-1 and 4-2.

```
:                                               SCH

        :                                       Country
                :                               Name
                :                               Person
                        :                       Name
                        :                       Hair-colour
                        :                       Height
                        :                       Loved-person
                               <>               Loved-person 'Person 'Country (Person
                               :                Identifier
```

Figure 4-1 Example External terminology schema separating name tags

The name tags in Figures 4-1 and 4-2 appear as the first element in the list of subordinate items of the data node.

```
:                                     POP

      :                               S
            <>                        S ‘ ‘ (SCH
      :                               Country
            :                         Name
                  :                   NORWAY
            :                         Person
                  :                   Name
                        :             JOHN
                  :                   Hair-colour
                  :                   Height (190
                  :                   Loved-person
                        <>            Loved-person ‘Person ‘Country (Person (Name (MARY
                        :             Identifier
                              :       1
                  :                   Loved-person
                        <>            Loved-person ‘Person ‘Country (Person (Name (ANN
                        :             Identifier
                              :       2
            :                         Person
                  :                   Name
                        :             MARY
                  :                   Hair-colour
                        :             RED
                  :                   Height
                        :             170
            :                         Person
                  :                   Name
                        :             ANN
                  :                   Hair-colour
                        :             BLACK
                  :                   Height
                        :             180
```

Figure 4-2 Example External terminology population separating name tags

We will even allow both schemata and populations to exist without name tags.

```
:                                     POP

      :                               S <> S ‘ ‘ (SCH
      :
            :
                  :
                        <>            ‘ ‘ ( ,
                  :
                        <>            ‘ ‘ ( , ,
            :
            :
```

Figure 4-3 Example External terminology population without name tags

Figure 4-3 shows the same population as of Figure 4-2, but without attributes and name tags.

The distinctions between entity, attribute group, attribute, value etc. are not essential for the interpretation and execution of the data. Only the tree structure matters. An entity may contain multiple subordinate entities of the same class or different classes, and contain multiple subordinate attribute groups or attributes of the same class or different classes. An attribute may have multiple values etc. Explicit constraints may constrain the cardinality of each.

5 Denotations

The terms in the External terminology layer may denote phenomena or concepts. This is shown for entities in Figure 5-1.

```
:                           System

    :                       EXTERNAL LAYER
        :                   Denotes <> ‘ ‘ (PHENOMENON LAYER
        :                   SCH
            :               Denotes <> ‘ ‘ ‘ (PHENOMENON LAYER (SCH
            :               Country
                :           Denotes <> ‘ ‘ ‘ ‘ (PHENOMENON LAYER (SCH (
                :           Person
                    :       Denotes <> ‘ ‘ ‘ ‘ ‘ (PHENOMENON LAYER (SCH ( (
                    :       Loved-person <> ‘ ‘ (Person
                            : Denotes <> ‘ ‘ ‘ ‘ ‘ ‘ (PHENOMENON LAYER (SCH ( ( (
        :                   POP
            :               Denotes <> ‘ ‘ ‘ (PHENOMENON LAYER (POP
            :               S <> S ‘ ‘ (SCH
            :               Country
                :           Denotes <> ‘ ‘ ‘ ‘ (PHENOMENON LAYER (POP (
                :           Person
                    :       Denotes <> ‘ ‘ ‘ ‘ ‘ (PHENOMENON LAYER (POP ( (
                    :       Loved-person <> ‘ ‘ (Person,
                            : Denotes <> ‘ ‘ ‘ ‘ ‘ ‘ (PHENOMENON LAYER (POP ( ( ( ,
                    :       Loved-person <> ‘ ‘ (Person, ,
                            : Denotes <> ‘ ‘ ‘ ‘ ‘ ‘ (PHENOMENON LAYER (POP ( ( ( , ,
                :           Person
                    :       Denotes <> ‘ ‘ ‘ ‘ ‘ (PHENOMENON LAYER (POP ( ( ,
                :           Person
                    :       Denotes <> ‘ ‘ ‘ ‘ ‘ (PHENOMENON LAYER (POP ( ( , ,

    :                       PHENOMENON LAYER
        :                   SCH
            :
                :
                    :        <> ‘ ‘ (
        :                   POP
            :               S <> S ‘ ‘ (SCH
            :
                :
                    :        <> ‘ ‘ ( ,
                    :        <> ‘ ‘ ( , ,
                :
                :
```

Figure 5.1 Denotation mapping from terms to phenomena

We observe in Figure 5-1 that the structure of terms is isomorphic to the structure of phenomena, but the mapping need not be onto either way. There may be phenomena for which we have no term, and there may be terms for which there is no phenomenon.

The phenomena need not have any name tag, but we are free to add name tags to phenomena, as this makes the references simpler.

The phenomena need not have identifier attributes, but we may add such attributes to simplify the referencing from the External terminology layer.

We note that the phenomena have classes, just like the terms in the External terminology layer.

The phenomena are themselves data inside some observer. We will come back to interpretation of the phenomena.

- *Observation 5a: We do not map string terms to sets; we map inscription terms to phenomena or concepts that both are data inscriptions inside some observer.*
- *Observation 5b: We will abandon all axioms of Set theory, as we interpret the terms against a phenomenon or concept layer that is an abstract syntax of the external terminologies.*

See more on this in [8].

We may now apply our requirement on isomorphic mapping on Predicate Calculus.

If a predicate shall describe anything, there must be a corresponding predicate functor among the phenomena.

- *Observation 5c: Most Universes of Discourse do not contain predicate functors; hence, the predicates cannot describe phenomena in these Universes of Discourse.*
- *Observation 5d: It is permissible to have data that do not describe anything; predicates do not describe anything.*
- *Observation 5d: The sequence of terms in a predicate, ie. the nested ordered pairs, may describe a corresponding sequence of phenomena in the Universe of Discourse.*
- *Observation 5e: Observation 5d requires that the phenomena are observed from outside the Universe of Discourse, and do not belong to classes.*

6 Recursion

The schema reference may be used recursively. This means that both the schema, eg. SCH, and its population, eg. POP, may contain additional schema references. Schema references inside a schema are shown in Figure 6-1.

```
:                                       SCH
      :                                 Country
            :                           Name
                  S<>                   S ‘ ‘ ‘ (Types (Letters
            :                           Person
                  :                     Name
                        S<>             S ‘ ‘ ‘ ‘ (Types (Letters
                  :                     Hair-colour
                        S<>             S ‘ ‘ ‘ ‘ (Types (Letters
                  :                     Height
                        S<>             S ‘ ‘ ‘ ‘ (Types (Digits
                  :                     Loved-person
                        <>              Loved-person ‘Person ‘Country (Person
                        :               Identifier
      :                                 Types
            :                           Letters
                  :                     a, b, c, d, e, f, g, h, I, j, k …
            :                           Digits
                  :                     0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

Figure 6-1 Example External terminology schema using recursion

The schema reference from Country (Name to Letters will allow the Name to contain any sequence of letters. Similarly, for the other attributes. If we want each attribute to contain one value that contain the letters, we will have to state the schema reference from this value that is depicted in Figure 4-2.

We will here not show any example of schema references within populations, but we will give an indication: A company has a product catalogue. This is a population in a database. Each customer has a selection of products according to the specification in the catalogue. This is provided by stating a schema reference from (a contract of) the customer to the product catalogue.

We will show how recursive use of a schema reference may allow creation of a tree of data. Suppose we want to allow a Product to be decomposed into subordinate Products recursively in a strict tree structure.

```
:                                       Product
      :                                 Product
            S <>                        S ‘Product ‘Product
```

Figure 6-2 Recursive definition of a tree structure

Figure 6-2 allows the subordinate Product to inherit any property of the superior Product, including its subordinate Product and its schema reference. Hence, you may create a tree of Product classes.

A population node may have several schema references in a sequence. The instantiation takes place in this sequence, and may only use a subset of the schemata.

When instantiating from a schema, only classes from this schema may be instantiated. The schema states everything that is permissible.

- *Observation 6a: Predicate Calculus has no means to state that other predicates are not permissible, as it cannot refer to inscriptions as of our alternative language, and does not support deletions.*

The schema notion ensures that you can instantiate no other statement than what is covered by the schema. When the schema is about <u>Person</u>-s only, you cannot create a population about <u>Dog</u>-s.

A schema may contain references to its meta-schema, which allows for more than the schema. This implies that instances from a schema may be decorated with instances from its meta-schema. This implies that Letters may be written in different fonts, colours and sizes, as long as they satisfy the basic forms defined in the schema. Extensional definitions of predefined fonts are not needed.

7 Logical connectives

Every list contained in a schema states a disjunction of what may be copied into instances. Hence, rather than S <> S, we may write v <> v.

This disjunction is not a logical disjunction, but refers to a list from which selections can be made, and each selection may be copied into instances multiple times. v refers between two terms - the population and the schema -, but applies for all subordinate terms.

Sometimes we want to instantiate only one of the permissible classes in the schema. This may be achieved by stating a cardinality constraint on the instances. Alternatively, we may constrain the schema reference by indicating an exclusive or, ie. ⊕ <> ⊕ or T <> T. T is read as ‘type’.

Every list contained in a population states a conjunction of what is actually the case. Sometimes the statements are about phenomena in a Universe of Discourse, but the statements need not be so.

We have already seen that there is no absolute distinction between populations and schemata; they are schemata and populations only relative to each other. Hence, we do not state conjunctions and disjunctions in an absolute sense. The same list may act both as a conjunction and a disjunction. Multiple populations may have the same schema. The roles are stated in references between the lists.

- *Observation 7a: Logical connectives in Predicate Calculus are replaced by schema references between lists; the schema states a disjunction of alternatives.*

We note that natural language expressions using ‘and’ are in Predicate Calculus interpreted as a binary conjunction. We replace the ‘and’ by lists. So also for ‘or’.

In the alternative language we do not use logical equations, and we do not use axioms, as of Set theory.

- *Observation 7b: We replace axioms by schemata containing classes that act as templates for their instances.*

8 Predicates

A predicate is a functor that maps from a series of terms to a truth-value. See the example one-placed predicates in Figure 8-1.

Person (JOHN) ∧ Person (MARY) ∧ Person (ANN)

The predicates in this populations map to the following truth-values

1 ∧ 1 ∧ 1

that are calculated into

1

Figure 8-1 Predicates map into truth-values

- *Observation 8a: Predicates map to truth-values that are not appropriate for expressing the structure of statements.*
- *Observations 8b: Truth-values express meta-statements about the statements; they are and should not be part of the statements.*
- *Observation 8c: Predicates have unsatisfactory contents and syntax for expressing the structure of statements.*
- *Observation 8d: Predicate Calculus splits the statements into fragmented chunks.*
- *Observation 8e: Predicate Calculus uses variables to state general statements, and the variables are replaced by constants in concrete applications.*
- *Observation 8f: The alternative language states generic laws through disjunctions of constant classes in schemata, the classes are copied into instances in concrete applications, and constant values are attached to constant attributes; there is no variable.*

9 Functions

Conditions (<>) with navigation identify the arguments of a function. Whatever is found at the tip of the referenced branches are taken as arguments.

Instructions (><) with navigation identify the places for the function values.

The elementary function is copying (>< : <>): Whatever is found in the arguments are copied to the function value.

Functions are specified in schemata and are copied into every corresponding instance, and they are executed when this instance is created. This means that a function on an entity is executed when this entity instance is created. A function on a value is executed when this value instance is created, etc.

From a syntactical and execution point of view there is no distinction between instances and classes.

Any node in the data tree can be a function. There is no restricted name set for functions. However, there may be predefined functions. The functioning of a function is defined by one or more schema references.

We will in this paper not show predefined functions. However, cardinality constraints (C(min, max)<>) will frequently be used. Also, identification attributes will be tagged by the Id<> function.

- *Observation 9a: Predicate Calculus distinguishes between terms, predicates and functions; we do not.*
- *Observation 9b: We use the same name space for terms, predicates and functions.*
- *Observation 9c: Predicate Calculus evaluates truth; we replace truth by rewriting.*

We may define a schema where all names are unique within this schema, and apply this schema for specialized nodes in the population.

A full text on functions in the alternative language will need much more space than what is available in this paper. The instruction part of a function may use the same navigation path as of its condition, and the navigation path may use wild cards.

10 World view

- *Observation 10a: Predicate Calculus applies a flat world view, where all terms, predicates, functions and denoted sets appear at the same global level - being seen by an outside-positioned observer.*
- *Observation 10b: The alternative language replaces the outside positioned observer by an observer moving and interpreting the phenomena inside its Universe of Discourse.*

We let the observer move inside its Universe of Discourse; it identifies phenomena and creates descriptions when being on the move.

- *Observation 10c: Predicate Calculus is interpreted in an abstract way, where idealized strings and sets have an absolute existence; we replace this by data being defined relative to each other.*

We let each phenomenon be an observer of its contained phenomena; hence, phenomena exist only relative to each other, and data are created relative to other data only.

- *Observation 10d: Relational mathematics distinguish entities, relationships and roles, we do not.*

Our entities are roles only, and we state conditions between roles. We do away with entities and relationships. See more on this in [8].

11 Observer

We allow any phenomenon to be an observer of other phenomena. A contained phenomenon is observed from its superior phenomenon. See more on this in [8].

When interpreting data in an observer, we use a seven layer architecture. The seven layers transform data between two media while doing translation between two terminologies. The architecture deals with inscriptions only; there is no abstraction, strings or sets. The ur element of any inscription is a pixel. All the colons (:) in this paper show pixels.

- *Observation 11a: All phenomena are organizations of pixels inside some observer; we do not allow for non-syntactical constructs, such as sets, infinity, continuity, points and lines.*

The layers are:

1. The Layout layer (L), ie. the concrete syntax of each screen of the user interface
2. The Contents layer (C), ie. the abstract syntax of each screen of the user interface
3. The External terminology layer (eT), ie. the concrete terminology and grammar of the application across all screens
4. The Concept layer (O); ie. the abstract syntax of the application across all terminologies; may also be called the Phenomenon layer
5. The Internal terminology layer (iT), ie. the concrete terminology and grammar of the application used at internal media

6. The Distribution layer (D), ie. the abstract syntax used for communication to the internal media
7. The Physical layer (P), ie. the concrete syntax used at the internal media

In this paper we have shown examples of the first four layer. These layers replace the interpretations of Predicate Calculus. See more on this in [8]. The architecture is depicted in Annex III.

- *Observation 11b: The Data transformation architecture is replacing interpretation according to the Type-token principle and Set theory.*

The alphanumeric screen pictures in Figures 1-2 and 2-1 will need means to insert and delete data.

- *Observation 11c: Predicate Calculus does not support insertion and deletion.*
- *Observation 11d: The alternative language delivers insertions through copying, and delivers deletion if conditions are failing.*

12 Conclusion

To define and inscribe the syntax of an alphanumeric screen is an unusual application of Predicate Calculus. However, it is a good test of its applicability, as this Universe of Discourse is explicitly inscribed on paper or some other medium. This allows for referencing from inscriptions to inscriptions without any imagination about concepts. We have found Predicate Calculus to be inadequate for this use.

If Predicate Calculus is inadequate for defining and inscribing screens, what should then be its application domains? The primary application may be as a common reference language. We believe that a language based on rewriting would provide a better foundation than a calculus on truth-values. These issues are addressed in [8].

We propose an alternative language to Predicate Calculus. The following observations indicate how notions in the other language replace Predicate Calculus. The bullets follow the sequence in which they appear in the paper. The bullets are not independent. There is no one-to-one correspondence between notions in the two languages.

The Observation numbering, ie. Observation nx, refers to section n bullet x.

1. *Observation 1a: Predicate Calculus assumes that all constant terms are globally unique; this does not support complex naming structures both depending on the class and superior terms, as of the alternative language.*
2. *Observation 1b: When defining data, there is no need for predicates, as we do not map to truth-values.*
3. *Observation 1c: Predicate Calculus does not ensure that each statement contains all information of the syntax tree to produce complete alphanumeric and graphical screens.*
4. *Observation 1d: Dot notation is not sufficient to state the branches of the syntax tree required to identify terms.*
5. *Observation 2a: In the alternative language, we will replace quantifiers in Predicate Calculus by conditions with navigation.*
6. *Observation 2b: Both schema and population data of screens form trees, ie. not flat tables as of normalized relations.*

7. *Observation 2c: Property attribute values within screens are not functionally dependent on identifier attribute values; all attributes are functionally dependent on the superior node in the syntax tree, eg. on their entity.*
8. *Observation 3a: Predicate Calculus assumes that similar inscription terms refer to the same string term; we assume that all inscriptions are different, even when looking identical, and they do not refer to strings.*
9. *Observation 3b: In the alternative language, we are replacing references by names in Predicate Calculus by references by navigation.*
10. *Observation 3c: Within screens and in the alternative language, we allow for use of significant duplicates.*
11. *Observation 5a: We do not map string terms to sets; we map inscription terms to phenomena or concepts that both are data inscriptions inside some observer.*
12. *Observation 5b: We will abandon all axioms of Set theory, as we interpret the terms against a phenomenon or concept layer that is an abstract syntax of the external terminologies.*
13. *Observation 5c: Most Universes of Discourse do not contain predicate functors; hence, the predicates cannot describe phenomena in these Universes of Discourse.*
14. *Observation 5d: It is permissible to have data that do not describe anything; predicates do not describe anything.*
15. *Observation 5d: The sequence of terms in a predicate, ie. the nested ordered pairs, may describe a corresponding sequence of phenomena in the Universe of Discourse.*
16. *Observation 5e: Observation 5d requires that the phenomena are observed from outside the Universe of Discourse, and do not belong to classes.*
17. *Observation 6a: Predicate Calculus has no means to state that other predicates are not permissible, as it cannot refer to inscriptions as of our alternative language, and does not support deletions.*
18. *Observation 7a: Logical connectives in Predicate Calculus are replaced by schema references between lists; the schema states a disjunction of alternatives.*
19. *Observation 7b: We replace axioms by schemata containing classes that act as templates for their instances.*
20. *Observation 8a: Predicates map to truth-values that are not appropriate for expressing the structure of statements.*
21. *Observations 8b: Truth-values express meta-statements about the statements; they are and should not be part of the statements.*
22. *Observation 8c: Predicates have unsatisfactory contents and syntax for expressing the structure of statements.*
23. *Observation 8d: Predicate Calculus splits the statements into fragmented chunks.*
24. *Observation 8e: Predicate Calculus uses variables to state general statements, and the variables are replaced by constants in concrete applications.*

25. *Observation 8f: The alternative language states generic laws through disjunctions of constant classes in schemata, the classes are copied into instances in concrete applications, and constant values are attached to constant attributes; there is no variable.*
26. *Observation 9a: Predicate Calculus distinguishes between terms, predicates and functions; we do not.*
27. *Observation 9b: We use the same name space for terms, predicates and functions.*
28. *Observation 9c: Predicate calculus evaluates truth; we replace truth by rewriting.*
29. *Observation 10a: Predicate Calculus applies a flat world view, where all terms, predicates, functions and denoted sets appear at the same global level - being seen by an outside-positioned observer.*
30. *Observation 10b: The alternative language replaces the outside positioned observer by an observer moving and interpreting the phenomena inside its Universe of Discourse.*
31. *Observation 10c: Predicate Calculus is interpreted in an abstract way, where idealized strings and sets have an absolute existence; we replace this by data being defined relative to each other.*
32. *Observation 10d: Relational mathematics distinguishes entities, relationships and roles; we do not.*
33. *Observation 11a: All phenomena are organizations of pixels inside some observer; we do not allow for non-syntactical constructs, such as sets, infinity, continuity, points and lines.*
34. *Observation 11b: The Data transformation architecture is replacing interpretation according to the Type-token principle and Set theory.*
35. *Observation 11c: Predicate Calculus does not support insertion and deletion.*
36. *Observation 11d: The alternative language delivers insertions through copying, and does deletion if conditions are failing.*

We outline an alternative language to Predicate Calculus. The alternative language is called Existence logic [8].

Existence logic needs only one symbol, colon (:), which corresponds to a pixel. Copies of this symbol are organized in a tree of lists in three dimensions:

- Subordinate items (() are observed from their superior item
- Next item (,) indicate items observed in parallel
- Conditions (<>) indicate algorithmic functions that include navigation

Existence logic needs no alphabet, but may define any alphabet. Existence logic needs no name, but can add names that are used for search. No logical connective, truth-value or quantifier is needed.

When projecting the three dimensions onto two or one dimensions, special symbols like (<>), (,) and (() are needed. See more on Existence logic in [8].

In summary:

Predicate Calculus is the latin of formal languages.

Set Theory is the theology of formal languages.

In a modern setting we need none of them.

References

[1] ***Z.351*** (03/93). ***Data Oriented Human-Machine Interface Specification Technique – Introduction***. http://www.itu.int/rec/T-REC-Z.351-199303-I

[2] ***Z.352*** (03/93). ***Data Oriented Human-Machine Interface Specification Technique – Scope, Approach and Reference Model.*** http://www.itu.int/rec/T-REC-Z.352-199303-I

[3] ***M.1401*** (05/2005). ***Formalization of interconnection designations among operators' telecommunication networks***. http://www.itu.int/rec/T-REC-M.1401-200505-S

[4] ***M.1402*** (05/2012). ***Formalization of data for service management.*** http://www.itu.int/rec/T-REC-M.1402-201205-I

[5] ***M.1403*** (08/2007). ***Formalization of generic orders.*** http://www.itu.int/rec/T-REC-M.1403-200708-I

[6] ***M.1404*** (08/2007). ***Formalization of orders for interconnections among operators' networks***. http://www.itu.int/rec/T-REC-M.1404-200708-I

[7] ***M.1405*** (08/2007). ***Formalization of orders for service management among operators.*** http://www.itu.int/rec/T-REC-M.1405-200708-I

[8] ***Meisingset Arve*** *(03/2017).* ***Formal Description of Anything in the Universe.*** http://www.movinpics.com

[9] ***MSTR-NREG*** (09/2016). ***Telecom Network Registration.*** Technical report, http://www.itu.int/pub/T-TUT-TLCMGT-2016.

[10] ***Z.601*** (02/2007). ***Data architecture of one software system.*** http://www.itu.int/rec/T-REC-Z.601-200702-I

Annex I Graphic Notation

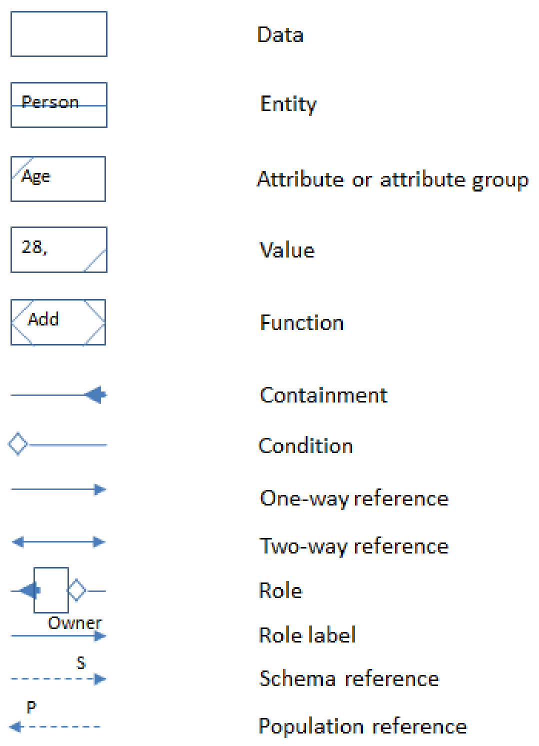

Figure AI.1 Graphic symbols

Also, attribute groups and attributes may be placed inside the boxes, as shown in the next Figure. Indentations are used to show the structure.

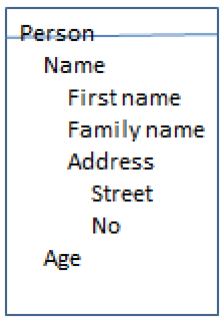

Figure AI.2 Attribute groups and attributes within entities

Cardinality constraints, C(n, m), may be added inside and outside boxes, with or without the C. Also, the Identifier constraint Id may be added.

Often the root node of the External terminology schema graph is not depicted. Hence, the schema may have the appearance of a network, even if it is a tree, with references between the nodes.

Annex II Example Documentation

External terminology schema

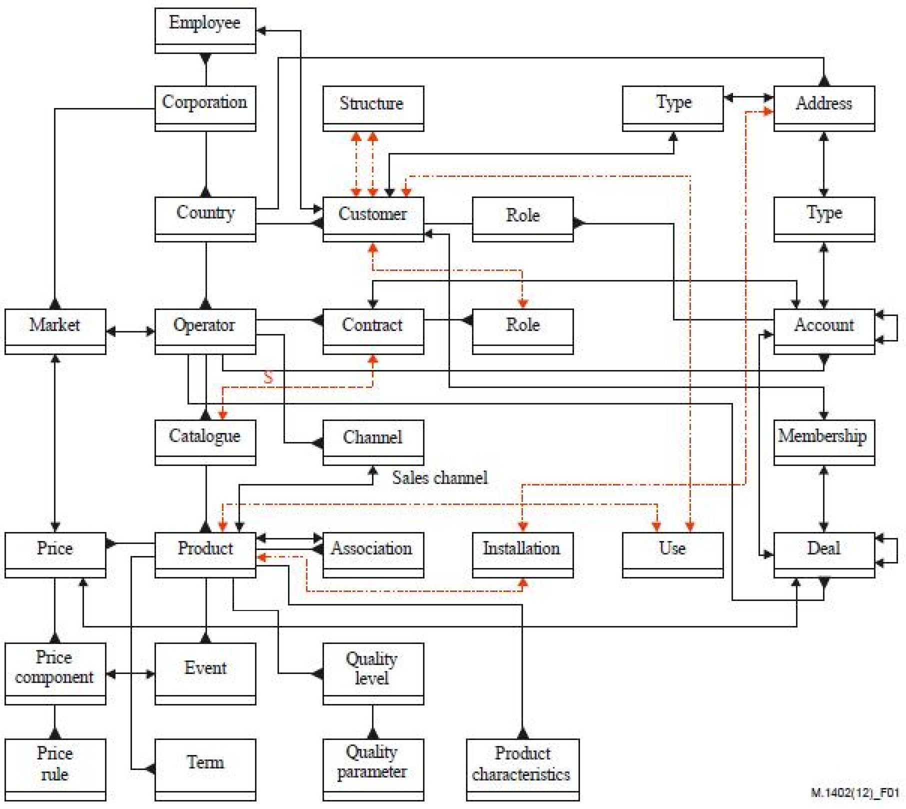

Figure AII.1 – External terminology schema graph for service management

Annex III Interpretation

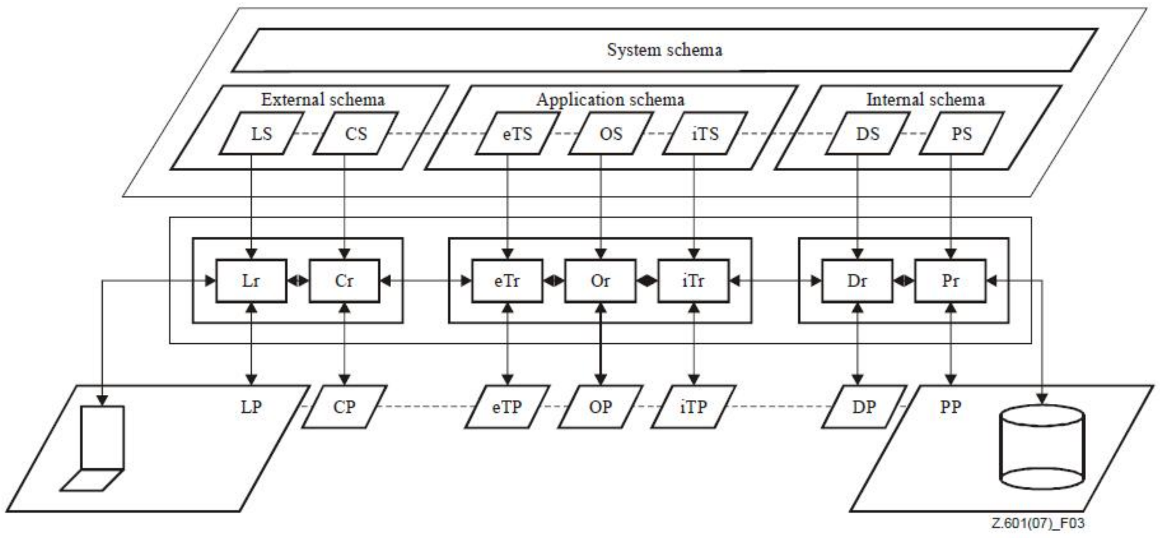

Figure AIII.1. The data transformation architecture

See explanation of the layers in section 11. The upper row depicts schemata, the middle row depicts processors, and the lower row depicts populations. Media appear in both end.