

# A Data Quality Metamodel Extension to CWM

Pedro Gomes, José Farinha, Maria José Trigueiros

ISCTE

Av. das Forças Armadas, Edif. ISCTE  
1600-083 Lisboa, Portugal

jose.farinha@iscte.pt

## Abstract

The importance of metadata has been broadly referred in the last years, mainly in the field of data warehousing and decision support systems. Contemporarily, in the adjacent field of data quality, several approaches and tools have been set out for the purpose of data profiling and cleaning. However, little effort has been made in order to formally specify metrics and techniques for data quality in a structured way. As a matter of fact, little relevance has been assigned to metadata regarding data quality and data cleaning issues. This paper aims at filling this gap, proposing a conceptual metamodel for data quality and cleaning, both applicable to operational and data warehousing contexts. The presented metadata model is integrated with OMG's CWM, offering a possible extension of this standard toward data quality.

*Keywords:* Data quality, Data cleaning, Data warehouses, Metadata, Metamodel, Standards, CWM.

## 1 Introduction

The effects of poor data quality on an organization productivity and profitability have been proficiently recognized and discussed both in research and industry contexts. Research efforts, tools, methods and several strategies have been taking place to cope with the problem and the results have been rather satisfactory. However, the definition of standards that allow the reuse and portability of solutions across different tools and application contexts has been absent from the data quality arena.

Contemporarily, metadata has been “le plat du jour” in the IT field, since information, processes and systems have reached complexity levels that no longer can be managed or used in an informal way. Mainly in data warehousing, business intelligence and several closely related fields metadata has been recognized as an issue of extreme relevance, due to the particular complexity of data and processes in these application fields.

Also, data quality is particularly critical in business intelligence application contexts, due to its impact on decision-making effectiveness.

In spite of their common affinity with data warehousing, metadata and data quality are topics that haven't been approached in an integrated way.

By having data quality information stored under a metadata framework and with the appropriate exploration tools in place, organizations would become able to obtain accurate data quality indicators regarding their data sources. Due to the structured nature of metadata, organizations would also be provided with the means to automatically detect and back trace erroneous information produced from data repositories, consequently improving decision performance and reducing operational costs. If metadata is stored under a common data model, metadata interoperability between tools further harnesses the aforementioned benefits (among others).

This paper approaches the data quality through a metadata perspective by proposing a metamodel extension to OMG's standard Common Warehouse Metamodel (CWM), providing modelling guidelines for storing formal specifications of data quality rules.

This paper adopts a definition of data quality that closely follows the one presented in Kimball & Caserta (2004). Data quality is achieved whenever data is *accurate*, considering that *accuracy* is a concept that encompasses the following properties: *correctness*, *unambiguity*, *consistency*, *completeness* and *timeliness*.

*Correctness:* data is correct if it conveys a lexically, syntactically and semantically correct statement – e.g., the following pieces of information are not correct: “Germany is an African country” (semantically wrong); Book.title: ‘De la Mancha Don Quixote’ (syntactically wrong); UK's Prime Minister: ‘Toni Blair’ (lexically wrong).

*Unambiguity:* data is not ambiguous if it allows only one interpretation – anti-example: Song.composer = ‘Johann Strauss’ (father or son?).

*Consistency:* data is consistent if it doesn't convey heterogeneity, neither in contents nor in form – anti-examples: Order.Payment.Type = ‘Check’; Order.Payment.CreditCard\_Nr = 4252... (inconsistency in contents); Order.requested\_by: ‘European Central Bank’; Order.delivered\_to: ‘ECB’ (inconsistency in form, because in the first case the customer is identified by the full name, while in the second case the customer's acronym is used).

*Completeness:* data is complete if no piece of information is missing – anti-example: “The Beatles were John Lennon, George Harrison and Ringo Starr”.

*Timeliness*: data is accurate if it is up to date – anti-example: “Current president of the USA: Bill Clinton”.

Additionally to these accuracy factors, some other considerations are required in order to fully define quality in data. As stated in Olson (2003): “data has quality if it satisfies the requirements of its intended use”. This implies that data quality is always defined upon a business context of application. Accordingly, the proposed metamodel provides support for expressing accuracy factors’ values by means of rules that are allowed to apply differently to different contexts, therefore being able to provide different levels of data accuracy for the same set of data, depending on the business purpose.

The proposed metamodel was conceived to provide support to both data profiling and data cleaning activity, as rules may be set up in order to detect data quality problems as well as to establish data cleaning solutions. Regarding data cleaning, metadata support is given in order to enable the ultimate goal of achieving the highest level of automation possible. Nevertheless, metadata support is also provided when user participation is required within the cleaning process.

In order to achieve applicability to data quality activities in both an operational context and in an Extract Transform and Load (ETL) context, the presented metamodel was conceived as an extension to OMG’s CWM - Common Warehouse Metamodel (OMG, 2003). CWM is a standard that enables metadata interoperability across data warehousing environments, allowing the characterization of data sources, target data warehouses and also ETL processes. Using CWM as the starting point for current paper’s piece of work enables the proposed metamodel to inherit a foundational metadata structure, while capitalizing upon a globally recognized standard. Currently, CWM doesn’t provide any support to data quality issues.

Data quality can be specified by means of rules. Rules are formal statements that accurate data should conform to. Data quality rules can be classified in two types: *rules on values* and *rules on structure*. Rules on values aim at evaluating the accuracy of values stored in data fields, whereas rules on structure assess groups of interconnected records regarding their conformance to expected structural patterns. For instance, a rule that states that an accurate value for *customer\_age* should not be greater than 100 is a rule on values; similarly, a rule that states that *customer\_age* must be filled if attribute *customer\_type* is equal to “Person” is also a rule on values. On the contrary, a data quality rule that checks the records in the *OrderItem* table against the total number of items registered in the *Order* table is a structural rule; and a rule that checks that every Rock’n’Roll group has a drummer and an element that sings is a rule on structure as well.

Due to space restrictions, this paper’s scope is metadata support for rules on values. The level of fulfilment of quality rules by actual data elements and the associated cleaning activity are also addressed. Data quality rules on structure are postponed for a future paper. The problem of

duplicate detection and merging, which involves both value and structure rules, is left out of this paper’s scope as well.

The proposed metamodel is presented in a pure conceptual perspective, as no advice is provided regarding any relational or physical setup.

Most of the class diagrams included in this paper possess some connection points to CWM. A differentiation is made in order to distinguish CWM’s metaclasses from data quality ones: the former are marked with a small shortcut icon on the lower-left corner, as provided by Sybase’s PowerDesigner modelling tool.

Most of the classes in the diagram possess very few or no attributes. All the attributes are presented in this paper (except the object identifier attribute, which is implicit). On the contrary, CWM’s classes’ attributes are not shown, due to space restrictions. Please refer to OMG (2003) for further details.

The paper is structured as follows: section 2 presents some related work; section 3 presents the proposed metamodel, divided in the following parts: section 3.1 presents the artefacts required to characterize domains of accurate values, which may be shared by several table attributes; section 3.2 presents data quality issues that require the context of a table and an attribute to be specified; section 3.3 presents the modelling of simple data cleaning artefacts based on value replacement; section 3.4 presents general aspects that are common to every data quality rule, including some concepts (like shares and rule criticality) that are required to accurately produce data quality metrics; section 3.5 proposes a solution for data quality activity logging and user intervention in the cleaning process.

## 2 Related Work

The approaches to data quality are innumerable, both from research and industry fields. There are also plenty of contributions regarding metadata, both in data warehousing and general contexts. But, surprisingly, there are very few efforts aiming at cross-referencing data quality with metadata.

Olson (2003) and Kimball & Caserta (2004) are two important pieces of work from the industry. Olson does a good characterization of data quality problems and its contributions considerably inspired the work behind current paper. However, it doesn’t formally express its classification within a metadata framework. Kimball & Caserta (2004) references the data quality viewpoint in Olson (2003) and presents a rather minimal approach to data quality within an ETL context. Kimball & Caserta refers that data quality must be dealt by ETL components called *screens*, which are intended to identify and filter out records with data quality problems, registering everything in a metadata framework. This metadata framework has a general structure that doesn’t provide any predefined support for data quality classification. Kimball & Caserta (2004) Olson (2003)’s main focuses are not on data cleaning issues. Olson concerns are mainly data profiling and Kimball & Caserta aim at

finding data quality issues that will result in demands to improve source systems, not in demands for more cleansing within ETL processes.

Galhardas *et al.* (2000) is a seminal contribution in the data cleaning research arena, which had put forth an approach centred on a declarative language, based on SQL, which is enriched with special operators for record matching, clustering and merging. The proposed framework models the data cleaning process as a graph of transformations, with the ability of explicitly including the human interaction in the process. Galhardas *et al.* (2000) metadata approach is focused on documenting and controlling data cleaning programs and also includes a data lineage mechanism for back tracing purposes.

In spite of not being a metadata approach, Kim *et al.* (2003) presents a comprehensive classification of data quality problems and corresponding solutions, which is a good starting reference for any data quality metadata modelling effort.

Although not approaching from a metadata perspective as well, Lenz (1997) and Hurtado (2002) provide two contributions that pursue what we classify as rules on structure, with the purpose of evaluating the summarizability of values in OLAP and statistical database contexts. Both can be seen as possible foundational references for a prosecuting complement of current paper's metamodel.

### 3 A Metamodel for Data Quality

#### 3.1 Domain characterization

The set of accurate values an attribute may assume is defined by means of a domain specification. A domain specification is stored as an object of class *Domain*, which aggregates several data specification details, like restrictions about maximum and minimum values, restrictions about the textual structure of values, enumerations, etc.

Class *Domain* has two reflexive associations in order to support domain definition by means of both *inheritance* and *composition*. In the case of *inheritance*, a domain may be modelled as a refinement of other domain, meaning that all rules specified for the base domain are also valid for the new domain, which also defines its own data characterization rules. In this case, a sub domain is not necessarily a subset of the base domain, because additional specifications may widen, as well as narrow, the universe of accurate values. In the case of *composition*, a domain may be defined as the union of all values that belong to that domain's inner domains. A value belongs to a composite domain if it fits within at least one of the domains that compose that domain.

A name must be provided to a domain if it is intended to be reused by several attributes.

##### 3.1.1 Enumerations, ranges and length restrictions

If an attribute is constrained to assume values within a limited set, its domain metadata representation must be

linked to an object of CWM's *Enumeration* metaclass, as seen in Figure 1. An association class named *Enumeration Rule* is introduced, in spite of not having any attribute or association, because it needs to be superclassed by a top level metaclass, as will be shown in section 3.4. Several other empty association classes are declared in this metamodel, for this very same reason.

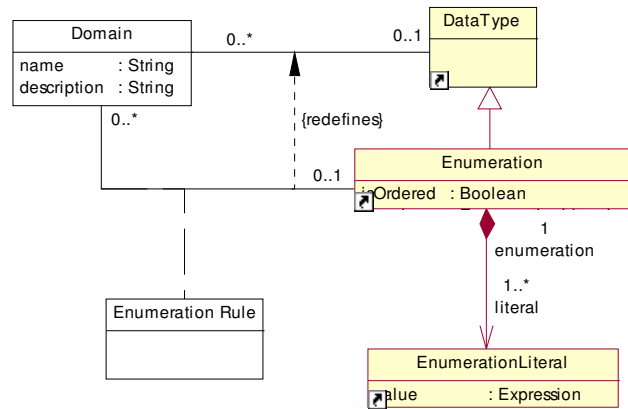


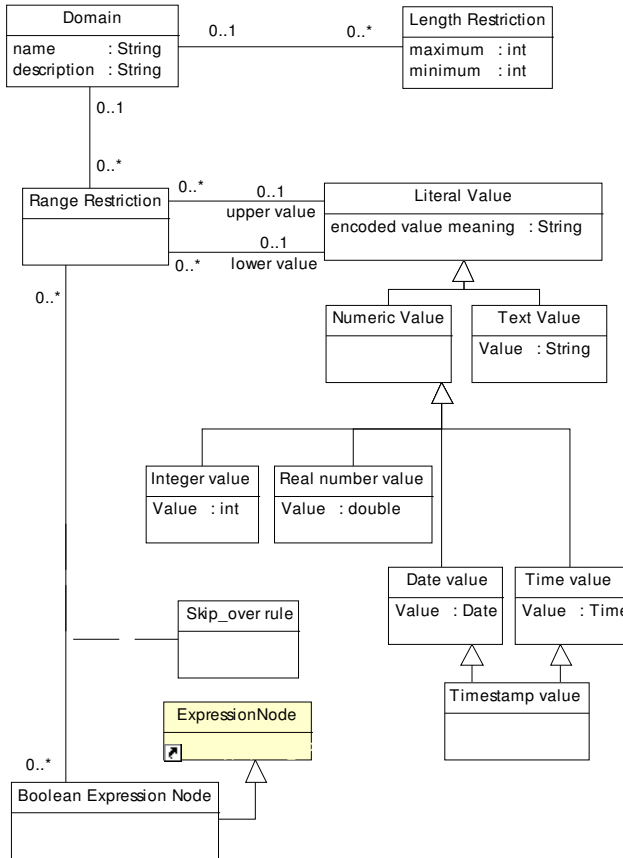
Figure 1: Enumerations.

A range restriction allows specifying that values must be within certain established minimum and maximum values. In our metamodel, metaclass *Range restriction* is associated to *Literal value*, whose subclasses allow setting up ranges of numbers, dates, timestamps and textual values (from 'A' to 'F', for instance). Refer to Figure 2. The metamodel allows that more than one range is associated to a domain, which may be valid as long as those ranges don't overlap. As an application example, consider an academic information context where values in a database column dedicated to exam dates should not be outside the established exam schedule. If this schedule encompasses two periods of time, then more than one range is needed to model this column's domain.

A useful capability inherent to ranges, as already mentioned in Olson (2003), is the ability to specify conditions that define skip-over sub-ranges, for the purpose of identifying values which are not valid besides of being within a valid range. For instance, in the former academic example it would be necessary to identify that, in spite of the specified date ranges, weekend days should not be allowed. These kind of cases, which are much easier to specify by means of a condition than by indication of explicit values, are supported by association *Skip-over rule* (Figure 2).

Length restrictions are also common in the definition of data. Usually, length restrictions are seen as an issue regarding text domains. They are specified as part of an attribute's data type and are used to establish data storage requirements as well as maximum allowed length for input data. However, very often additional length restrictions are needed to fully define accurate data, for at least the following reasons: (1) frequently, accurate values adhere to shorter textual lengths than the one specified for the corresponding data type, since this one must be the maximum for all values conceivable for the involved table column; by associating a length restriction to an applicability condition (refer to section 3.4.1), one is

able to tell expected lengths for specific subsets of a column's values; (2) sometimes the specification of a minimum length is required, while data types only provide maximum lengths; (3) length restrictions also apply to numeric domains as well; for instance, Portuguese telephone numbers are 9 digits wide; if a telephone number has 8 digits, then we can be sure the typist missed a keystroke. By means of metaclass *Length Restriction* our metamodel provides for the specification of more than one length restriction for a domain, both because different lengths may apply to different applicability conditions and because sometimes more than a length restriction may apply simultaneously.



**Figure 2: Length and range restrictions.**

Most of these domain considerations are already supported by traditional DDL languages. They are also included in a data quality metamodel because they express issues that, typically, are not fully identified at information system design time, leading for the need to distinguish those data rules that should be verified by data quality software from those that are originally embodied in the information system's functionality.

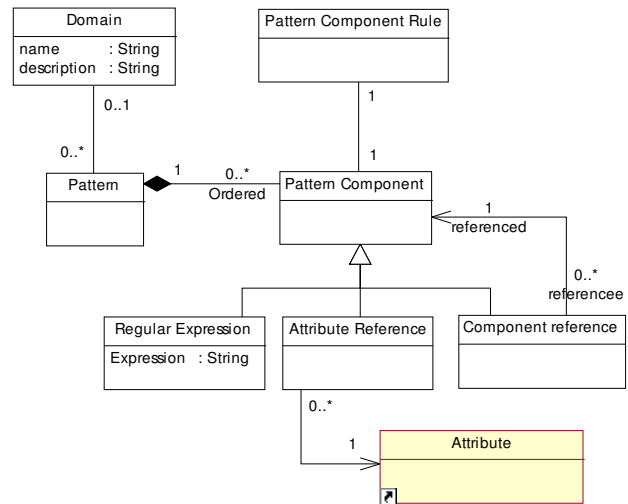
### 3.1.2 Textual patterns

When a field is expected to follow a lexical and/or a syntactic structure, identifiable as one or more textual pattern this fact is modelled with an association between classes *Domain* and *Pattern* (Figure 3), which allows linking several patterns to a domain. For instance, the domain of car plates is a patterned domain that exemplifies the possibility of several patterns, as car

plates differ from country to country and even also within a single country.

Tools and languages that support handling of syntactic constructs usually use regular expressions and/or (E)BNF to specify valid patterns. In our metamodel, regular expressions were adopted, so they can be used whenever they are adequate. However, regular expressions do not provide the necessary expressiveness to fully model text patterns in databases. Specifically, they do not provide an appropriate way of specifying a pattern that includes references to certain database attributes, i.e., a pattern that includes a slice whose contents are expected to match with one of the values existing in a certain column of a certain table. For instance, in a database of a multinational company, customer's tax numbers are prefixed with their country code. The identifiable pattern would be "<country-code> <number>". While the second component of the pattern can be represented as "[0-9]+", the first component cannot be specified as a regular expression, because it depends on the contents of a table (assuming the existence of a table with countries). Additionally, the limitations of regular expressions are further emphasized by the need for supporting cross-references between patterns, which is required for data cleaning and conforming purposes, as will be mentioned in section 3.3.

The metadata structure that covers common requirements is presented in Figure 3. A text pattern is modelled as an ordered sequence of components and each component may be specified in one of three ways: (1) as a regular expression, (2) as a reference to a database attribute or (3) as a reference to another pattern component.

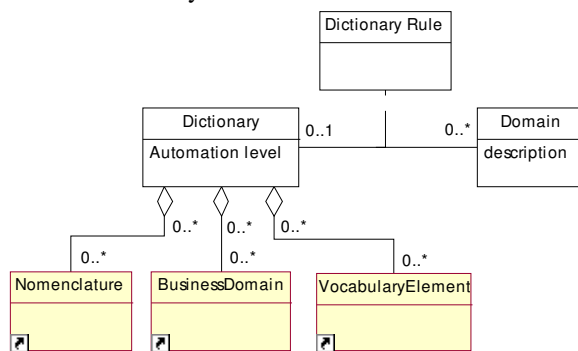


**Figure 3: Text patterns and their components.**

### 3.1.3 Dictionaries

Frequently, looking at words and phrases within attribute values is appropriate to infer data accuracy. Matching those with dictionaries of words, phrases and concepts may help in finding mistyping, outdated values, and other data quality problems. CWM already provides an infrastructure for terminology (OMG 2003, cp. 14) that can be adopted for this purpose. CWM's *Business Nomenclature* package includes classes for grouping

terms (words and phrases) in glossaries, grouping concepts in taxonomies, and cross relating both these classifications. The advised approach for data quality metadata is to aggregate *Nomenclature* objects, *BusinessDomain* objects, and *VocabularyElement* objects in *dictionaries*, in order to identify words and phrases that are “natural” to each domain, as presented in Figure 4. If a domain is linked to a nomenclature that happens to be taxonomy (in CWM, *Taxonomy* is subclass of *Nomenclature*), then all concepts within that taxonomy and all terms representing those concepts may be considered proper to that domain. The full set of words and phrases within *Business Nomenclature* package that are linked, directly or indirectly, to a domain is called this domain’s dictionary.



**Figure 4: Identifying accurate words and phrases for domains.**

A domain’s dictionary may be used to spell checking words within attributes using that domain. Slicing an attribute value in words and matching them (individually or combined in phrases) with the dictionary of that attribute’s domain allows detecting potentially wrong words. Automatically correcting these words replacing it by the nearest word in the dictionary can be a viable data cleaning operation, depending on both the domain and the dictionary. For instance, if the domain is meant to represent all possible personal (full) names, automatic correcting by dictionary is a risky data cleaning strategy. Due to the diversity of personal names in real world, chances are that a significant percentage of automatic corrections of names are incorrect, even if the dictionary includes a large quantity of names. On the contrary, correcting a database column that gets values from a domain of major European cities is reasonably safe. Therefore, in order to tell data cleaning software whether automatic correcting is safe or not, a *level of automation* must be assigned to every dictionary.

### 3.2 Attribute properties

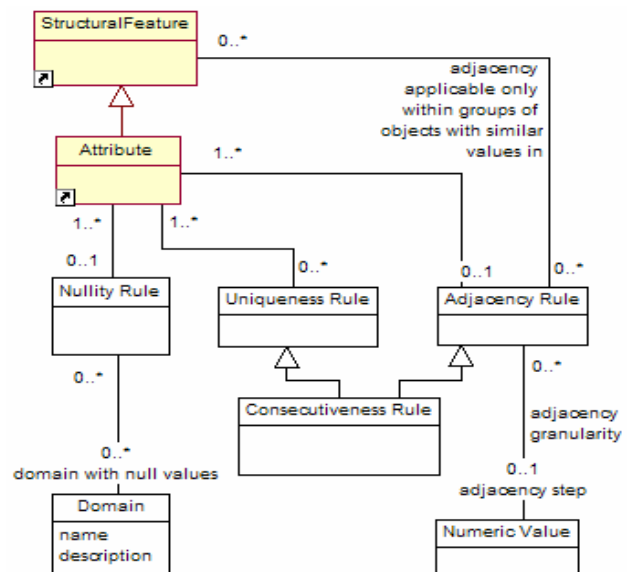
Figure 5 shows the class diagram of relevant metadata directly applicable to attributes (and not to domains, because it represents aspects that need the context of a table and attribute to be specified).

#### 3.2.1 Nullity

The absence of data is the more obvious manifestation of low data quality. When absent data prevents the user from getting a specific piece of information out of the database,

this is a clear symptom of low data quality. In a relational system, a column is defined as *not null* if its values are mandatory for the usefulness of data in that table’s records. This is typically specified by means of a DDL (Data Definition Language) during system development time. However, for the majority of cases, nullity prevention is not fully specified and there is a multitude of possible reasons for that. E.g., sometimes requirements for non nullity only rise up in later phases of system implementation, after the deployment of it. In such cases, the enforcement of a *not null* constraint is commonly unfeasible, because there are plenty of rows without a value in the column(s) and no way to infer the exact values that are missing. In other cases, nullity of certain columns is only a problem for certain business applications and, therefore, it should not be prevented by DDL. If it were, it would restrain data rows from being inserted and being used by other business processes.

Whenever null values contribute to the *completeness* dimension of data quality, the ability to specify that a field should not be null and the ability to recognize a null value are of great importance. Those are the aspects that should be stored as metadata, by means of class *Nullity Rule*. If nullity is only critical to certain application scenarios, this can be specified as presented in section 3.4.3.



**Figure 5: Attribute data quality specifications.**

For the sake of effectively detect null values, it must be taken into account that relational *Null* value is not the only possibility users have to leave a field in blank.

In non-relational systems, mainly in file and mainframe systems, null values are usually represented physically as empty strings, blank strings or zero values. Furthermore, users sometimes adopt their own conventions to say “no value”. This is prone to happen if the system forces that a value is provided when a value is not available and also happens when users want to say why there is no value. For instance, “Not provided by the customer”, “Unknown”, “Not applicable” and “Dec 31, 9999” are typical null values. Therefore, metadata should provide a way to identify what values should be considered

equivalent to *Null*. A many-to-many association allows the identification of more than one domain of null values, if necessary, while the hierarchical nature of domains allows the identification of null synonyms by means of inheritance. Note that, by associating class *Nullity Rule* to class *Domain* instead of to *Enumeration* (which could be assumed to be the more obvious option) gives greater flexibility in the definition of null values, because a domain can be defined implicitly rather than explicitly – for instance, if applicable, it can be said that all values less than 3 characters length are considered null values.

Note also the multiplicity of *many* of the association on the *Attribute* class' end. This has a specific purpose: if two or more attributes share a single nullity rule, it means that at least one of those attributes must have a value. As an example consider a table of bank account transactions with two columns for storing the amount, one for debits and the other for credits.

### 3.2.2 Uniqueness

This has a similar meaning to relational *UNIQUE*. CWM's metaclass *UniqueKey* already embodies the necessary features that are required to identify (groups of) attributes whose values should not repeat. Hence, our metamodel merely introduces a new class, which inherits from *UniqueKey* without introducing any specificity. Data quality processes should not use *UniqueKey* directly both because uniqueness may not have been specified as a data integrity constraint and because every data quality rule metaclass must be superclassed by a general *Data Quality Rule* class (as will be seen in section 3.4).

### 3.2.3 Adjacency and consecutiveness

In some fields, values are expected to be contiguous. For instance, in a temperature registering system, *Time* field in new records is expected to assume values that are adjacent to values in previous records. In cases like this, the existence of gaps between values means that information is missing. So, adjacency between values is an accuracy rule.

Adjacency, however, needs the specification of an interval or of granularity in order to be validated. For instance, if a field is of the integer type, the data quality system can assume that two values are adjacent if they differ by 1. However, in some contexts, adjacency may require different value intervals. For instance, if temperature is registered at every 5 minutes, 00:05:00 is the granularity to be specified along a rule of adjacency. Expected adjacency between values in a certain attribute is modelled using class *Adjacency Rule* and a granularity is specified by means of association *Adjacency granularity*, which links to numeric values of any type (integer, real, etc.). If granularity is not specified explicitly, it can be inferred from the numeric precision of the data type associated to the field – for instance, if the data type is *Decimal* with a precision of 2 decimal digits, than two values differing by 0.01 are considered adjacent.

When an attribute is both unique and adjacent, it is classified as *consecutive*. This is the usual case of *serial*

(automatically incremented by the system) numeric fields.

### 3.3 Explicit cleaning and conforming rules

For inaccurate values data cleaning processing may take place in order to increase accuracy. Automatic data cleaning ranges from complex algorithmic operations to value mapping operations that simply state that a certain inaccurate value should be converted to a certain accurate value. Certain data cleaning situations can only be resolved by value mapping. For instance, if there are some customers whose country is registered as “German Democratic Republic” we want to correct them to the accurate value “Germany”. It is unlikely that such a case can be treated by a general processing algorithm. Most likely, the only way to establish a cleaning rule for all current and future occurrence of such case is storing as metadata that “German Democratic Republic” corresponds to “Germany” and that a conversion should take place. A metamodel support for explicit value mapping is presented in Figure 6.

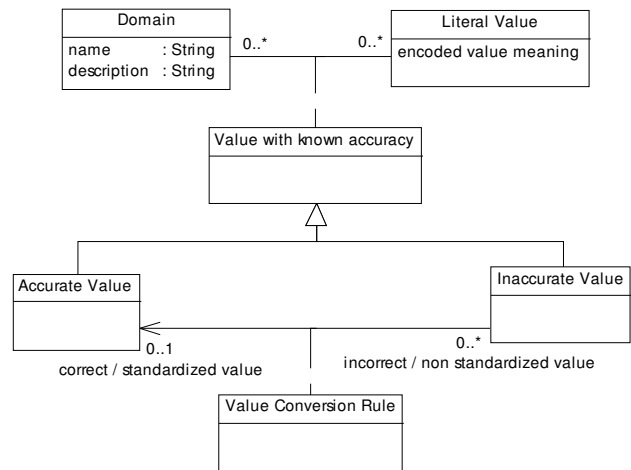


Figure 6: Conversion rules between values.

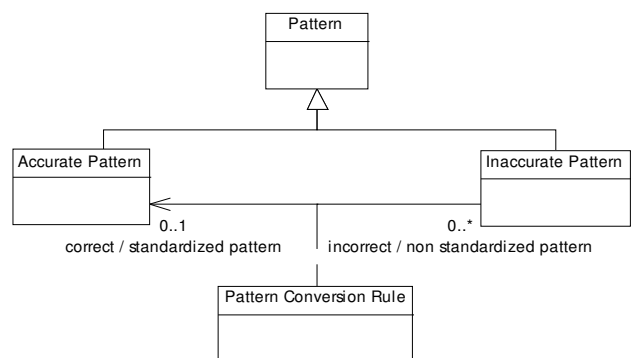


Figure 7: Conversion rules between patterns.

Some data mapping rules can be generalized as mapping rules between text patterns, which establish conversions from inaccurate patterns to accurate ones. A metadata structure similar to the previous one is required for these cases, as illustrated in Figure 7.

Pattern mapping requires that the source (inaccurate) pattern refers the target accurate pattern, including at

pattern component level. For example, consider a table column with customer tax numbers whose values should be prefixed with customer’s country codes. Consider also that several of those numbers are – inaccurately – suffixed instead. In such a case, one might (wrongly) think that the *good* pattern would be:

Component [1]: Reference-to (Country.code);

Component [2]: “[0-9]+”

...while the *bad* pattern would be:

Component [1]: “[0-9]+”;

Component [2]: Reference-to (Country.code);

However, saying that, in both cases, country codes must match with table *Country*’s contents is not enough. Country code actual values between these two patterns must match between them as well. Hence, in order to include this requirements, the specification of the bad pattern must be the following:

Component [1]: “[0-9]+”

Component [2]: Reference-to (*good pattern*. component [1]);

One might also consider that using the similarity between component specifications would be enough to infer that they must match. Note, however, that there may be, within the same pattern, two or more pattern components with the same specification but with different semantics, which would force to explicitly indicate which maps to which. A general solution would be the one presented, with component level mapping.

### 3.4 Data quality rules and general specifications

Some aspects are generically applicable to all types of data quality specifications, which leads to the need for a super class of all metaclasses representing data quality issues. This superclass is called *Data Quality Rule*, as seen in Figure 8, and possesses three immediate subclasses for the purpose of distinguishing those rules that are defined upon domains of values (subclass *DQ Rule for Domain*), those that are defined for individual attributes (subclass *DQ Rule for Attribute*), and those that do not follow a strictly predefined structured associated to a domain or an attribute, being otherwise defined by ad-hoc expressions. Every metaclass already presented in this paper and with the purpose of modelling a data quality rule inherits from one of the above-mentioned subclasses.

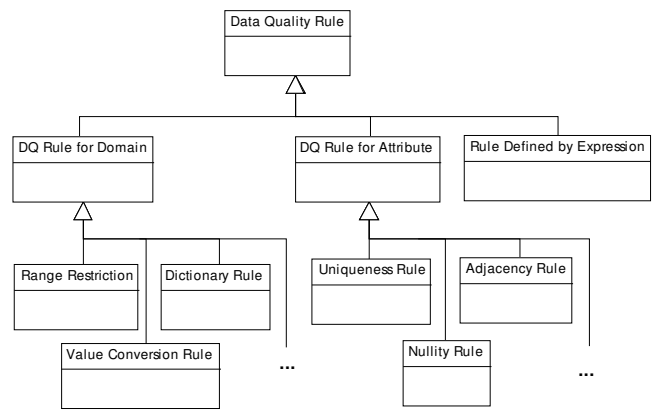


Figure 8: Data quality rules inheritance hierarchy.

#### 3.4.1 Rule definition by expressions

The most general way of specifying a data quality rule is by providing a Boolean expression, to which every data record must evaluate True. Our metamodel encompasses this type of specification for the sake of generality (Figure 9), as certain data quality rules can only be defined by means of expressions.

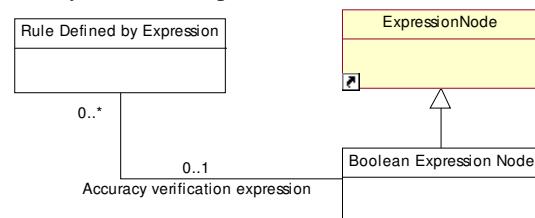


Figure 9: Specifying data quality rules using expressions.

Boolean Expression Node is a metaclass that models every expression that is either a comparison or a function with a Boolean return type. The self-referenced composite structure of CWM’s *Expressions* package allows building tree-like structures representing expressions, whose nodes may reference almost any type of metaobjects derived from CWM’s *ModelElement*. This allows to define expressions upon attributes as well as domains (which also derive from *ModelElement*).

#### 3.4.2 Restricting the applicability of a rule

Frequently, a data rule applies only to a subset of the records in a class/table. For instance, a rule stating a mandatory pattern for Portuguese car plates should be verified only for records representing Portuguese cars. A possible way of specifying this is by linking the pattern rule to a condition, such as *Car.country=“Portugal”*, that could filter out all not applicable records. This could be supported by the excerpt in Figure 10.

Note that those rules stored in subclasses of *DQ Rule for Attribute* (refer to previous section) belong to classes that already associate with metaclass *Attribute*. Hence, the associative class now presented should override that association, which was presented.

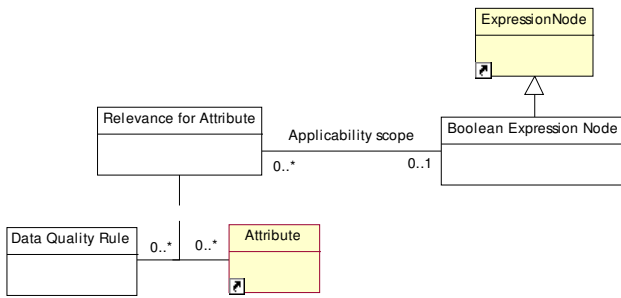


Figure 10: Defining the applicability scope of rules.

### 3.4.3 Relevance/criticality

Some data quality rules are more critical than others. The violation of certain data quality rules is more critical than the violation of others. Hence, in order to provide the appropriate contribution to data quality metrics, each rule should be quantified by a level of criticality or of relevance for a specific purpose.

The relevance of a rule usually depends on the attribute(s) it applies to and on the business process that such attribute(s) is(are) intended to be used. Support for the quantification of relevance is proposed as depicted in Figure 11.

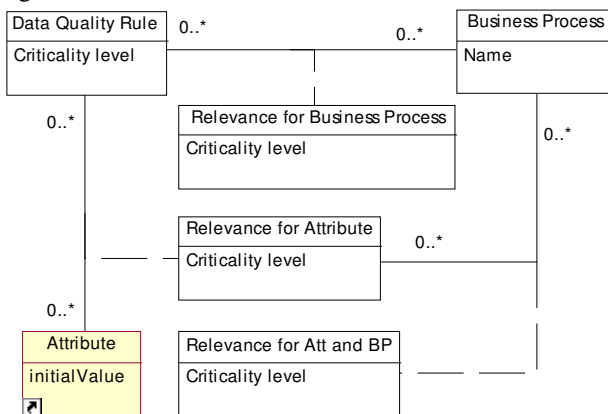


Figure 11: Specifying the criticality of rules.

Four *Criticality level* attributes are defined, which may assume numeric values within an established range (e.g., from 1 to 10, from least critical to most critical). Those four attributes should be used in a default-overriding basis, depending on the factors affecting the rule's relevance: (1) *Criticality level* in class *Data Quality Rule* is used to store the default relevance level of a rule; (2) *Criticality level* in class *Relevance for Attribute*, overrides previous value, when the rule relevance assumes specific values for some attributes it applies to; (3) *Criticality level* in class *Relevance for Business Process* overrides previous attribute, when relevance depends on the business process; (4) and, finally, *Criticality level* in class *Relevance for Att and BP* overrides all of the former, when relevance depends both on the attribute and the business process. Note that rules are allowed to have several levels of criticality: a rule may be moderately critical by itself (with no context; specification in class *Data Quality Rule*), highly critical when applied to attribute X (specification in class *Relevance for Attribute*), while no critical for business process Y

(specification in class *Relevance for Business Process*), except when applied to attribute Z and business process Y (specification in class *Relevance for Att and BP*). The level of criticality that applies depends on the context, i.e., on the perspective by which data quality is being assessed.

Rule's criticality should be used as a weight factor when determining the data quality level of a data source and its fit for use if it is intended to be used for a specific purpose (or business process). Note that, in order to get effectively correct accuracy indicators, it is necessary to have metadata that cross relates attributes with processes, in order to express which data is used by which processes. This is a concern that belongs to a wider scope than this metamodel's, since it is not specifically a data quality issue. But if metadata for such a wider scope is not in place, an association must be defined between *Attribute* and *Business Process*, in order to more accurately evaluate data quality on the perspective of specific business processes.

### 3.4.4 Specified vs. Inferred

Every data quality rule can either be provided by a data quality expert or inferred by a data profiling system. In the first case, it is classified in class *Specified DQ Rule*. In the second case, it is inserted in class *Inferred DQ Rule*. The specificities of both cases are shown in Figure 12.

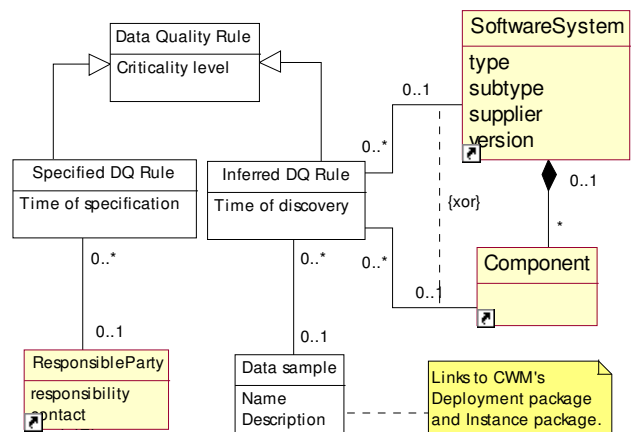


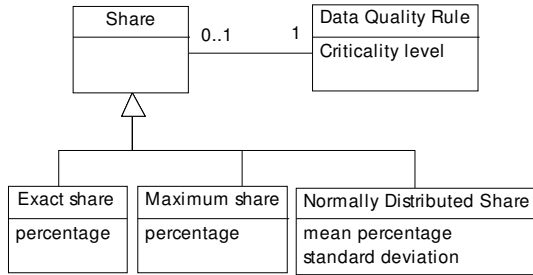
Figure 12: Specified vs. inferred rules.

Note that these metaclasses inherit from DQ Rule in an orthogonal way (different generalization branches), meaning that a rule may have been firstly inferred by the system and, in a later stage, explicitly validated by an expert.

### 3.4.5 Shares

Data quality rules are not strict data integrity rules. Some rules are not fulfilled by 100% of data records even when data is totally accurate. Even then, those rules are worth being declared as data quality indicators. For instance, temperature values collected hourly in an average Mediterranean city would range from -3 to 42 degrees Celsius, however sometimes values actually go beyond those boundaries. Reporting some occasional and rare values outside of that range as erroneous would be an

overreacting behaviour. On the contrary, widening the range that defines reasonable values for that case, in order to avoid warning messages, would allow for completely inaccurate situations not being noticed. In order to deal with these types of cases, metadata should allow specifying what percentage of records is expected to fulfil each accuracy rule. This leads to a share specification metamodel, as presented in Figure 13.



**Figure 13: Rules and shares.**

In the previous example, it would be reasonable to say that a rule establishing a range of [-3, 40] °C upon temperature values would expect to match at least 99.5% of records. This would mean that only a share of more than 0.5% of records should trigger data quality issues regarding that data.

Three ways of quantification are proposed regarding shares: (1) exact percentage of records, (2) maximum percentage and (3) normally distributed percentage. If not provided, an exact share of 100% should be assumed for a rule, meaning that, actually, it should apply to the whole data set.

Shares can also be used to specify data quality by means of histograms. Certain data quality problems can not be assessed in a per-record basis, but otherwise by means of a global assessment of data: matching a whole data set against an estimated histogram, a total deviation indicator may be obtained.

When associated with inferred DQ rules, shares may also be used for data profiling.

### 3.4.6 Specifying exceptions

In order to identify data records that are not compliant with certain rules, without being inaccurate, an association between metaclass *DQ Rule* and CWM's *DataObjectSet* is established. This will filter out data objects from future data profiling and cleaning operations related with the indicated rules.

Exceptional cases should be taken into account when evaluating shares (refer to section 3.4.5).

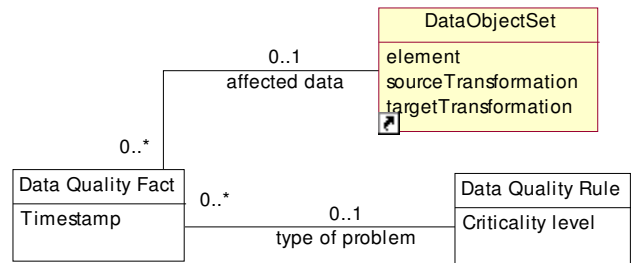
## 3.5 Registering DQ facts and cleaning activity

Additionally to providing support to define what is accurate data, through rule specification, a data quality metamodel should also provide support to recording running data profiling and cleaning activity.

After applying the defined rules to a data repository, data quality software will need to record data quality facts, i.e., what data quality problems were found and what data

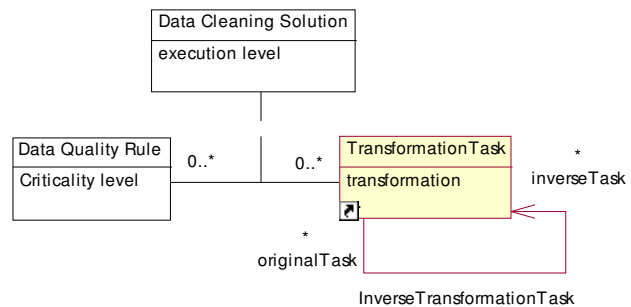
is affected. Moreover, if data cleaning activity is applicable, it is required that data transformation logging takes place, for data lineage (back tracking) purposes.

The metaclass provided to record data quality problems detected in data is *Data quality fact*. This metaclass has an association with *DQ Rule* that allows identifying the type of problem and an association with CWM's *DataObjectSet* that allows identifying the data conveying the problem (Figure 14). *DataObjectSet* is used for identifying groups of objects of *ModelElement*, another CWM's metaclass, which superclasses almost everything in CWM. This provides a flexibility that allows specifying the occurrence of data quality issues for every type of data elements: from individual object slots to complete tables (as needed to store an aggregated value with the percentage of records with null values in a certain column, for instance). *Data quality fact* is also the class that provides the support for storing the value of metrics, obtained by aggregation of individual data quality facts.



**Figure 14: Recording data quality issues.**

It is important to identify for every problem if there is a computational process that would improve the quality of the affected data and, if there is, what process(es) is(are) adequate. This information may be registered by means of an association defined between *Data Quality Rule* and CWM's *TransformationTask* (Figure 15). CWM's *TransformationTask* is used to represent data cleaning solutions because every data cleaning operation is a data transformation. But also due to the fact that, since *TransformationTask* is part of CWM's package to model ETL transformation workflows, data cleaning operations can inherently be integrated within such ETL transformation chains. *TransformationTask*'s reflexive association may be used to define reverse transformations for undoing data cleaning.



**Figure 15: Specifying data cleaning solutions to data quality problems.**

Among data cleaning operations, a differentiation must be done regarding the adequate level of automation and user participation. While some data cleaning processes have a nature that exhibits a high degree of confidence and can be performed right away by the system, other transformations might need human approval before being performed (personal name correcting is the canonical example of the later). This differentiation should be specified by means of attribute *execution level* in metaclass *Data Cleaning Solution*. This attribute may range, for instance, from 0 (low human approval dependency) to 10 (high human approval dependency) and could be matched against a system configurable parameter that indicates the level below which data cleaning operations are performed automatically.

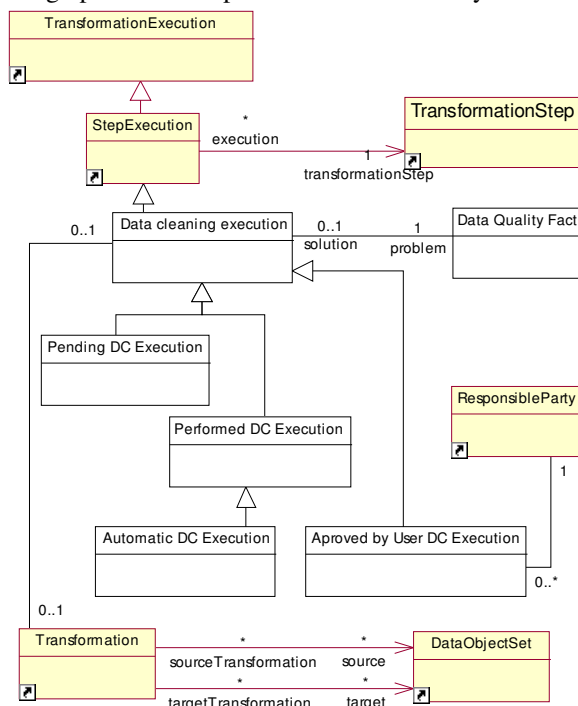


Figure 16: Logging data cleaning activity.

Recording individual and actual data cleaning operations can be accomplished subclassing from *StepExecution* (in CWM's *Operations* package). CWM's provides two metaclasses to record transformation execution: *ActivityExecution*, for coarse grained transformation activity, and *StepExecution*, for finer grained transformations. As data cleaning may take place within more general ETL activity, *Data Cleaning Execution* inherits from the later. Further subclassing from *Data Cleaning Execution* provides support for registering automatic data cleaning as well as human intervention, as seen in Figure 16. *Pending DC Execution* stores data cleaning transformations that haven't yet been executed, because they relate to *TransformationTasks* that depend on human approval (refer to Figure 15). Such data cleaning will only be performed upon approval, either immediately, triggering further ETL workflow and refreshing data warehouse contents on the fly, or waiting for the next scheduled run of the ETL process. This should be registered in a new subclass of *TransformationStep*.

Actual data values before and after the transformation should be registered as instance metadata, which can be reached through path *Transformation* → *DataObjectSet* → *ModelElement*.

#### 4 Conclusions and future work

In this paper we presented a conceptual model for data quality metadata.

From an applicational point of view, the presented metamodel has been already successfully tested against two actual databases (one with student academic data and another with logistics data) and have proven to be useful for the purpose of data profiling and data quality management. The ability to structurally store data quality rules and their actual fulfilment by data records allowed to produced summary data which provides indicators on data quality by table, repository, type of problem, etc. Regarding data cleaning, the metamodel provided direct support to simple text substitution rules, while more complex cleaning algorithms were only identified and had their activity logged, so far. Globally, the results achieved were rather satisfactory and encouraged to proceed with future work, suggesting that further efforts should be put both on expanding the data model and on the development of effective presentation and reporting solutions.

On going work aims at providing support to parameterization of complex cleaning algorithms, quality rules on structure, and duplicate detection and merging.

#### 5 References

Galhardas, H., Florescu, D., Shasha, D., & Simon, E. (2000). *AJAX: An extensible Data Cleaning Tool*. In 'Proceedings of the ACM International Conference on Management of Data (SIGMOD)', page 590.

Hurtado, C. A., & Mendelzon, A. O. (2002). *OLAP Dimension Constraints*. In 'Proceedings of the ACM Symposium on Principles of Database Systems (ACM PODS)', 169-179.

Kim, W., Choy, B.-J., Hong, E.-K., Kim, S.-K., & Lee, D. (2003), *A Taxonomy of Dirty Data*. In *Data Mining and Knowledge Discovery*, 7, 1, 81-99.

Kimball, R., & Caserta, J. (2004), *The Data Warehouse ETL Toolkit*. Wiley Publishing, Inc.

Lenz, H.-J., & Shoshani, I. (1997). *Summarizability in OLAP and Statistical Data Bases*. In 'Proceedings of the Ninth International Conference on Scientific and Statistical Database Management (SSDBM)', 132-143.

Pool, J., Chang, D., Tolbert, D., & Mellor, J. (2003), *Common Warehouse Metamodel Developer's Guide*. Wiley Publishing, Inc.

Olson, J. E. (2003), *Data Quality: The Accuracy Dimension*. Morgan Kaufmann.

OMG (2003), *Common Warehouse Metamodel (CWM) Specification*, Version 1.1, Object Management Group. <http://www.omg.org/technology/documents/formal/cwm.htm>. Accessed July 2006.