Texture-based Visualization of Metrics on Software Architectures

Heorhiy Byelas∗
Institute of Mathematics and Computer Science,
University of Groningen, the Netherlands

Alexandru Telea†
Institute of Mathematics and Computer Science,
University of Groningen, the Netherlands

Abstract

We present a method that combines textures, blending, and scattered-data interpolation to visualize several metrics defined on overlapping areas-of-interest on UML class diagrams. We aim to simplify the task of visually correlating the distribution and outlier values of a multivariate metric dataset with a system’s structure. We illustrate our method on a class diagram of a real-world system.

1 Introduction

Object-oriented software systems are typically visualized using UML diagrams. Software metrics, computed by static analysis, simulation, and dynamic analysis tools [Wust 2006], can be combined with diagrams, e.g. by mapping them to diagram element sizes and/or colors [Lanza and Marinescu 2006; Tilley et al. 1994; Termeer et al. 2005]. However, such methods do not work when element sizes are constrained (fixed) to fit a preferred layout, or when class colors are constrained because e.g. we like to draw class method names on a fixed hue background.

Showing metrics with icons or via the size/color of diagram elements can only show metrics on individual elements. Sometimes we need to visualize metrics of groups of elements, e.g. safety of all multithreaded components in a system, speed of all performance-critical components, and so on. Byelas et al. show such groups, called areas of interest (AOIs), by surrounding them with a smooth textured contour [Byelas and Telea 2006]. However, this technique cannot show the relationship between the elements’ metric values, e.g. safety, and the areas these elements belong to.

We present a new way to visualize software metrics defined on areas-of-interest, together with a system’s structure modeled as (UML) architecture diagrams. We use blending and texturing to render several metrics atop of a set of AOIs so that users can easily spot correlations of metrics and areas. We remove the need of drawing metric icons atop of the diagram elements, thereby freeing this space for displaying other information, such as textual annotations. Section 2 presents our new technique for rendering AOIs enhanced with metrics on UML diagrams, and an application thereof. Section 3 discusses the results and outlines future work directions.

2 Multivariate metric rendering

Consider a system diagram with n areas-of-interest A1 . . . An defined over its elements eij, j ∈ [1, |A|], where |A| denotes the number of elements in A. For each area A = Ai, we have a metric

\[ m_i : [1, |A|] \rightarrow \mathbb{R} \cup \text{None} \]

defined over its elements, where None denotes that ei does not have a value in A. We want to render all metric values for all areas on a single image, so that we can (a) discern the metrics of all elements; (b) can visually follow how a metric varies over an area; (c) spot elements having missing values; and (d) do not draw on the elements themselves. We use a two-step solution, as follows.

2.1 Rendering a single metric

In contrast to icon-based methods [Termeer et al. 2005], we draw metrics outside the diagram elements and inside the areas. Our key idea is to build an interpolation function \( M(x) \) of the values \( m_i \) over the extent of an area \( A \), equal to the individual metric values \( m_i \) for points \( x \) inside or close to the elements \( e_i \), and varying smoothly in-between. For this, we first compute the Delaunay triangulation of \( A \) and set \( M \) at each mesh vertex \( x \) to the metric value of the element \( e_i \) closest = argmin\( |x - e_i| \) \( |x - e_j| \) closest to point \( x \). \( M \) is a piecewise-constant interpolation of \( m_i \) over \( A \). Figure 1 a shows \( M \), using a blue-to-red colormap, modified show missing (None) values, as follows. We compute an interpolation \( \mathcal{P} \) of the dataset \( \{p_i = (m_i \neq \text{None})\} \) over \( A \), just as the interpolation \( M \) of \( m_i \). Next, we compute the hue-saturation-value color hsv(x) of any point \( x \in A \) as:

\[ h(x) = \text{rainbow}(M(x)) \]
\[ s(x) = \mathcal{P}(x) \]
\[ v(x) = 1 \]

Next, we smooth the signals \( M \) and \( \mathcal{P} \) using a simple Laplacian filter. Finally, we render the area’s border using a fuzzy gray band-texture, as in the original method [Byelas and Telea 2006]. All in all, the method delivers a smooth shading of the area which shows the values \( m_i \) close to their elements \( e_i \), smooths values in-between, and grays out colors close elements without values (Fig. 1 c).

2.2 Combining several metrics

We combine several metrics on possibly overlapping areas \( A \), by texturing each area, colored as described above, with a different opacity texture from a predefined set (see Fig. 2 right; black=opaque, white=transparent). Now, the area’s colors (showing metrics) blend with the area’s texture (showing the area’s identity), see Fig. 1 d.

Figure 2 shows an UML class diagram of a C++ graphics editor. We manually identified 7 high-level ‘subsystems’ (class groups) by looking at the actual code: the application’s entry point (main), log classes (logging), user interface code (GUI), saving/loading code (I/O), rendering code (OpenGL), and 3D model loading code (XML). For each of the diagram’s 50 classes, we computed the percentage of code within each of the 7 subsystems, e.g. a class in the OpenGL subsystem has a value of 0.5 if it contains 50% OpenGL code. We now render each subsystem and its code per-area; (a) combines with diagrams, and an application thereof. Sec-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

Figure 1: Construction of metrics visualization of an area with four elements

Figure 2: Large UML class diagram with 7 areas and over 50 classes. Metrics show the involvement of classes in several concerns

We discover several facts. There are few classes involved in 2, and none in 3, subsystems. Indeed, during the manual classification used, one tended to draw quite strong borders. Class B is strongly involved in both the main and core areas - in fact, it is the system’s entry-point. Class D was classified as strongly involved in I/O operations, and also part of the system’s core. However, its code is quite complex, so we were unable to assess how strongly it belongs to the core (missing metric value in A6). Hence, D was classified as strongly involved in A1, but involved in A6 up to an unclear level. Class E participates in both the core and GUI - it is, in fact, the main window. Its blue color in GUI shows that it contains very little GUI code actually, but a lot of control (core) code.

3 Conclusion and Future Work

We proposed a new method to render a set of metrics, with possible missing values, on areas-of-interest of UML class diagram elements, using a combination of texturing, blending, and spatial data interpolation, so that metrics-area and metric-metric correlations are easy to distinguish. Visually weaving different textures emphasizes different types of concerns (areas) and also shows several colors representing the different metrics. Careful tuning of the texture granularity, transparency, and pattern choice allows displaying up to three overlapping areas (and thus metrics) at any point on up to 10 areas of interest on diagrams of 20-50 classes.

We next plan to investigate the addition of shading to strengthen the visual emphasis of the structure, and study how interaction can make the examination of the metrics correlation more effective.

References


