3D Hierarchical Edge Bundles to Visualize Relations in a Software City Metaphor

Pierre Caserta
INPL Nancy University
LORIA Laboratory
Nancy, France
pierre.caserta@loria.fr

Olivier Zendra and Damien Bodénès
INRIA Nancy Grand-Est
LORIA Laboratory
Nancy, France
{olivier.zendra, damien.bodenes}@inria.fr

***Abstract*—Software systems are often very complex because of their huge size and the tremendous number of interactions between their components. However, understanding relations between software elements is crucial to optimize the development and the maintenance process. A good way to ease this understanding of software relations is to use advanced visualization techniques to graphically see interactions between elements. Nevertheless representing those software relations is not an easy task and often leads to hard to understand clutter. We believe that combining both edge clustering techniques and real-world metaphors can help address this issue, producing easier-to-read visualizations that ease the cognitive process and thus significantly help understanding the underlying software. In this paper, we explain how we adapted the existing 2D Hierarchical Edge bundles technique to represent relations in a 3D space on top of city metaphors.**

***Index Terms*—Visualization of Software, Software Relations, Software Maintenance, Human Perception, Hierarchical Edge Bundles**

I. Introduction

Large software is complex and hard to fully understand, especially when, as is often the case, it is built by developers reusing existing parts. The exact software structure produced may be unclear, which can harm the development and maintenance processes. Studying relations between software elements, be they classes, packages, methods, etc. thus becomes crucial to improve this comprehension.

In addition, the relations in programs very quickly come in overwhelming numbers. Indeed, even in a simple Java program such as "Hello World", hundreds of classes are present and create many relations. For instance the inheritance relation, which gives insight about the software design, one of the fundamental concepts of the object-oriented paradigm, has hundreds of occurrences; the dynamic call relation, which identifies runtime calls of a particular element to others, occurs a hundred of thousands times. Visualizing these relations in an effective way can thus lead the developer to better understand how the software works.

Visualizing software relations has been a subject of many publications over the past years [1], be they based on 2D or 3D representations. Meaningfully representing software relations nonetheless remains challenging.

By considering software elements as nodes and relations as edges, *graphs* have all the required characteristics to represent relations [2]–[6]. Nevertheless, visualizing all relations in a piece of software may amount to visualize a huge graph with many different interconnections, especially for large software applications. However, being able to have a global overview of the system is very important. It makes it possible to decide which components need further investigation and to focus on a specific part of the software while keeping the overall visual context [7]. When no specific visualization technique is used, the resulting visual representation of a very big graph is likely to become confusing, with numerous edge congestions, overlapping and occlusions, which makes it almost impossible to investigate a single node or an individual edge.

Several common techniques exist to overcome graph visualization problems:

- substitute a group of visual objects by a single one [8].
- place nodes so as to minimize edge overlapping [5].
- bend edges and force them to join together according to a certain criterion [9].
- use 3D visualization to be able to navigate and find a view without occlusions [3].

In this paper, we combine the two techniques: a 3D visualization and a edge bending technique. We think that 3D visualization provides a global view of the nodes and that the edge bending technique improves its overall readability.

Software visualization [1] relies on the human perceptual system [10]. Indeed, using the latter effectively is important to reduce the user cognitive load [11]–[13]. A technique to ease the understanding of source code is to represent it through suitable abstractions and metaphors. This way, software visualization can provide a more tangible and intuitive view of the software [14], [15].

In this this paper we describe 3D Hierarchical Edge Bundles (3D-HEB), a 3D visualization technique based on graphs to represent relations within software, implementing solutions to overcome some of the issues with graphs, and relying on the software city real-world metaphor to improve understanding.

Our paper is organized as follows. In section II, we briefly reference and present research works that have inspired our relations visualization technique. In section III, we explain

978-1-4577-0823-7/11/$26.00@2011 IEEE

two different possible layouts of our software city metaphor, to show its adaptability. Then in section IV we talk about how we graphically draw edges using 3D hierarchical attraction points, which are the core of the technique. We discus edge bending and how hierarchical attraction points can influence edges path. In section V, we give some examples of the resulting visualization of relations, using our VITRAIL Visualizer tool. Finally, we conclude and present future work in section VI.

II. Related Work

Our technique to visualize relations builds upon two different existing and very interesting techniques. First, is it based on the city metaphor [16]–[18], that represents the software hierarchy in a convenient way. Second, it relies on the hierarchical edge bundles (HEB) technique [9], [19], that shows relations while avoiding the main clutter issue.

In this section we first give an overview of the city metaphor in sub-section II-A, then describe the hierarchical edge bundles technique in sub-section II-B.

A. The City Metaphor

Visualizing software with a *real-world metaphor* consists in representing software as a familiar context, using graphic conventions a user immediately understands [20]–[24]. According to Dos Santos et al. [25], this allows faster recognition of the global software structure and better and faster understanding of complex situations, preventing the most problematic aspect of 3D software visualization, i.e. disorientation. Visualization based on real-world metaphors relies on the human natural and intuitive understanding of the physical world, including spatial factors in perception and navigation.

For instance, the software city metaphor is divided into districts, districts into sub-districts that contain streets, buildings, etc. Usually, the software city geometry is laid out over a 2D plane [26]. Some works divide the 2D plane on many pieces and add a degree of elevation on each of them to express software metrics [27] or project metrics [28].

Even if the software city looks kind of a real city, it is strongly believed that real-world metaphors do not have to correspond exactly to reality and small discrepancies do not hinder understanding [29]. In fact, we consider that small discrepancies, especially simplification of the reality, can help focus on the important information of the visualization.

The city metaphor is largely studied and leads to many publications [27], [30]–[34]. Experiments in [35] show that using CodeCity (a visualization tool with a city metaphor) improves, in both correctness (+24%) and completion time (-12%), over the state-of-the-art exploration tools.

B. The Hierarchical Edge Bundles (HEB) Technique

Software has a hierarchical structure: in most object-oriented languages, for example, methods are defined in classes themselves included in packages. Therefore this hierarchy can easily be represented as a tree and visualized by one of the many Tree representations existing, like the Treemap [36], [37], the Balloon tree [3] and many others [7], [38].

Holten [9] introduces the *Hierarchical Edge Bundles* visualization technique to represent software relations as a 2D graph on top of a 2D hierarchy representation. Software elements are mapped as nodes of the graph and the relations as edges. The basic idea is to use a 2D geometric layout to characterize the software hierarchy and edges on top to visualize relation between elements of the hierarchy. But instead of drawing the relations directly from one element to another, the HEB technique very interestingly introduces imaginary points to bend and somewhat bundle together edges, in order to reduce clutter and improve readability. These points, called hierarchical attraction points (HAPs), are defined for each element of the software and placed according to the position in the software hierarchy in attempt to form a tree which mirrors the hierarchy of the layout. Nodes are then connected together with edges bent according to the hierarchical attraction points.

The interest of this technique resides in the fact that each relation is affected by the power of attraction of the hierarchical attraction points. As a result, edges are grouped together at each level of the hierarchy to form clusters of edges.

III. Software City Layouts

Before explaining in section IV how we coupled, in a 3D visualization, the Software City metaphor and the Hierarchical Edge Bundle techniques, we briefly explain the software city layouts we created and used.

Indeed, the layout of a software city can be defined in many different ways. In fact when the layout is based on a representation of the software hierarchy, any tree representation may be used as a layout for the software city. In our work, we created and implemented two layouts, in order to show that our 3D Hierarchical Edge Bundles (3D-HEB) technique was largely applicable, and not dependent on a specific layout. In this section, we present these two layouts, implemented in VITRAIL Visualizer, our software city tool.

The hierarchy described in figure 1 is used as an example for the explanations we give in this section on the layouts and in section IV on our 3D-HEB technique. The arrow between `class 2` and `class 4` means that there is a relation (e.g a call relation) between these two classes.

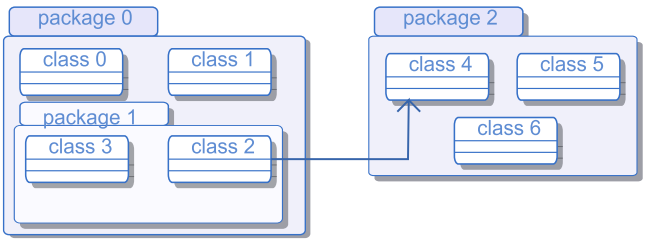


Fig. 1. Software hierarchy used in our examples

A. Nested Layout

Usually, a treemap splits the available space according to the hierarchy. To have something similar but with more control

on the graphical representation, we have adapted this layout to our needs, creating a new variant: the ”nested layout”.

First we travel into the software hierarchy from leaf to root and we compute the size of each node according to the size of its descendants. Then we graphically represent each element according to its size. Sub-packages are drawn on top of their parent package, and classes on top of the package they belong to. This way we have complete control on every visual parameter such as the size of classes, the space between geometric elements and so on.

Figure 2 represents the hierarchy of figure 1 according to our nested layout.

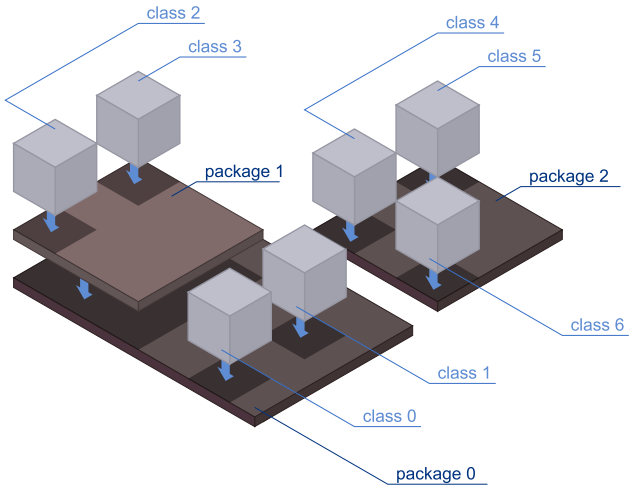


Fig. 2. Nested software city layout

B. Street Layout

We called our second software city layout the ”street layout” because it looks like long streets (packages) surrounded by buildings (classes). This layout was inspired by the one described in [28]. Package appears as streets. Sub-packages are placed perpendicularly to their parent package. Classes contained in packages are drawn as building surrounding the street they belong to.

Figure 3 represents the hierarchy of figure 1 drawn with our street layout.

IV. Drawing Relations

There are many publications on the software city metaphor, but few of them give an effective way of representing relations in the city [27], [34], [39]. The basic idea is to draw an edge from a building to another. The underlying problem resides in the fact the city metaphor is usually disposed on a 2D plane, which complicates an effective drawing of the edges. Representing straight edges on top of this layout produces much overlapping and many occlusions.

In our visualization technique, inspired by the (2D) Hierarchical Edge Bundles technique, we place 3D Hierarchical Attraction Points (3D-HAPs) which affect edge paths across

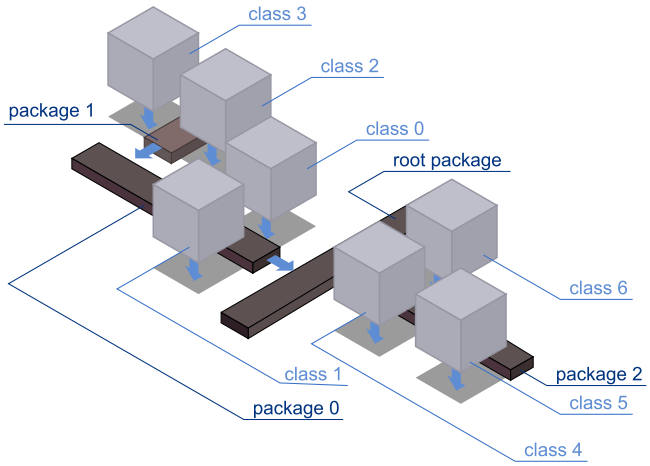


Fig. 3. Street software city layout

the city visualization. This way, edges are grouped together, resulting in a more understandable visualization of relations.

We explain in section IV-A how we place the 3D-HAPs to drive edges. In section IV-B we detail how we choose which 3D-HAPs form our path from one software element to another. In section IV-C we discuss how the power of attraction of the 3D-HAPs can influence edge representation. Finally, in section IV-D we describe how we use color to represent direction and quantification of relations. Note that it makes no semantic difference (besides aesthetic) whether edges are curved or lines with angles. We will show both representations in section V-B.

A. 3D Placement of Hierarchical Attraction Points

Our technique is derived from the (2D) Hierarchical Edge Bundles technique [9]. We put virtual attraction points in space according to the software hierarchy, but we do this in 3D. These 3D Hierarchical Attraction Points (3D-HAPs) are used to drive edges from one point to another across the visualization.

Figure 4 schematises our technique to place 3D-HAPs. There are seven 3D-HAPs at the `class level` , each one for its related class. There is one 3D-HAP at `level 1` , corresponding to `package 1` . Finally two 3D-HAPs at `level 0` correspond to the root packages: `package 0` and `package 2` . We can see that the 3D-HAPs mirror the layout in an opposite way.

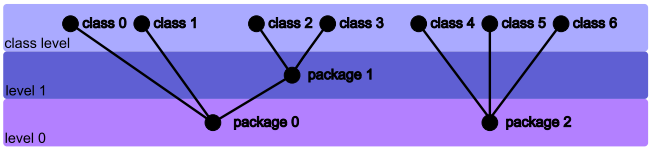


Fig. 4. Level decomposition

Figure 5 shows how 3D-HAPs are arranged in space. They are placed at the center of each geometric object at different levels of elevation corresponding to the depth in the software

hierarchy. The deeper into the hierarchy tree, the closer the 3D-HAP is to its geometric element.

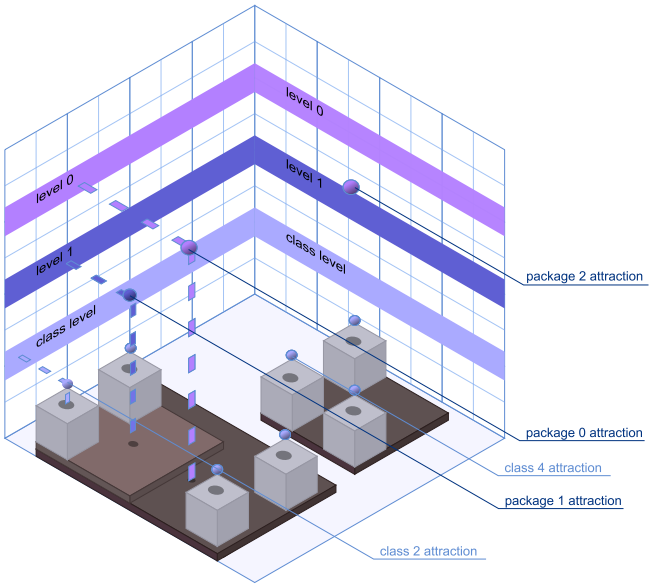


Fig. 5. 3D Placement of Hierarchical Attraction Points

B. Relation-Relevant Hierarchical Attraction Points

This section describes how we choose the 3D-HAPs which compose the path of each relation.

With our 3D-HEB technique, a relation between two classes is represented by an edge passing by several 3D-HAPs. Our algorithm finds the shortest path, in terms of number of 3D-HAPs traveled, between two elements according to the hierarchy.

The implementation of the algorithm tracing an edge between two geometric objects using the 3D-HAPs is as follows.

Let's define:

- $path(X, Y)$: the set of 3D-HAPs to travel from origin X to destination Y[1].
- $higher3DHAP(s)$: the higher 3D-HAP of set s.
- $level(P)$: the level number of 3D-HAP P.
- $Root$: the top-level 3D-HAP corresponding to the root of the tree hierarchy.

We compute the set of 3D-HAPs from the origin class 3D-HAP to the destination class 3D-HAP by doing the symmetric difference between two sets:

- the set of 3D-HAPs from origin O to the $Root$. We call this set the origin 3D-HAP set.
- the set of 3D-HAPs from destination D to the $Root$. We call this set the destination 3D-HAP set.

The symmetric difference between those two sets eliminate the non-needed 3D-HAP (those are common to both sets).

$path(O, D) = path(O, Root) \Delta\ path(D, Root)$

[1]Naming points X and Y "origin" and "destination" is purely a geometrical convention: it does not imply that the relation between the software elements corresponding to X and Y is oriented.

With our 3D-HEB technique, it is necessary to differentiate the path of the origin and the one of the destination. In consequence the $path(O, D)$ can be divided in two distinct sets of 3D-HAPs: one related to the origin package (this set is called $pathOrigin$) and one related to the destination package (this set is called $pathDestination$).

We thus have :

$pathOrigin = path(O, D) \backslash path(D, Root)$

$pathDestination = path(O, D) \backslash path(O, Root)$

We will see later in section IV-D that we color edges according to these two sets.

C. Attraction Power

As in the original (2D) HEB technique [9], our (3D) HAPs attract edges according to variable β, which defines the attraction power of each 3D-HAP. In this section we explain the mathematics implemented to bend edges according to the β factor.

Figure 6 shows a schematic view of a cluster of edges. It is composed of two relations: one from `class 2` to `class 4` , and one from `class 0` to `class 6` . Edges are attracted by 3D-HAPs to travel from the origin class to the destination class.

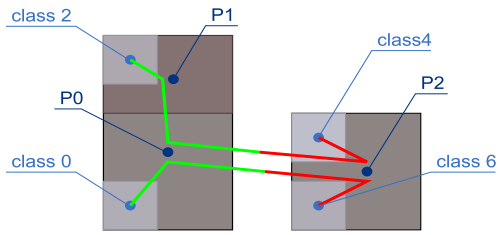


Fig. 6. Schematic view of a cluster of edges.

Figures 7 and 8 show a 2D representation of the 3D-HAPs needed to draw the relation between `class 2` and `class 5` . Points P_0, P_1, P_2 are the 3D-HAPs representing respectively `package 0, package 1, package 2` .

In figure 7, we perform a projection of the 3D-HAPs on line OD. This projection is used to compute the power of attraction of the 3D-HAPs on edges.

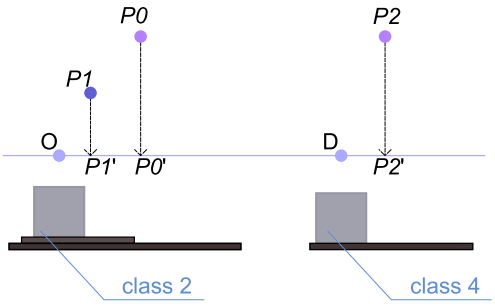


Fig. 7. Projection of HAPs on the line OD

Figure 8 shows three edge representations with three different β. The path with $\beta = 0$ is not attracted by the 3D-HAPs,

so it is drawn as a straight edge from the 3D-HAP O (at class level) to the 3D-HAP D (at class level). On the opposite, the path with $\beta = 1$ forces the edges to be completely attracted by the 3D-HAPs (P_1,P_0,P_2).

The mathematics to compute edges position are the following (we take P_0 in figure 8 as an example):

$P'_0 = O + \frac{\vec{OD} * (\vec{OP_0} \cdot \vec{OD})}{||\vec{OD}||^2}$

$P''_0 = P_0 * \beta + (1 - \beta)(P'_0 * \beta)$

with:

O : the 3D-HAP of the origin class (at class level)
D : the 3D-HAP of the destination class (at class level)
β : the attraction power $\beta \in [0, 1]$ of 3D-HAPs
P_n : the 3D-HAPs
P'_n : the projections of each P_n on the OD line
P''_n : the 3D-HAPs interpolated according to β.

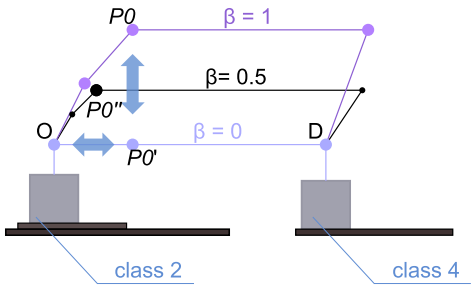


Fig. 8. Interpolation of P_n according to the β factor.

In section IV, we showed that 3D-HAPs are placed at different levels of elevation and that the lowest one is the `class level` . This level is a bit different from the others because it makes the connection between the 3D-HAPs and the center of the top of the geometric objects representing the software (buildings for classes in the cite metaphor). We do not apply β on these 3D-HAPs at class level, in order to connect vertically to the boxes representing the software elements. To be independent of different elevations introduced by different layouts, we place each 3D-HAP related to a building at the same level (the class level). Note that if $\beta = 0$, 3D-HAPs have no influence on edges, so the latter are flattened on the class level.

Each 3D-HAP implements the same attraction algorithm, so if many geometric edges leave from exactly the same coordinates they would take exactly the same path in space (it would appear as a single geometrical line) until the paths diverge. To solve this problem, we decided to scatter the edges around the first 3D-HAP (the one at class level). This way, no edge has exactly the same coordinates as another one around the class level 3D-HAP.

D. Quantification and Direction

The quantification, or magnitude, of a relation is an important information for the software developer.

Indeed, an edge represents a relation between two software elements, but this relation can have many occurrences. For example, let assume that the relation between `class 2` and `class 4` in figure 1, is a call relation and that `class 2` calls `class 4` 50 times. In this case, we must represent these 50 occurrences to be able to see and understand the magnitude of the relation.

Drawing bigger edges according to the number of occurrences was not effective because edges overlap to much to clearly see their size.

After some experiments, we decided color was appropriate to visualize relation quantification in our 3D-HEB technique. Four different variations of green are used to color the edge. We compute the statistic quartiles of the data collection to divide the relations into four sets of the same size. Then we map the four greens with the number of occurrences, red meaning no occurrence, as shown in figure 9. Each extremity of the relation is colored according to its weight (ie number of occurrences) in the relation.

In case the relation is bi-directional, both extremities are colored in green according to their respective weight. The difference in color shows which extremity corresponds to the larger number of occurrences, hence indicating a dominant direction in the relation. In case a relation is unidirectional, the green color of the source indicates its weight, while the destination remains red. This corresponds to an analogy with the traffic light: green represents the origin (”go”), and red represents the destination (”stop”).

From a drawing point of view, we explained in section IV-B that the path is composed of two sub-sets of 3D-HAPs. One is related to the origin package and the other to the destination package. The change of color is performed in the middle of the edge connecting the $higher3DHAP(pathOrigin)$ and the $higher3DHAP(pathDestination)$, to provide a clear vision of edge quantification.

Fig. 9. Division of the data collection into 4 sets

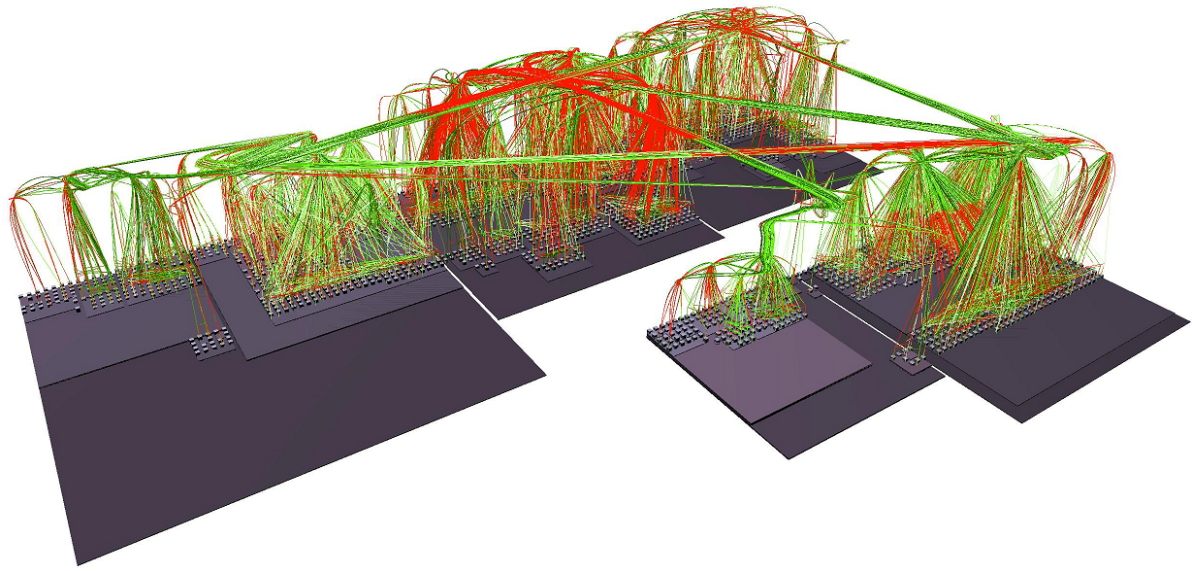
V. Results

The huge number of interactions between software elements in software makes software relations difficult to represent in a comprehensive way. In this section we present real case visualization examples of our 3D-HEB technique on top a software city metaphor, obtained with our VITRAIL Visualizer tool.

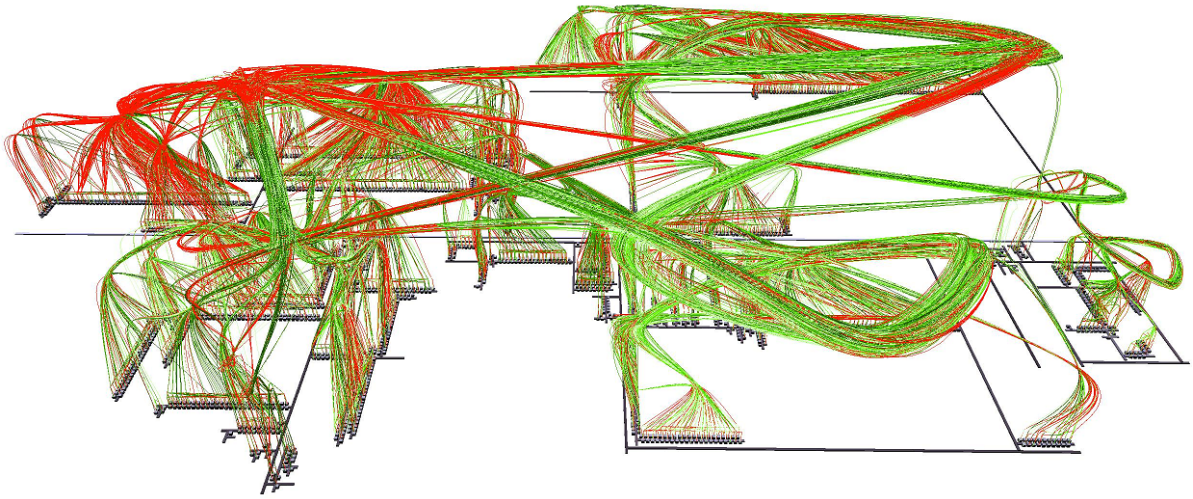
In section V-A we show that our 3D-HEB technique is layout-independent. In section V-B we discuss the different parameters which can be adjusted to modify the visualization of relations and improve readability and understanding.

A. Layout-Independency

One specificity of the original 2D HEB technique of Holten is that it can be displayed on top of any layout. In section V-A1 we present views of our 3D-HEB technique on top of



(a) Relations on top of the nested layout of the software city metaphor



(b) Relations on top of the street layout of the software city metaphor

Fig. 10. Visualization of dynamic call relations on an execution of JEdit, Java JRE classes included. 2710 classes, 10870 edges representing 4 632 680 calls. Strong attraction power ($\beta = 0.9$) resulting in bundled edges.

our nested layout then in section V-A2 on top of our street layout.

1) Relations On Top of the Nested Layout: Figure 10(a), which represents edges with $\beta = 0.9$, shows how edges are attracted by the 3D-HAPs. Edges elevate and join together to form clusters. This technique gives some volume to the representation and shows the "ground" layout better when relations are represented. It also makes it possible to see how much elements have relations with each others, and their occurrences and directions. For instance, the package represented at the top center of figure 10(a) is the `java.lang` package which contains core Java classes and the one on the left is the core package of JEdit. The fact that calls coming from the core package of JEdit to `java.lang` are going in a single direction can also be seen in the figure.

Our 3D-HEB over software city metaphor visualization technique thus provides a big picture of relations in the software. To be able to perform more detailed analysis of the visualization, its user will have to navigate and select specific edges for further inspection[2].

2) Relations On Top Of The Street Metaphor: Figure 10(b) shows how relations are displayed over the street metaphor explained in section III-B. With this layout, the 3D-HAPs are scattered more largely. As a result, edges travel more distance from a point to another and, as an advantage, provide more space between clusters of edges.

[2]The details of the navigation capabilities of our tool are omitted, since they fall beyond the scope of this paper.

B. Impact of visualization parameters

This section briefly mentions how several parameters of the visualization can impact its readability.

Figure 11 shows, with $\beta = 0.8$, the dynamic call relations in an execution of the JEdit software, considering only the calls within JEdit classes (this excludes the JRE). This figure is used as a basis for comparison with the others figures in this section.

Section V-B1 shows how the β factor can affect edge bending. Section V-B2 explains how some 3D-HAPs can be removed to simplify edge paths. Finally, section V-B3 discusses curved edges and straight edges with angles.

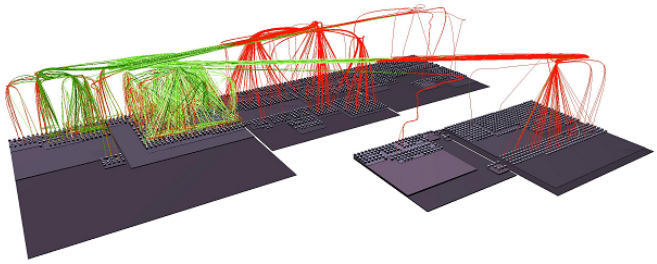


Fig. 11. Calls within core classes of JEdit. 2710 classes, 2350 edges representing 1 430 347 calls. $\beta = 0.8$. Curved edges.

1) Impact Of The Attraction Power: Figure 12 shows the same relations as figure 11, but with $\beta = 0.4$. This produces straighter edges, but results in more overlapping and more occlusions. Compared to the figure 11, it is much more difficult to see where relations come from and where they go.

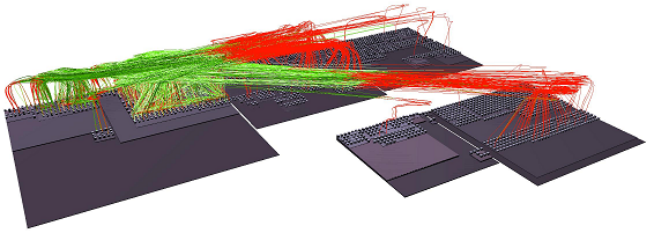


Fig. 12. $\beta = 0.4$

2) Hierarchical Attraction Points Removal: With our 3D-HEB visualization technique, 3D-HAPs can be removed to streamline the path between elements. For instance, we can keep only 3D-HAPs of the class level and the higher level of each path. In some cases, this 3D-HAPs removal is useful because edges take less detour, becoming easier to follow.

Figure 13 shows the impact of this 3D-HAP removal, the visualization being otherwise the same as the one on figure 11.

With the streamlining, each path has only two curves: one to leave the $originPath$ and one to enter the $destinationPath$.

3) Curved Edges or Straight Edges With Angles: Figure 14 shows the 3D-HEB technique with straight edges with angles instead of curves. It makes no semantic difference (besides aesthetic) whether edges are curved or straight with angles. In situations with a β near its limits([0 1]), one particular technique may be favored. In our experiments, we found that with low β (less than 0.3) the use of the straight edges with angles reduces the overlapping. On the other hand, with higher β (more than 0.97) curved edges form thicker clusters, which are easier to follow.

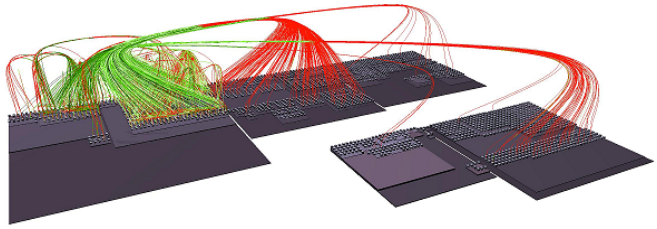


Fig. 13. With 3D-HAPs at the lower and the higher level of each path

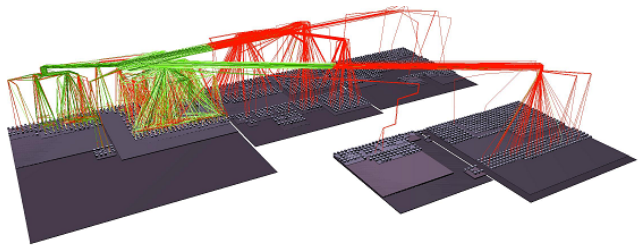


Fig. 14. With the straight edges with angles option.

VI. Conclusion and Future Work

Visualizing large software has always been an issue.

On the one hand, very interesting works exist in this domain, including research on software metaphors such as the city metaphor [16]. Their main asset is the effective display of the software structure, even for very large systems. However, the drawing of relations is generally one of their weak points, remaining unexplored or limited in the number of relations displayed in a clear and understandable way.

On the other hand, the Hierarchical Edge Bundles technique (HEB) [9] is a thrilling recent innovation which makes it possible to visualize relations in software an much more appealing and understandable way. However, it displayed software structure as a ring, which while being very readable might be considered as less intuitive by some developers.

In this paper, we explained our new visualization technique, which tries to build upon and combine the assets of the software city metaphor and the hierarchical Edge Bundles techniques, taking better advantage of the third dimension to avoid clutter because of relations.

In our technique, which we named 3D Hierarchical Edge Bundles (3D-HEB), the software structure is represented by a software city metaphor. Hierarchical Attraction Points are then placed in the 3D space (3D-HAPs), mirroring the software city layout, and are used to create the paths for edges that represent

the relations between the software elements. Thus clusters of edges are formed at different levels of the hierarchy, which produces less overlapping and less occlusions.

We proposed two different software city layouts, showing that our technique works independently of the layout of the software city, as did the original HEB technique [9]. The way edges are attracted by the 3D-HAPs is also configurable with the β factor, which helps the user have a visualization that fits her needs, with more or less elevation and dispersion of the edges in the 3D space. We detailed how the quantification and direction of relations are represented. Finally, we explained that the visualization may also be improved by controlling the density of 3D-HAPs, or the shape of the relations (curves or straight lines with angles).

As first results, we showed that with our 3D-HEB visualization it was possible to effectively display very large software systems, up to more than 2700 classes and 4.6 million calls, while keeping a good readability. The resulting visualization indeed tends to be more complete, displaying more of the information needed to understand software relations, while remaining understandable thanks to the clustering of edges. In addition, the visualization also provides an easy way to see relations between different levels of the hierarchy.

Much future work remains. On of our main goals is to perform larger experiments with numerous users, to better quantify the usability, efficiency and effectiveness of our 3D-HEB technique, and to further improve it. Another short-term goal is to continue building our VITRAIL Visualization tool so as to improve the navigability in this software visualization, and, on a much longer-term, allow users to directly interact with the visualization to modify the software.

REFERENCES

[1] P. Caserta and O. Zendra, "Visualization of the static aspects of software: a survey," *IEEE Trans. on Vis. & Computer Graphics*, 2011.

[2] T. Munzner, "H3: Laying Out Large Directed Graphs in 3D Hyperbolic Space," in *IEEE Symp. on Info. Vis.*, 1997, pp. 2–10.

[3] I. Herman, M. Marshall, and G. Melançon, "Graph Visualization and Navigation in Information Visualization: A Survey," *IEEE Trans. on Vis. & Computer Graphics*, 2000.

[4] C. Lewerentz and A. Noack, "CrocoCosmos - 3D Visualization of Large Object-Oriented Programs," *Graph Drawing Software*, 2003.

[5] A. Noack and C. Lewerentz, "A Space of Layout Styles for Hierarchical Graph Models of Software Systems," in *Proc. ACM Symp. on Soft. Vis.*, 2005, pp. 155–164.

[6] H. Schulz and H. Schumann, "Visualizing Graphs - a Generalized View," in *IEEE Conf. on Info. Vis.*, 2006, pp. 166–173.

[7] J. Stasko and E. Zhang, "Focus + Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations," in *IEEE Symp. on Info. Vis.*, 2000, pp. 57–65.

[8] M. Balzer and O. Deussen, "Level-of-Detail Visualization of Clustered Graph Layouts," in *Asia-Pacific Symp. on Vis.*, 2007.

[9] D. Holten, "Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data," *Trans. on Vis. & Computer Graphics*, pp. 741–748, 2006.

[10] M. Storey, F. Fracchia, and H. Müller, "Cognitive Design Elements to Support the Construction of a Mental Model During Software Exploration," *J. of Syst. & Soft.*, vol. 44, 1999.

[11] M. Scaifeand and Y. Rogers, "External Cognition: How Do Graphical Representations Work?" *Int'l J. of Human-Computer Studies*, vol. 45, pp. 185–213, 1996.

[12] M. Petre, A. Blackwell, and T. Green, "Cognitive Questions in Software Visualization," *Soft. Vis.: Prog. as a Multimedia Experience*, 1998.

[13] M. Tudoreanu, "Designing Effective Program Visualization Tools for Reducing User's Cognitive Effort," in *Symp. on Soft. Vis.*, 2003.

[14] J. Stasko, *Software Visualization: Programming as a Multimedia Experience*. MIT press, 1998.

[15] S. Diehl, *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer Verlag, Inc., 2007.

[16] A. Dieberger, "The Information City - a Step Towards Merging of Hypertext and Virtual Reality," *Conf. on Hypertext*, vol. 93, 1993.

[17] A. Dieberger, "Navigation in Textual Virtual Environments Using a City Metaphor," Ph.D. dissertation, Vienna Univ. of Tech., 1994.

[18] A. Dieberger and A. Frank, "A City Metaphor to Support Navigation in Complex Information Spaces," *J. of Visual Languages & Computing*, vol. 9, no. 6, pp. 597–622, 1998.

[19] D. Holten, B. Cornelissen, and J. Van Wijk, "Trace visualization using hierarchical edge bundles and massive sequence views," in *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*. IEEE, 2007, pp. 47–54.

[20] M. Green and J. Rekimoto, "The Information Cube: Using Transparency in 3D Information Visualization," in *3rd Workshop on Info. Tech. & Syst.*, 1993, pp. 125–132.

[21] C. Dos Santos, P. Gros, P. Abel, D. Loisel, N. Trichaud, J. Paris, C. Telecom, and S. Antipolis, "Mapping Information onto 3D Virtual Worlds," in *Int'l Conf. on Info. Vis.*, 2000, pp. 19–21.

[22] C. Knight and M. Munro, "Virtual But Visible Software," in *4th IEEE Int'l Conf. on Info. Vis.*, 2000, pp. 198–205.

[23] D. Ploix, "Analogical Representations of Programs," in *1st Int'l Workshop on Vis. Soft. for Understanding and Analysis*, 2002.

[24] E. Kleiberg, H. van de Wetering, and J. Van Wijk, "Botanical Visualization of Huge Hierarchies," in *Symp. on Info. Vis.*, 2001.

[25] C. Russo Dos Santos, P. Gros, P. Abel, D. Loisel, N. Trichaud, and J. Paris, "Metaphor-Aware 3D Navigation," in *IEEE Symp. on Info. Vis.*, 2000, pp. 155–165.

[26] G. Langelier, H. A. Sahraoui, and P. Poulin, "Exploring the Evolution of Software Quality with Animated Visualization," in *IEEE Symp. on Visual Lang. and Human-Centric Comp.*, 2008.

[27] T. Panas, T. Epperly, D. Quinlan, A. Saebjornsen, and R. Vuduc, "Communicating Software Architecture using a Unified Single-View Visualization," in *12th IEEE Int'l Conf. on Eng. Complex Computer Syst.*, 2007, pp. 217–228.

[28] F. Steinbruckner and C. Lewerentz, "Representing development history in software cities," in *Proceedings of the 5th international symposium on Software visualization*. ACM, 2010, pp. 193–202.

[29] H. Yang and H. Graham, "Software Metrics and Visualisation," Univ. of Auckland, Tech. Rep., 2003.

[30] G. Langelier, H. Sahraoui, and P. Poulin, "Visualization-Based Analysis of Quality for Large-Scale Software Systems," in *20th IEEE/ACM Int'l Conf. on Automated Soft. Eng.*, 2005, pp. 214–223.

[31] K. Dhambri, H. A. Sahraoui, and P. Poulin, "Visual Detection of Design Anomalies," in *12th Euro. Conf. on Soft. Maintenance and Reeng.*, 2008.

[32] R. Wettel and M. Lanza, "Visualizing Software Systems as Cities," in *4th IEEE Int'l Workshop on Vis. Soft. for Understand. & Analysis*, 2007.

[33] R. Wettel and M. Lanza, "Visually Localizing Design Problems with Disharmony Maps," in *4th ACM Symp. on Soft. Vis.*, 2008.

[34] S. Alam and P. Dugerdil, "Evospaces Visualization Tool: Exploring Software Architecture in 3D," *14th Conf. on Rev. Eng.*, 2007.

[35] R. Wettel, M. Lanza, and R. Robbes, "Software systems as cities: A controlled experiment," in *Proceedings of ICSE*, vol. 11, 2011.

[36] B. Johnson and B. Shneiderman, "Tree-Maps: a Space-Filling Approach to the Visualization of Hierarchical Information Structures," in *IEEE Conf. on Vis.*, 1991, pp. 284–291.

[37] B. Shneiderman, "Tree Visualization with Tree-Maps: 2D Space-Filling Approach," *ACM Trans. On Graphics*, vol. 11, no. 1, 1992.

[38] W. Wang, H. Wang, G. Dai, and H. Wang, "Visualization of Large Hierarchical Data by Circle Packing," in *ACM SIGCHI Conf. on Human Factors in Computing Systems*, 2006, pp. 517–520.

[39] S. Alam and D. Ph, "EvoSpaces: 3D Visualization of Software Architecture," in *Int'l Conf. on Soft. Eng. and Knowledge Eng.*, 2007.