Manhattan — 3D City Visualizations in Eclipse

Alberto Bacchelli, Francesco Rigotti, Lile Hattori, and Michele Lanza

REVEAL @ Faculty of Informatics - University of Lugano, Switzerland

Abstract. Software visualization eases program comprehension through visual metaphors, which leverage the power of the human eye to identify colors, shapes, patterns, and differences. Even though many powerful software visualization tools exist, the majority of them consists of stand-alone systems that are not integrated with the development tools already in use.
We present *Manhattan*, an *Eclipse* plugin that visualizes projects in the workspace as 3D cities. While working on a software system project, developers can see its representation, updated in real time according to the code changes performed. Moreover, Manhattan does not work in isolation: When other developers modify the same project in their Eclipse instance, Manhattan visualizes them to increase developers' awareness to facilitate team collaboration and coordination.

1 Introduction

The first requirement to properly evolve a software system is program comprehension: Before making changes, one needs to know what to change and how. However, understanding a software system is not trivial, because software systems are complex, intangible, and constantly evolving. Software visualization techniques can ease this process, as they allow one to perceive software through vision, the sense from which humans acquire most information [2]. Many powerful software visualization tools have been created to support program comprehension, design assessment, evolution analysis, and reverse engineering tasks. Most of these tools are not integrated on the IDE, thus forcing programmers to move away from the development environment. To best support developers, however, a software visualization tool should be tightly integrated in the development process and used throughout the entire life of the system.

With these considerations in mind, we devised *Manhattan*, a software visualization tool in the form of an *Eclipse* plugin. Manhattan's goal is twofold:

1. Help developers to reason on a system, thus making it easier to understand its architecture, and, as a consequence, help them to properly drive the system's evolution.
2. Support collaboration between members of a development team by increasing each member's awareness of the activity of the team.

Manhattan visualizes projects in the Eclipse workspace by using the 3D city metaphor devised by Wettel *et al.* in *CodeCity* [3][4], which exploits the similarities between software constructs and the urban landscape of a city. The city metaphor gives a shape to the otherwise intangible software, exploiting the human capacity to build a visual mental model of the system to ease its comprehension.

In addition, Manhattan improves awareness by showing to developers a living city where changes from the members of the team and potential conflicting code are animated with different colors and shapes. To do that, Manhattan relies on APIs provided by the *Syde* plugin for Eclipse [1]. Merge conflicts can be examined in an instance of *Eclipse*'s Compare Editor opened from within the visualization.

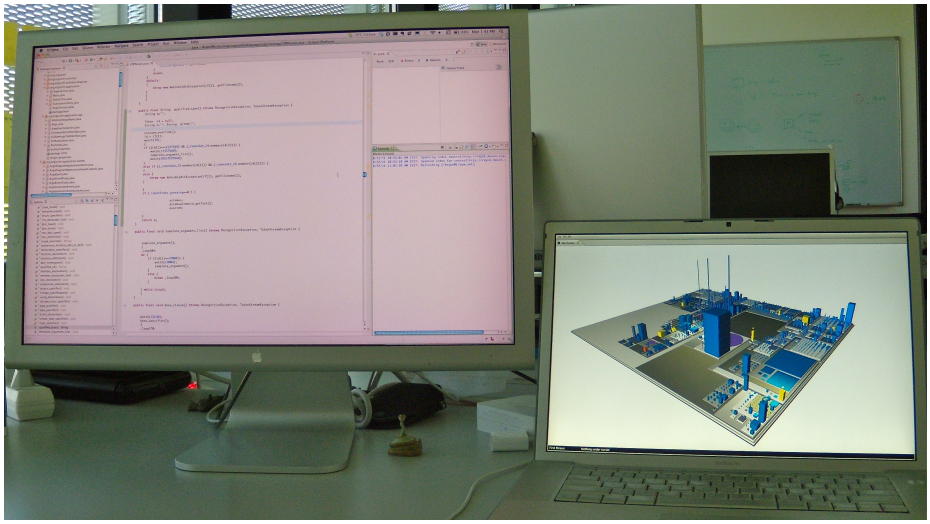

Fig. 1. The Manhattan Eclipse Plugin In Action

2 Manhattan In Action

Figure 1 shows Manhattan working using our recommended setup: A dual-monitor layout in which one of the two displays shows the interactive city visualization. The city metaphor maps architectural elements to software entities: Projects are represented as cities, packages as districts, and classes as buildings. For space reason, further information about the 3D city metaphor can be found in the CodeCity website[1]. In our approach, the dimensions of each building is mapped to the number of fields (NOF) and the number of methods (NOM) of any class. To clearly distinguish them, interfaces are represented by circular buildings in a special color.

The user can navigate the city through the *orbital* mode, where the camera is fixed on a dome centered on top of the city and users move along the dome, or the *first-person* navigation mode, which gives full control over the movement and allows any kind of translation or rotation. By mouse hovering on any building, a tooltip that describes

[1] `http://codecity.inf.usi.ch`

the metrics (*i.e.,* NOF, NOM, and LOC) of the corresponding class appears; by right-clicking, Manhattan opens the class in the Java editor. Manhattan also includes a *focused* view, in which only the chosen entities are displayed: The users select any number of entities and press the v key to activate this view. Moreover, by clicking on a class and pressing h, Manhattan presents a view with only the hierarchically related classes.

To visualize systems and information, Manhattan needs a language-specific parser that extracts the model from the system's code. Manhattan exploits the abstract syntax trees provided by the language-integration plugins (*e.g.,* JDT for JAVA). Currently, Manhattan handles JAVA projects by building on top of X-Ray[2], a software visualization plugin that uses the JDT APIs to extract projects' information.

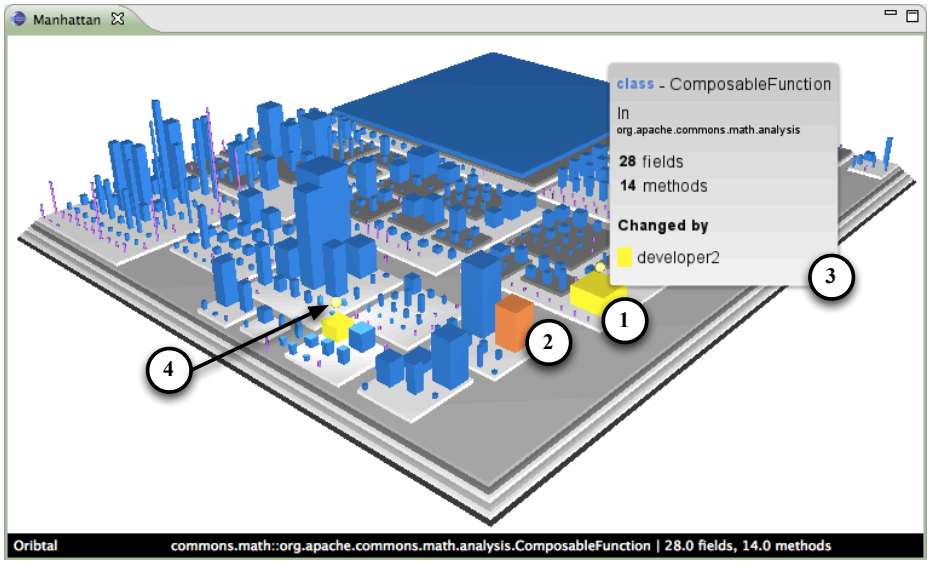

Fig. 2. Collaboration Support Visualization

3 Manhattan - Visualization of Collaborative Development

Figure 2 how Manhattan visualizes information to improve collaboration and team awareness. Manhattan shows (i) the emerging conflicts in which the developer is involved, (ii) the classes that have been changed or deleted by the other developers, and (iii) the developer who modified a class more frequently.

Manhattan can visualize the system while it is being changed by several developers in parallel, because Syde[3] monitors source code changes directly at each developer's

[2] `http://xray.inf.usi.ch/`
[3] `http://syde.inf.usi.ch`

workspace. Thus, it captures changes and broadcasts its information at every build action, rather than at every commit.

Let us consider a team of three developers working together and analyze their interaction from the perspective of developer 1. When any class is changed by someone other than 1, she sees the corresponding building turning yellow (Point 1). As soon as one of her colleagues removes any class, the building does not disappear in her view, but becomes orange (2). Supplementary change information is included in the tooltip description for classes (3), which lists the names of the developers who recently modified the given class. Developers are grouped by the kind of change they performed and those that deleted the class are put first.

To visualize a conflict alert on a class, we put a sphere on top of its building (4). The color depends on the kind of conflict the sphere represents: Emerging conflicts are showed with a yellow sphere, while committed conflicts (those in which one of the involved developers has committed his changes) are shown with a red sphere. We also use the concept of *conflict beacon*: a spotlight positioned above of a conflict sphere and pointing towards the ground. These beacons illuminate an area of the city around their associated conflict spheres, so that conflicts are always clearly visible. After the user notices an alert and puts the mouse on the conflict sphere, the beacon is deactivated.

4 Conclusion

We presented Manhattan, a plugin to visualize software systems as 3D cities within Eclipse. One of its key points is the tight integration with the IDE: Manhattan offers developers a high-level view of the system, updated in real-time, *while it is being developed*. Users can navigate on the representation of the system and interact with it, for instance, to obtain source code metrics (*e.g.,* number of methods), or to directly open and read the corresponding code in the Eclipse editor. Manhattan offers *focused* views to concentrate on specific entities or on the hierarchy of a systems' class. In the context of the city metaphor, Manhattan includes visualizations—built on top of the Syde plugin's API—to improve collaboration and team awareness.

As future work, we plan to improve how we visualize change notifications and conflict alerts, by using simple textures and small animations. We also plan to provide an additional view of the city, focused on displaying code ownership information, based on the change information provided by Syde.

Manhattan is available at `manhattan.inf.usi.ch`, where the reader finds additional screenshots, a screencast, and the Eclipse update site to download the plugin.

References

1. Hattori, L., Lanza, M.: Syde: A tool for collaborative software development. In: Proc. of ICSE 2010 (32nd Int'l Conf. on Software Engineering). pp. 235–238 (2010)
2. Ware, C.: Information Visualization: Perception for Design. Morgan Kaufmann (2004)
3. Wettel, R.: Software Systems as Cities. Ph.D. thesis, University of Lugano, CH (Sep 2010)
4. Wettel, R., Lanza, M., Robbes, R.: Software systems as cities: A controlled experiment. In: Proc. of ICSE 2011 (33rd Int'l Conf. on Software Engineeering). pp. 551 – 560 (2011)