Visual Maps for Data-Intensive Ecosystems

Efthymia Kontogiannopoulou, Petros Manousis, and Panos Vassiliadis

Univ. Ioannina, Dept. of Computer Science and Engineering, Ioannina, 45110, Hellas
{ekontogi,pmanousi,pvassil}@cs.uoi.gr

Abstract. Data-intensive ecosystems are conglomerations of one or more databases along with software applications that are built on top of them. This paper proposes a set of methods for providing visual maps of data-intensive ecosystems. We model the ecosystem as a graph, with modules (tables and queries embedded in the applications) as nodes and data provision relationships as edges. We cluster the modules of the ecosystem in order to further highlight their interdependencies and reduce visual clutter. We employ three alternative, novel, circular graph drawing methods for creating a visual map of the graph.

Keywords: Visualization, data-intensive ecosystems, clustered graphs.

1 Introduction

Developers of data-intensive ecosystems construct applications that rely on underlying databases for their proper operation, as they typically represent all the necessary information in a structured fashion in them. The symbiosis of applications and databases is not balanced, as the latter act as "dependency magnets" in these environments: databases do not depend upon other modules although being heavily depended upon, as database access is performed via queries specifically using the structure of the underlying database in their definition.

On top of having to deal with the problem of tight coupling between code and data, developers also have to address the disperse location of the code with which they work, in several parts of the code base. To quote [2] (the emphasis is ours): "Programmers spend between 60-90% of their time reading and navigating code and other data sources ... *Programmers form working sets of one or more fragments corresponding to places of interest* ... Perhaps as a result, *programmers may spend on average 35% of their time in IDEs actively navigating among working set fragments* ..., since they can only easily see one or two fragments at a time."

The aforementioned two observations (code-data dependency and contextualized focus in an area of interest) have a natural consequence: developers would greatly benefit from the possibility of jointly exploring database constructs and source code that are tightly related. E.g., in the development and maintenance of a software module, the developer is interested in a specific subset of the database tables and attributes, related to the module that is constructed, modified or studied. Similarly, when working or facing the alteration of the structure

E. Yu et al. (Eds.): ER 2014, LNCS 8824, pp. 385–392, 2014.

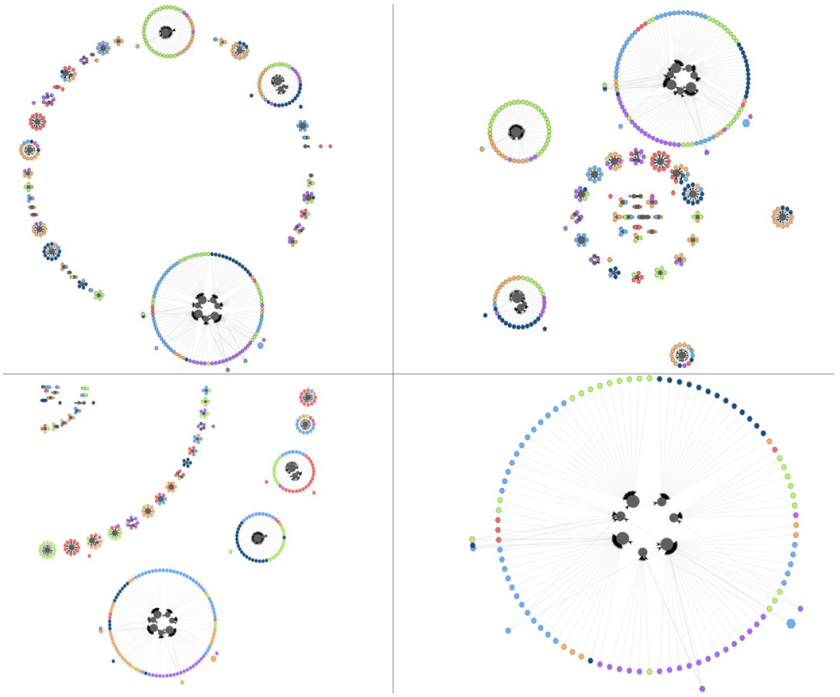

Fig. 1. Alternative visualizations for Drupal. Upper Left: Circular layout; Upper Right: Concentric circles; Lower Left: Concentric Arches. Lower Right: zoom in a cluster of Drupal.

of the database (e.g., attribute deletions or renaming, table additions, alteration of view definitions), the developer would appreciate a quick reference to the set of modules impacted by the change.

This *locality of interest* presents a clear call for the construction of a map of the system that allows developer to understand, communicate, design and maintain the code and its internal structure better. However, although (a) circular graph drawing methods have been developed for the representation general purpose graphs [11], [10], [6], and, (b) visual representations of the structure of code have been used for many decades [7], [4], [2], [3], the representation of data-intensive ecosystems has not been adequately addressed so far.

The research question that this paper addresses is the provision of a visual map of the ecosystem that highlights the correlation of the developed code to the underlying database in a way that supports the locality of interest in operations like program comprehension, impact analysis (for potential changes at the database layer), documentation etc.

Our method visualizes the ecosystem as a graph where all modules are modeled as nodes of the graph and the provision of data from a database module –e.g., a table– to a software module is denoted by an edge. To automatically

detect "regions" of the graph with dense interconnections (and to visualize them accordingly) we cluster the ecosystem's nodes. Then, we present three circular graph drawing methods for the visualization of the graph (see Fig. 1). Our first method places all clusters on a embedding "cluster" circle, our second method splits the space in layers of concentric circles and our last method employs concentric arcs. In all our methods, the internal visualization of each cluster involves the placement of relations, views and queries in concentric circles, in order to further exploit space and minimize edge crossings.

2 Graph Layout Methods for Data-Intensive Ecosystems

The fundamental modeling pillar upon which we base our approach is the *Architecture Graph* $G(V, E)$ of a data-intensive ecosystem. The Architecture Graph is a skeleton, in the form of graph, that traces the dependencies of the application code from the underlying database. In our previous research [9], we have employed a detailed representation of the queries and relations involved; in this paper, however, it is sufficient to use a summary of the architecture graph as a zoomed-out variant of the graph that comprises only of *modules* (relations, views and queries) as nodes and edges denoting data provision relationships between them. Formally, a *Graph Summary* is a directed acyclic graph $G(V, E)$ with V comprising the graph's module nodes and E comprising relationships between pairs of data providers and consumers.

In terms of visualization methods, *the main graph layout we use is a circular layout.* Circular layouts are beneficial due to a better highlight of node similarity, along with the possibility of minimizing the clutter that is produced by line intersections. We place clusters of objects in the periphery of an embedding circle or in the periphery of several concentric circles or arches. Each cluster will again be displayed in terms of a set of concentric circles, thus producing a simple, familiar and repetitive pattern.

Our method for visualizing the ecosystem is based on the principle of *clustered graph drawing* and uses the following steps:

1. Cluster the queries, views and relations of the ecosystem, into clusters of related modules. Formally, this means that we partition the set of graph nodes V into a set of disjoint subsets, i.e., its clusters, $C_1, C_2, \ldots, C_n$.
2. Perform some initial preprocessing of the clusters to obtain a first estimation of the required space for the visualization of the ecosystem.
3. Position the clusters on a two-dimensional canvas in a way that minimizes visual clutter and highlights relationships and differences.
4. For each cluster, decide the positions of its nodes and visualize it.

2.1 Clustering of Modules

In accordance with the need to highlight locality of interest and to accomplish a successful visualization, it is often required to reduce the amount of visible

elements being viewed by placing them in groups. This reduces visual clutter and improves user understanding of the graph as it applies the principle of proximity: similar nodes are placed next to each other. To this end, in our approach we use clustering to group objects with similar semantics in advance of graph drawing.

We have implemented an average-link agglomerative clustering algorithm [5] of the graph's nodes, which starts with each node being a cluster on its own and iteratively merges the most similar nodes in a new cluster until the node list is exhausted or a sued-defined similarity threshold is reached.

The distance function used in our method evaluates node similarity on the grounds of common neighbors. So, for nodes of the same type (e.g., two queries, or two tables), similarity is computed via the Jaccard formula, i.e., the fraction of the number of common neighbors over the size of the union of the neighbors of the two modules. When it comes to assessing the similarity of nodes of different types (like, e.g., a query and a relation), we must take into account whether there is an edge among them. If this is the case, the nominator is increased by 2, accounting for the two participants. Formally, the distance of two modules, i.e., nodes of the graph, M_i, M_j is expressed as:

$$dist(M_i, M_j) = 1 - \begin{cases} \dfrac{|neighbors_i \cap neighbors_j|}{|neighbors_i \cup neighbors_j|}, & if \quad \nexists\ Edge(i,j) \\ \dfrac{|neighbors_i \cap neighbors_j| + 2}{|neighbors_i \cup neighbors_j|}, & if\ \exists\ Edge(i,j) \end{cases} \tag{1}$$

2.2 Cluster Preprocessing

Our method requires the computation of the area that each cluster will possess in the final drawing. In our method, each cluster is constructed around three bands of concentric circles: an innermost circle for the relations, an intermediate band of circles for the views (which are stratified by definition, and can thus, be placed in strata) and the outermost band of circles for the queries that pertain to the cluster. The latter includes two circles: a circle of *relation-dedicated queries* (i.e., queries that hit a single relation) and an outer circle for the rest of the queries. This heuristic is due to the fact that in all the studied datasets, there was a vast majority of relation-dedicated queries; thus, the heuristic allows a clearer visualization of how queries access relations and views.

In order to obtain an estimation of the required space for the visualization of the ecosystem, we need to perform two computations. First, we need to determine the circles of the drawing and the nodes that they contain (this is obtained via a topological sort of the nodes and their assignment to strata, each of which is assigned to a circle), and second, we need to compute the radius for each of these circles (obtained via the formula $R_i = 3 * \log(nodes) + nodes$). Then, the outer of these circles gives us the space that this cluster needs in order to be displayed.

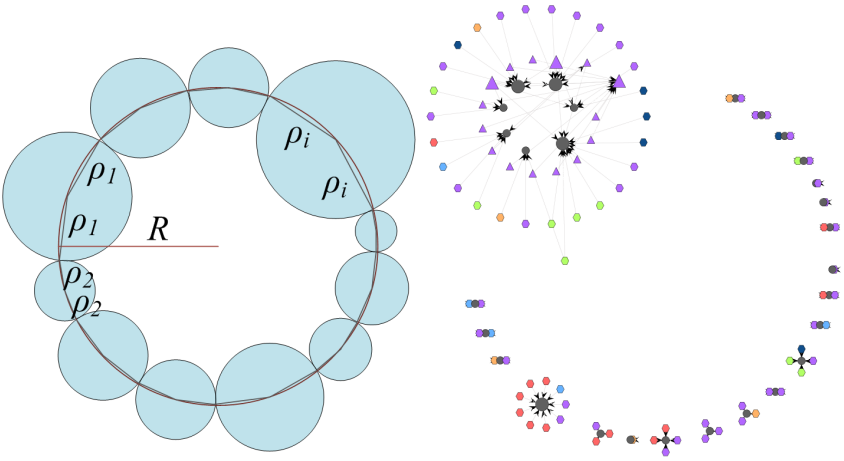

Fig. 2. Circular cluster placements (left) and the BioSQL ecosystem (right)

2.3 Layout of Cluster Circle(s)

We propose three alternative circular layouts for the deployment of the graph on a 2D canvas.

Circular Cluster Placement with Variable Angles. In this method, we use a single circle to place circular clusters on. As already mentioned, we have already calculated the radius r of each cluster. Given this input, we can also compute R, the radius of the embedding circle. We approximate the contour of the inscribed polygon of the circle, computed via the sum of twice the radius of the clusters by the perimeter of the embedding circle, which is equal to $2\pi * R$ (Fig. 2). We take special care that the layouts of the different clusters do not overlap; to this end, we introduce a white space factor w that enlarges the radius R of the cluster circle (typically, we use a fixed value of 1.8 for w). Then, $R = \sum_{i=0}^{|C|} \frac{2 * \varrho_i}{2\pi * w}$, where C is the set of clusters, and ϱ_i the radius of cluster i. As the arc around which each cluster will be placed is expanded, this leaves extra whitespace between the actually exploited parts of the clusters' arcs. Given the above inputs, we can calculate the angle ϕ that determines the sector of a given cluster, as well as its center coordinates (c_x, c_y) via the following equations:

$$\phi = 2 * \arccos\left(\frac{2 * R^2 - \varrho^2}{2 * R^2}\right),\ c_x = \cos\left(\frac{\phi}{2}\right) * R * w,\ c_y = \sin\left(\frac{\phi}{2}\right) * R * w \quad (2)$$

Concentric Cluster Placement. This method involves the placement of clusters to concentric circles. Each circle includes a different number of segments, each with a dedicated cluster. The proposed method obeys the following steps:

1. Sort clusters by ascending size in a list L^C
2. While there are clusters not placed in circles
   (a) Add a new circle and divide it in as many segments as $S = 2^k$, with k being the order of the circle (i.e., the first circle has 2^1 segments, the second 2^2 and so on)
   (b) Assign the next S fragments from the list L^C to the current circle and compute its radius according to this assignment
   (c) Add the circle to a list L of circles
3. Draw the circles from the most inward (i.e., from the circle with the least segments) to the outermost by following the list L.

Practically, the algorithm expands a set of concentric circles, split in fragments of powers of 2 (Fig. 3). As the order of the introduced circle increases, the number of fragments increases too ($S = 2^k$), with the exception of the outermost circle, where the segments are equal to the number of the remaining clusters. By assigning the clusters in an ascending order of size, we ensure that the small clusters will be placed on the inner circles, and we place bigger clusters on outer circles since bigger clusters occupy more space.

Radius Calculation. We need to guarantee that clusters do not overlap. This can be the result of two problems: (a) clusters of subsequent circles have radiuses big enough, so that they meet, or, (b) clusters on the same circle are big enough to intersect. To solve the first problem, we need to make sure that the radius of a circle is larger than the sum of (i) the radius of its previous circle, (ii) the radius of its larger cluster, and (iii) the radius of the larger cluster of the current circle. For the second problem, we compute R_i as the encompassing circle's periphery ($2 * \pi * R_i$) that can be approximated the sum of twice the radiuses of the circle's clusters. Then, to avoid the overlapping of clusters, we set the radius of the circle to be the maximum of the two values produced by the aforementioned solutions and we use an additional whitespace factor w to enlarge it slightly (typically, we use a fixed value of 1.2 for w).

$$R_i = w * \max \begin{cases} R_{i-1} + b_{i-1} + b_i \\ \\ \dfrac{1}{\pi} * \displaystyle\sum_{j=1}^{|C|} \varrho_j \end{cases} \tag{3}$$

where (a) b_α: is the rad of biggest cluster of circle α, and (b) ϱ_j: is the rad of cluster c_j which is part of C, where C is the set of clusters of circle i.

Clusters on Concentric Arches. It is possible to layout the clusters in a set of concentric arcs, instead of concentric circles (Fig. 3). This provides better space utilization, as the small clusters are placed upper left and there is less whitespace devoted to guard against cluster intersection. Overall, this method is a combination of the previous two methods. Specifically, (a) we deploy the clusters on concentric arches of size $\dfrac{\pi}{2}$, to obtain a more compact layout, and (b) we partition

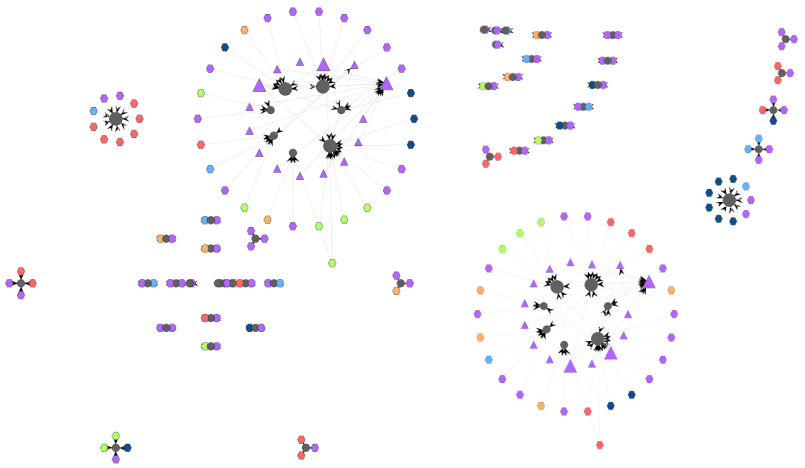

Fig. 3. Concentric cluster placement for BioSQL: circles (left), arcs (right)

each cluster in proportion to the cluster's size by applying the method expressed by equation (2).

2.4 Layout of Nodes inside a Cluster

The last part of the visualization process involves placing the internals of each cluster within the area designated to the cluster from previous computations. As already mentioned, each cluster is aligned in terms of several concentric circles: an innermost circle for relations, a set of intermediate circles for views and one or more circles for queries, as we previously stated at section 2.2. Now, since the radiuses of the circles have been computed, what remains to be resolved is the order of nodes on their corresponding circle. We order relations via a greedy algorithm that promotes the adjacency of similar relations (i.e., sharing the large amount of views and queries). Once relations have been laid out, we place the rest of the views and queries in their corresponding circle of the cluster via a traditional barycenter-based method [1] that places a node in an angle that equals the average value of the sum of the angles of the nodes it accesses.

3 To Probe Further

The long v. of our work [8] contains a full description of our method, along with its relationship to aesthetic and objective layout criteria and related experiments. Naturally, a vast area of research issues remains to be explored. First, alternative visualization methods with improved space utilization is a clear research area. Similarly, the application of the method to other types of data sets is also necessary. The relationship of graph metrics to source code properties potentially hosts interesting insights concerning code quality. Navigation guidelines

(e.g., via textual or color annotation, or an annotated summary of the clusters of the graphs) also provide an important research challenge.

Acknowledgments. Prof. I. Fudos and L. Palios have made useful comments to an earlier version of this paper. This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

References

1. Battista, G.D., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall (1999)
2. Bragdon, A., Reiss, S.P., Zeleznik, R.C., Karumuri, S., Cheung, W., Kaplan, J., Coleman, C., Adeputra, F., LaViola Jr., J.J.: Code bubbles: rethinking the user interface paradigm of integrated development environments. In: Kramer, J., Bishop, J., Devanbu, P.T., Uchitel, S. (eds.) ICSE (1), pp. 455–464. ACM (2010)
3. Caserta, P., Zendra, O.: Visualization of the static aspects of software: A survey. IEEE Trans. Vis. Comput. Graph. 17(7), 913–933 (2011)
4. DeLine, R., Venolia, G., Rowan, K.: Software development with code maps. ACM Queue 8(7), 10 (2010)
5. Dunham, M.H.: Data Mining: Introductory and Advanced Topics. Prentice-Hall (2002)
6. Halupczok, I., Schulz, A.: Pinning balloons with perfect angles and optimal area. J. Graph Algorithms Appl. 16(4), 847–870 (2012)
7. Johnson, B., Shneiderman, B.: Tree maps: A space-filling approach to the visualization of hierarchical information structures. In: IEEE Visualization, pp. 284–291 (1991)
8. Kontogiannopoulou, E.: Visualization of data-intensive information ecosystems via circular methods. Tech. rep., MT-2014-1, Univ. Ioannina, Dept. of Computer Science and Engineering (2014), `http://cs.uoi.gr/~pmanousi/publications/2014_ER/`
9. Manousis, P., Vassiliadis, P., Papastefanatos, G.: Automating the adaptation of evolving data-intensive ecosystems. In: Ng, W., Storey, V.C., Trujillo, J.C. (eds.) ER 2013. LNCS, vol. 8217, pp. 182–196. Springer, Heidelberg (2013)
10. Misue, K.: Drawing bipartite graphs as anchored maps. In: Misue, K., Sugiyama, K., Tanaka, J. (eds.) APVIS. CRPIT, vol. 60, pp. 169–177. Australian Computer Society (2006)
11. Six, J.M., Tollis, I.G.: A framework and algorithms for circular drawings of graphs. J. Discrete Algorithms 4(1), 25–50 (2006)