Support for quality metrics in metamodelling

Xavier Le Pallec
LIFL - Université Lille 1
Cité Scientifique, bâtiment M3
59655 Villeneuve d’Ascq Cedex, France
xavier.le-pallec@univ-lille1.fr

Sophie Dupuy-Chessa
LIG - Université Pierre Mendes France
38041 Grenoble Cedex 9, France
Sophie.Dupuy-Chessa@imag.fr

ABSTRACT
The maturity of Model Driven Engineering facilitates the development of domain specific languages. Their creation relies on the definition of metamodels, but also on their corresponding visual notations. One can wonder about the quality of any new language, which can result in inunderstandable diagrams with inappropriate notations. Then our goal is to provide indicators about the quality of notations thanks to metrics. In this paper, we present functions that are necessary to calculate these metrics in a metamodelling environment. Then we introduce how metrics are integrated in a modeling environment named ModX.

Categories and Subject Descriptors
H.4 [Information Systems Applications]: Miscellaneous; D.2.8 [Software Engineering]: Metrics—complexity measures, performance measures

Keywords
diagram quality, language quality, metrics, notations, metamodelling tool

1. INTRODUCTION
Thanks to the current outstanding technological improvements, accessing information anywhere, at anytime in a highly customizable way is becoming a reality. Unfortunately, the larger the field of possibilities becomes, the more complexity of design increases. Palen [28] has already confirmed this upward trend about design complexity for Human Computer Interaction ten years ago. Many scientific work has investigated Model Driven Engineering (MDE) this last decade in order to deal with this phenomena: 1) a model provides an efficient support for discussion, understanding a software architecture or components and 2) technological evolutions has generally less impacts on it. This has led to the definition of a lot of modelling languages for specific domains like interactive devices [5] or context of use [12].

Unfortunately, a significant risk about those new languages is to produce useless and unclear diagrams, or to use inappropriate metamodel and/or modelling notation as a basis of a software environment. In such context, the quality of visual modelling languages and their diagrams becomes an issue. So our goal is to help authors of visual modelling languages by providing them with evaluation mechanisms that may assist the evaluation of quality of their production. In this work, we focus on quantitative metrics, that is to say, mechanisms that can be automatically computed. Despite the fact that this approach is restrictive (because evaluation of quality also includes qualitative aspects), it presents results that are useful and usable as it has been shown in the domain of code metrics.

First, we define the functions that are required to calculate metrics about visual notations. Then we explain how metrics can be defined in our metamodelling tool, named ModX. In this way, a designer may use our predefined metrics or define new ones.

Before getting any further, we can draw a list of terms uses in the remainder of this paper.

- Abstract syntax (absStx) refers to the set of concepts and their relations of a modelling language. It is usually specified through a metamodel. When one describes a abstract syntax, we may also use the term "abstract elements" (absElm) which refers to the components of the abstract syntax.

- Concrete syntax (conStx) refers to a visual notation which is associated to an abstract syntax. The latter may be linked to several concrete syntaxes. We consider here that a concrete syntax may be decomposed into concrete sub-syntaxes: UML has twelve types of diagrams and so twelve concrete sub-syntaxes. It is just a practical choice of classification. Similarly to abstract syntax elements, we also call elements of a concrete syntax or "concrete elements" (conElm), all components of its definition (shapes, images, layouts...).

- Model (mdl) is a set of elements that conform to an abstract syntax. Models are called "instances" of metamodels because the widespread formats MOF and EMF adopt the object paradigm.

- Diagram (diag) is a visual representation of model (or a part of it) that conforms to a concrete syntax.

The remainder of the paper is organized as follows. In Section 2, we present existing scientific work related to model...
quality. We point out what approach we adopt and its related metrics. Section 3 describes ModX and its extensions to define metrics about models and metamodels. Section 4 reports conclusions and presents directions for future work.

2. CONTEXT

We first investigate existing work concerning quality of visual modelling languages and diagrams. We use them as a basis to propose an approach which is focused on quantitative metrics. Those ones rely on elementary functions which determine elements that have to be considered to compute automatic evaluations.

2.1 Related work

The quality of visual notations and diagrams is a scientific issue that has benefitted from the increasing interest in MDE. However quality frameworks have been defined long before the emergence of MDE:

- Hierarchical frameworks that propose a tree-view of quality. The root concept is the global quality which is further decomposed into different aspects which can be decomposed themselves. Each leaf is generally associated to a metric. One of the most famous framework related to quality is the ISO9126 standard [ISO 2001].

- Formal frameworks that propose methodology to construct quality metrics through mathematical properties that metrics have to respect. They aim to propose means to improve validity of new metrics while they do not provide any definition of quality.

- Causality-oriented frameworks [34, 20] which, unlike hierarchical ones, focus on the mutual influence between properties and intend to measure related correlations.

- Semiotic frameworks which are based on sign theory and have been initiated by Lindland [19]. They have been designed to evaluate any kind of diagrams. In [19], quality is detailed as syntactic, semantic and pragmatic. Syntactic quality evaluates the model according to the modelling language structure. Semantic quality deals with the correspondence between a model and its associated domain (or experts knowledge about it). Finally, pragmatic quality is about how the audience interprets the model. Even if [25] has demonstrated the benefits of Lindland’s framework, the pragmatic quality has been extended with organizational and technical qualities [17] and MDE considerations have been integrated in the global framework [32] [23].

One of the main interests of those frameworks lies in their definition of quality and how they structure the way to deal with it. However, we think that their usability is reduced when they do not provide automatic evaluation mechanisms. For this reason, we promote an approach similar to hierarchical frameworks which lead to metrics-based evaluations.

This metrics-based approach has already been applied to metamodels by [30] to calculate the theoretical conceptual complexity. Here we will propose to define metrics related to visual notations. Of course this approach can be combined with complementary proposals based on users’ feedback.

2.2 Approach

If our approach does not consider the whole complexity related to quality, using metrics has the benefit of providing useful and usable results to designers. In the same way of tools like JDepend which dynamically compute metrics during coding, our tool aims to allow designers to dynamically compute metrics about the quality of their abstract and concrete syntaxes and relative diagrams. In that way, designers may avoid errors that may be revealed far later.

Our (meta-)modelling environment ModX proposes a set of already implemented metrics, that are proposed to designers of languages and diagrams. One of these metrics deals with informational density: it indicates if a concrete syntax contains too many elements. It is based on the magic number of Georges A. Miller (7±2) which should not be exceeded. This is also relevant for diagrams. For example, an UML sequence diagram that presents more than nine lifelines may be considered as too complex (this is confirmed by the UML practitioners’ feedback about diagram complexity [4]). So our tool can calculate metrics about diagrams quality as well as language quality.

Our approach also allows designers or experts in model quality to define themselves new metrics, like in [21]:

- Quality experts define metrics related to the general evaluation of abstract and concrete syntaxes and of any type of diagrams.

- Designers of a modelling language use the previous metrics to define efficient modelling languages and may further establish their own metrics for their diagrams but they may also modify existing ones.

Since Green’s work on Cognitive Dimensions [10], visual aspects in Software Modelling have been considered as an important issue. Unfortunately, to the best of our knowledge, there are still few work and much less tools about their related quality [15]. If our approach intends to be global, for the moment, we focus on graphical concrete syntaxes, i.e. visual notations. We assume that the quality of diagrams is composed of two parts: the quality of its underlying concrete syntax and the quality related to its components (ex: number of elements, layout). We are mainly concerned by the first part (concrete syntax). We use the Physics Of Notations (PON) has defined by Moody in [24] as a basis of our work. If there is still no tool with metrics compliant with PON, an increasing number of work adopt it in order to evaluate visual notations ([9, 3, 35]). In this perspective, we propose here a software basis to such kind of study: we draw a list of elementary functions that PON criteria require and that each metamodelling tool should provide in order to define novel metrics.

2.3 Elementary functions to evaluate quality of concrete syntax

As we previously mentioned, specifying quality indicators about language quality remains a scientific issue. However, it is possible to draw a first list of properties which form elementary data to calculate metrics.

Metrics about the abstract syntax require an access to construction properties of language elements. MOF or EMF provide reflexive interfaces which are enough in this perspective. Similar interfaces are also needed concerning concrete syntaxes and unfortunately, they are not yet defined. We
intend to propose this kind of interfaces by studying the PON criteria. For each of them, we indicate what elementary access functions it may require. Table 1 shows the correspondence between the PON criteria and the elementary functions. We use an informal formalism to specify functions: the star refers to a collection of values while braces permit to define objects whose properties are separated by comma.

Now let’s study the PON criteria.

**Semiotic Clarity.** It is a bijective relation between the set of abstract elements and the set of concrete elements. To avoid redundancy, overload, excess or deficit of symbols, it is recommended to associate one and only one concrete element to each abstract one. This concrete element should not be linked to several abstract elements. So we need a function that permits to get the concrete element(s) for a given abstract one (\( \text{concreteElement (absStx, conStx, absElm)} = \text{conElm}* \)). We also need a function which returns the list of concrete elements for a given abstract syntax (\( \text{concreteElements (conStx)} = \text{conElm}^* \)).

**Perceptual Discriminability.** It refers to the level of visual discriminability between two concrete elements. The higher the level is, the quicker the perception of diagrams will be and the lower cognitive effort will be. Visual distance is the main function of this criteria. However, there is still no well-established associated formula. Coming from cartography area, visual distance refers to several visual variables [1]: location (x,y), size, value (clear versus dark), grain (scale of texture pattern), color, orientation and shape. Visual distance between two elements is related to the number of visual variables on which both elements are different. Access to these variables is highly required when dealing with metrics about perceptual discriminability.

\[
\begin{align*}
\text{location (conElm)} & = \{x, y\} \\
\text{size (conElm)} & = \{\text{width, height}\} \\
\text{value (conElm)} & = \text{brightness} \\
\text{grain (conElm)} & = \{\text{texturePattern, scale}\} \\
\text{color (conElm)} & = \{r, g, b\}
\end{align*}
\]

**Semantic transparency.** When a person reads a diagram, she/he has to associate a meaning to each kind of visual representation. For author(s) of the concrete syntax, this meaning corresponds to its original semantic. Unfortunately, an inadequate visual representation may lead non-expert readers to associate a wrong meaning. Semantic transparency refers to the distance between the perceived meaning by reader(s) for a particular visual representation and the meaning as it has been defined by language author(s). It is a criteria which is difficult to automatically evaluate because it relies on many parameters: reader profile, business context, related practices about visual symbols in corresponding business area... Evaluation needs a comparative study from different corpus rather than a formula that one has just to apply. So we do not detect here a required function.

**Complexity Management.** Visual representation of complex systems is a cross-disciplinary issue in science. We face a similar issue when it comes to represent complex diagrams, where complex means many elements and relations. Managing this complexity is necessary and implies that concrete syntaxes has dedicated mechanisms for that. Graphical inclusion and diagrams partitioning are examples of such mechanisms. So an interesting elementary function for this criterion should return the associated mechanism for a given container-contained element relation because such type of relation is the most likely to be used for complexity management.

\[
\text{complexityMechanism (absStx, conStx, container-contained element relation)} = \text{graphicalMechanism*}
\]

**Cognitive Integration.** When a reader browses diagrams thanks to complexity management mechanisms, she/he needs visual artifacts which help her/him to not forget for
which reason she/he came to this diagram or in which context the diagram takes place. In other words, it is important to integrate the part of the model, which is currently read, into the mental model of the reader. For example, a good practice in the Web is to display the path of the current page. A possible elementary function to evaluate this criteria may return visual elements for a given type of a complexity management mechanism.

\[
\text{contextRepresentation} \left( \text{absStx}, \text{conStx}, \text{container}-\text{containedElement relation} \right) = \text{conElm}^* 
\]

**Visual Expressiveness.** The more a concrete syntax exploits efficiently visual variables, the more it is visually expressive. Visual expressiveness positively and significantly impacts on cognitive effectiveness. This criteria uses the same properties as perceptual discriminability.

**Dual coding.** It is not recommended to only use textual annotations to visually represent an abstract element. But reinforcing visual artifacts with such annotations is a good practice. To evaluate this criteria, we need to know what textual annotations are used for a given concrete element.

\[
\text{textualAnnotation} \left( \text{conElm} \right) = \text{textualAnnotation}^* 
\]

**Graphic Economy.** A concrete syntax should not have a visual vocabulary that is too large/​rich, that is to say with (too many values of) too many visual variables. Otherwise, mental activity dedicated to representation-meaning association will be too important and will negatively impact reading associated diagrams. This criteria implies for example to partition a model in different kinds of diagrams which are visually different (previously mentioned as concrete sub-syntaxes). This is not totally compliant with the semiotic clarity because it may imply symbol deficit. This criteria requires a function to get the number of associated concrete element(s) for a given abstract one. This function has already been mentioned above.

**Cognitive fit.** Abilities to fully exploit visual variables are relative to the support that is used to draw diagrams. For example, using complex images for concrete elements is not a good choice if related diagrams are planned to be drawn on paper. In a similar way, one also has to know if future readers (for a new modelling language) are not used to handling software engineering diagrams. If not, it is not recommended to use advanced mechanisms (like those previously mentioned) because they may require important mental efforts. A function to know what drawing support is planned to be preferred may be interesting because any language author has to indicate it and so to be aware of it. To deal with skills of future readers, we may refer to comparative studies as for semantic transparency. This is out of the scope here.

\[
\text{preferredDrawingSupport} \left( \text{absStx}, \text{conStx} \right) = \text{support} 
\]

Evaluating quality of concrete syntaxes with metrics implies to access to all listed primitives. We have implemented them in ModX in order to propose a tool able to calculate metrics about concrete syntaxes.

### 3. PROPOSITION

We have implemented the previous elementary functions in ModX [29]. These functions return values that are, of course, specific to ModX, that is to say, that they are limited to ModX’s abilities to represent concrete syntaxes. To report this implementation, we first describe ModX and the previously mentioned abilities. We further show how it integrates the principle of metrics. Finally, we illustrate how to use elementary metrics in order to write a metric related to visual distance.

#### 3.1 Presentation of ModX

ModX is a tool for modelling and metamodelling. It has been created in 2004 at Lille and is based of MOF v1.4 (Meta-Object Facility) [11]. Initiated in the context of the Kaleidoscope network of excellence [14], this editor aims to manipulate graphically any kind of models in Software Engineering (e-Learning [2], User Interface [31]). Figure 1 shows how ModX allows designers to create metamodels (abstract syntaxes), to associate them with visual notations (concrete syntaxes) and to edit derived instances i.e. models throw diagrams. There is no compilation or generation phase: syntaxes and models can be modified at any time and side effects are instantaneously visible. However, ModX’s inheritance rules remain simple: for example, if the type of an attribute is modified in a metamodel, all the associated properties in the models (present in ModX) will be reset. The interested reader could refer to ModX web site for more information (http://www.lifl.fr/modx).

Based on a simplified version of UML Use Cases, Figure 1 gives an overview of mechanisms which are proposed to designers to author their own concrete syntaxes. For a metamodel class, one can choose between one of the given shapes or an image, set the colors (background, border, text) and the size. One can also indicate if instances may embed other elements. For an association, one can choose between a line (and set its properties like color or style), an graphical inclusion relation (isNested or isNestedInText) or if linked elements will be on periphery (isJuxtaposed) like the dashed rectangle for UML template parameters. Concerning the edition of the concrete syntax, we have chosen simplicity (like TopCased [36]) rather than power (like Obeo Designer [26]); it is "easy" to define/parameterize a concrete syntax but possibilities are limited.

#### 3.2 Metrics in ModX

One of ModX’s main goal is to study how to allow a person, who has no skill in Computer Science, to design or to parameterize a software system through graphical diagrams. We have first proposed and implemented a method with which metamodel authors can define and associate modelling methodology to their metamodels (Incremental Modelling Process [18]). We plan now to inform them about the quality of their concrete syntaxes. So we propose:

- A programming interface (in Javascript) to access to abstract/concrete syntaxes defined in ModX and also to their models/diagrams;

- Two areas that are dedicated to the Javascript implementation of metrics related to abstract/concrete syntaxes and diagrams.

Two simple metrics samples are proposed for each previous area: the informational density for syntaxes metrics and the
Figure 1: Abstract and Concrete Syntaxes and Diagrams in ModX
connectivity coefficient for diagram metrics. If the algorithm is the same regarding to syntaxes and diagrams, the types of handled elements are different. The connectivity coefficient is related to the ratio between the number of relations and the number of elements.

Figure 2 illustrates metrics display. We only focus on display related to language because we are mainly interested by concrete syntaxes in this paper. The remainder of this section concerns these syntaxes and their properties. Metrics display is the execution result of scripts which are accessible at any time through areas we have previously mentioned.

Such scripts can start as follows:

```javascript
var conStx_classes = concreteSyntax._.classes;
var conStx_associations = concreteSyntax._.associations;
for (var idx=0; idx<conStx_classes.length; idx++) {
    ...}
addMetric("my indicator", itsValue, "one comment", "one description");
```

### 3.3 Elementary functions related to syntaxes

The metrics implemented in ModX rely on the elementary functions that we have previously presented in section 2.3. We now draw a list of correspondence between these functions and the properties and functions that are present in MoX to show how these elementary functions can be implemented here. This list is presented in Table 3.3 with one column for classes and one for associations. Each property which is present in the ModX columns is an object property that is related to the visual representation of classes or associations.

The concrete syntax can be reached thanks to variable `concreteSyntax`. It gives access to an array of the concrete elements corresponding to classes (`_classes`) and associations (`_associations`). ModX uses the properties of these elements to draw the diagrams. For example, the `shape` property of the concrete element associated to Use-Case class is set to `ELLIPSE`. This element also has a property `source` which refers to the UseClass class that has the properties `_name`, `_contents` (for attributes and references) or `_container`.

The visual variables (color, value, grain) are applied to the different parts of the concrete element. For example, the `shape` variable can refer to the geometric shape associated to a class or the shape that is used as a pattern to draw a border (or as a pattern of the border of a pattern). For concrete elements in ModX, there is no such recursion, and visual variables are limited to those that are proposed by ModX.

The listing 1 aims to determine if the visual distance is bigger enough between each pair of concrete elements. Note that the visual distance depends on its context of use, and for now, no scientific work has presented concrete empirical materials in order to define a formula for visual distance in Software Engineering. For this reason, the script is a simplified version used as a simple example. For each pair, the `computeAll` function will calculate the corresponding visual distance. If it is more than 1 - it means that there are at least two visual variables where elements have the same value - then the distance is considered as big enough. For classes, the distance may be calculated only if both elements use geometric shapes. If at least one element uses an image, then the distance is set to 2, meaning that the distance is also big enough. Otherwise, the function looks for differences concerning geometric shape, color, border, size, etc. For associations, the distance may be calculated only if both elements use lines. In this case, the functions looks for differences between the end line and the grain.

### 3.4 In other meta-case tools

Other meta-case tools (MCT) do not integrate metrics facilities for concrete syntaxes. But the amount of work to implement them vary according tools. First, a lot of meta-case tools only focus on textual notations or form-based ones like MPS [13], Spoofax [37], Whole platform[33] or Rascal[16]. So, we may assert that they are not candidate for concrete syntax metrics. Second, writing codes or rules to automatically calculate metrics requires an API to access to specifications of visual specifications. Such an API is rarely directly proposed. Eclipse-based tools like Obeo Designer [27] or Eugenia[8] do not propose it. However, as specification of concrete syntaxes are EMF models, it is possible to use Eclipse plug-ins like EOL[7] or QVT[6] to programmatically navigate within them. But this requires to install additional plug-ins and the connection between them and previous models is generally not automatic. Besides, some functions like visualRepresentation or textualAnnotation will be not present and will require implementing higher-level function. MCT like MetaEdit+[22] or xOWL[38] provide a direct access to concrete syntax specifications. Third, it is better to propose a complete integration of metrics: a place to write script of rules about metrics that are expected and also a place where result of metrics calculation will be displayed.
<table>
<thead>
<tr>
<th>Properties in ModX</th>
<th>Classes</th>
<th>Associations</th>
</tr>
</thead>
<tbody>
<tr>
<td>visualRepresentation</td>
<td>concreteSyntax</td>
<td>untreated</td>
</tr>
<tr>
<td>visualRepresentations</td>
<td>untreated</td>
<td>untreated</td>
</tr>
<tr>
<td>location</td>
<td>untreated</td>
<td>untreated</td>
</tr>
<tr>
<td>size</td>
<td>width (width)</td>
<td>untreated</td>
</tr>
<tr>
<td></td>
<td>height (height)</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>background (backgroundBrightness)</td>
<td>untreated</td>
</tr>
<tr>
<td></td>
<td>border (borderBrightness)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>text (textBrightness)</td>
<td></td>
</tr>
<tr>
<td>grain</td>
<td>border (borderType)</td>
<td>stroke (stroke)</td>
</tr>
<tr>
<td>color</td>
<td>background (backgroundColor)</td>
<td>untreated</td>
</tr>
<tr>
<td></td>
<td>border (borderColor)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>text (textColor)</td>
<td></td>
</tr>
<tr>
<td>orientation</td>
<td>untreated</td>
<td>untreated</td>
</tr>
<tr>
<td>shape</td>
<td>geometric shape (shape)</td>
<td>left/right ends</td>
</tr>
<tr>
<td></td>
<td>image (image)</td>
<td>(left/rightEndStyle)</td>
</tr>
<tr>
<td>complexityManagement</td>
<td>ability to contain</td>
<td>type of link (style)</td>
</tr>
<tr>
<td></td>
<td>(containerAbility)</td>
<td></td>
</tr>
<tr>
<td>contextRepresentation</td>
<td>untreated</td>
<td>untreated</td>
</tr>
<tr>
<td>textualAnnotation</td>
<td>display name (displayName)</td>
<td>display association name</td>
</tr>
<tr>
<td></td>
<td>display class name (displayType)</td>
<td>(displayType)</td>
</tr>
<tr>
<td>preferredDrawingSupport</td>
<td>untreated</td>
<td>untreated</td>
</tr>
</tbody>
</table>

Table 2: Elementary functions / ModX Properties

```javascript
visualDistance = {
  between2classes : function ( element1, element2) {
    var difference = 0;
    if (element1.formMode==element2.formMode &&
      element1.formMode==concreteSyntax.SHAPE_MODE) {
      difference+=((element1.shape!=element2.shape ? 1 : 0 ));
      difference+=((element1.backgroundColor!=element2.backgroundColor
        ? 1 : 0 ));
      difference+=((element1.borderType!=element2.borderType ? 1 : 0 ));
      if (element1.resizeMode==element2.resizeMode &&
          element1.resizeMode==concreteSyntax.NO_RESIZE) {
        difference+=((element1.width!=element2.width ||
          element1.height!=element2.height ? 1 : 0 ));
      } else difference = 2;
      return difference;
    },
  between2associations : function (element1, element2) {
    var difference=0;
    if (element1.style==element2.style && element1.style==
      concreteSyntax.LINK_MODE) {
      difference+=((element1.leftEndStyle!=element2.leftEndStyle ? 1
        : 0 ));
      difference+=((element1.rightEndStyle!=element2.rightEndStyle ?
        1 : 0 ));
      difference+=((element1.stroke!=element2.stroke ? 1 : 0 ));
    } else difference = 2;
    return difference;
  },
  computeForAll : function (concreteSyntax) {
    // uses function addMetric (title, value, comments)
    // to display pair of elements which are visually too close
  }
}
```

Listing 1: Simplified function for visual distance
place well-known and easily access, in order to guide the design of concrete syntaxes. At the best of our knowledge, we did not see such integration in studied MCT. However, tools based on meta-programming like MetaEdit+ provides a better support for such integration rather that compiled MCT (like xOWL).

4. CONCLUSION AND FUTURE WORKS

In this paper, we present a metric-based approach to assess quality of modelling languages. The metrics are integrated in (meta-)modelling environment in order to adopt and to test them. We illustrate our approach with some quality indicators about concrete syntax, because few work focus on it. The originality is twofold: first, we propose an extensible environment to automatically calculate metrics on modelling languages; second, the metrics address an innovative domain i.e. the quality of concrete syntaxes.

If we show the feasibility of our approach, it remains to prove or to confirm its relevance with empirical studies. First, we plan to determine a more realistic measure of visual distance between elements of visual notation through significant experiments. We aim to define a formula that permits to grade the perceptual discriminability of visual notations. We will be able then to start experiments with the authors of visual notations and diagrams to assess if the presence of metrics is really useful.

5. REFERENCES


