VisUML: Live & Interactive Diagrams

Mickaël Duruisseau, Jean-Claude Tarby, Xavier Le Pallec, Sébastien Gérard

VisUML: Live & Interactive Diagrams

Mickaël Duruisseau
CEA LIST
CEA - Boîte 94
91191, Gif sur Yvette, France
mickael.duruisseau@cea.fr

Jean-Claude Tarby
University of Lille
CRIStAL, UMR 9189
59650, Villeneuve d'Ascq, France
jean-claude.tarby@univ-lille1.fr

Xavier Le Pallec
University of Lille
CRIStAL, UMR 9189
59650, Villeneuve d'Ascq, France
xavier.le-pallec@univ-lille1.fr

Sébastien Gérard
CEA LIST
CEA - Boîte 94
91191, Gif sur Yvette, France
sebastien.gerard@cea.fr

ABSTRACT

A classic Integrated Development Environment (IDE) allows displaying information only with a textual representation. This kind of representation is perfect for the linear aspect of the code, but not effective to represent links between code fragments. Current graphical code representation modules in IDE are suited to apprehend the system from a global point of view. However, the cognitive integration cost of those diagrams is disproportionate related to the elementary coding task.

Our approach considers graphical representation but only with code elements that are parts of the developer's mental model during his programming task. The corresponding cognitive integration of our graphical representation is then less costly and the information that text struggles to display will be clearly explicit. We use UML for this representation because it is a widespread and well-known formalism.

We want to show that dynamic diagrams, whose content is modified and adapted in real-time by monitoring each action of the programmer in the IDE can be of great benefit as their contents are perfectly suited to the developer current task. With our live diagrams, we provide to developers an efficient way to navigate through textual and graphical representation.

CCS CONCEPTS

• **Human-centered computing → Human computer interaction (HCI)**; **Graphical user interfaces**; **Interaction devices**; **Interaction techniques**; *HCI theory, concepts and models*; *Interactive systems and tools*; Interaction design; • **Software and its engineering → Model-driven software engineering**; Software prototyping;

IHM '17 , August 28–September 1, 2017, Poitiers, France

KEYWORDS

HCI; MDE; Software Engineering; UML; Papyrus; Human-Centered Design

RÉSUMÉ

Les IDE actuels permettent d'afficher des informations sous forme de texte. Ce genre de représentation est parfait pour l'aspect linéaire du code, mais est moins efficace pour représenter les liens entre les différents morceaux du code. Les modules actuellement développés qui affichent une représentation graphique du code sont conçus pour appréhender le système d'un point de vue global. Cependant, l'intégration cognitive de ces diagrammes est disproportionnée par rapport à la tâche de programmation.

Notre approche utilise des représentations graphiques contenant uniquement les éléments du code qui font partie du modèle mental du développeur pendant sa tâche de programmation. L'intégration cognitive résultante à nos représentations est donc moins coûteuse et les informations mal représentées par le texte sont plus clairement explicitées. Nous utilisons UML pour ces représentations car c'est un langage connu et répandu.

Nous voulons montrer que des diagrammes dynamiques, dont le contenu est modifié et adapté en temps réel à chaque action sur l'IDE du développeur, peuvent être utiles. En effet, leurs contenus sont adaptés à la tâche active du développeur. Avec nos live diagrammes, nous fournissons une façon efficace de naviguer entre le code et les différentes représentations graphiques.

MOTS-CLEFS

Interaction Homme-Machine; Ingénierie Dirigée par les Modèles; Génie Logiciel; Méta-Modèle; UML; Papyrus; Conception Centrée sur l'Humain

1 INTRODUCTION

Human-Computer Interaction (HCI) has significantly evolved in recent years with the appearance of mobile and tactile devices, voice and gesture recognition, augmented and virtual reality, etc. Nowadays, most of the smartphone users know how to interact with a map, using simple interactions like touch, but also some more complex, like swipe or pinch. In the meantime, software practitioners still develop applications only with a keyboard and a mouse.

Furthermore, 'development tools are showing mainly text with (so much) obstinacy' [3] despite some improvements concerning HCI in their IDE, like syntax coloration and auto-completion.

We may consider software visualization tools as an improvement of the HCI, but their place in IDE and their use remain anecdotal. Visualization tools generally help developers to understand the global architecture of the application they are working on or the impact of what they are changing. Development consists mainly in producing code but not dealing with considerations of macroscopic nature. We argue these visualization tools are not focused on the most important and elementary task: programming. We claim that a graphical representation of elements that are currently knitted by a programmer may be more easily accepted.

The first reason is it can quickly provide information that is less visually explicit in textual code and still relevant for coding. In particular, it may highlight the different relations between elements (structural relations or specific execution flow). The second reason is that such representations (the graphical ones) are more suited to mobile and tactile devices (like tablets) than textual code and so, by taking advantage of them, they can provide HCI improvements of IDE.

We have chosen the UML language for the graphical representation because it is a language known and mastered by developers, even if according to different surveys[1, 2, 6] it is not enough used in firms. This choice was made according to the principle of cognitive integration[5]: adapt to the knowledge of developers. This kind of concern is the heart of the cognitive dimensions[4] and we aim to reduce the cognitive charge of the developers. In our case, when switching from a textual code editor to a graphical representation, it is clearly necessary that programmers keep their references, therefore the graphical representation has to be close to their mental model.

2 VISUML PRESENTATION

VisUML is a live diagramming approach that we designed and that implements this point of view of software visualization. It allow developers to have a live and interactive view of their code.

Once enabled, VisUML renders a live and interactive Class Diagram, shown in Figure 1, that displays classes opened in the IDE, as well as related unopened ones. Easy navigation interactions are implemented:

- Click on a class: switch the active tab to the related file
- Click on an attribute: scroll to the associated line and highlight it
- Click on a method: same, and update the sequence diagram to display the clicked method.

In addition, VisUML also displays a Sequence Diagram, presented in Figure 2, that reflect the currently browsed method (in the code), or the clicked one in the Class Diagram. This diagram shows information about the sequential flow of the body of the method. Every element is interactive and a click on it will scroll and highlight the associated line in the IDE. Moreover, a special interaction (currently implemented with an alt+click on a message that refers to a method that can be displayed) allows users to easily navigate between methods in this diagram. In addition to these interaction, a caret listener triggers the update of the sequence diagram when the user's caret is inside a method. Several utility functions are also

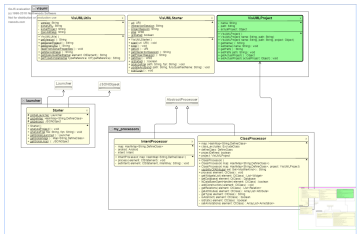

Figure 1: An example of class diagram in VisUML

available, and allow users to show more or less information, such as the depth level of the diagram.

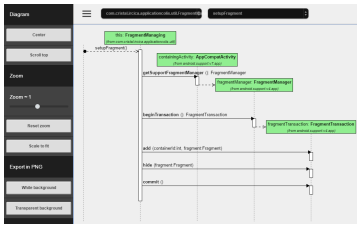

Figure 2: An example of sequence diagram

In this demo we want to show the benefits of using live and interactive diagrams. Our scenario will refer to an understanding phase, when the developer must learn how an existing project is structured. Using only VisUML, he will be able to see and navigate in a graphical representation of this project.

A presentation video is available at this address: https://www.youtube.com/watch?v=buyGojmbUpQ. This video shows how the tool works and all the possible interactions that are actually implemented. Since our tool is working both on the IDE and a web page, it is simpler to show the interactions with a movie than with pictures. Finally, an often updated document, is available online: http://these.mickaelduruisseau.fr/VisUML/doc/index.html. It present how to install the plugin, and sum-up the different interactions and utility functions.

REFERENCES

[1] Ateret Anaby-Tavor, David Amid, Amit Fisher, Avivit Bercovici, Harold Ossher, Matthew Callery, Michael Desmond, Sophia Krasikov, and Ian Simmonds. 2010. Insights into enterprise conceptual modeling. *Data and Knowledge Engineering* 69, 12 (2010), 1302–1318. DOI: http://dx.doi.org/10.1016/j.datak.2010.10.003

[2] Michel R. V. Chaudron, Werner Heijstek, and Ariadi Nugroho. 2012. How effective is UML modeling ? *Software & Systems Modeling* 11, 4 (2012), 571–580. DOI: http://dx.doi.org/10.1007/s10270-012-0278-4

[3] T Gîrba and A Chiş. 2015. Pervasive software visualizations (keynote). In *2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT)*. 1–5. DOI: http://dx.doi.org/10.1109/VISSOFT.2015.7332409

[4] T R G Green and M Petre. 1996. Usability Analysis of Visual Programming Environments: a 'cognitive dimensions' framework. *JOURNAL OF VISUAL LANGUAGES AND COMPUTING* 7 (1996), 131–174.

[5] Daniel L Moody. 2009. The "Physics" of Notations: Towards a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering* 35 (2009), 756–779.

[6] Marian Petre. 2013. UML in practice. *Proceedings - International Conference on Software Engineering* (2013), 722–731. DOI: http://dx.doi.org/10.1109/ICSE.2013.6606618