Towards Visual Reflexion Models

Marcello Romanelli, Andrea Mocci, and Michele Lanza
REVEAL @ Faculty of Informatics — University of Lugano, Switzerland

***Abstract*—Source code and models of a software system, like architectural views, tend to evolve separately and drift apart over time. Previous research has shown that it is possible to effectively relate them through a *reflexion model*, defined as a "summarization of a software system from the viewpoint of a particular high-level model". While effective, the process of constructing and analyzing reflexion models was supported by text-based tools with limited visual representation. With the original approach, it was relatively hard to understand which parts of the system were represented, and which parts of the system contributed to specific relations in the reflexion model.**

We present our vision on augmenting the construction and analysis of reflexion models with visual support, effectively providing the basis for *visual reflexion models*. We describe our approach, implemented as a web-based application, and two promising case studies involving two open-source projects.

I. Introduction

Despite long-lasting research on software architecture, the typical vehicle to describe the high-level structure of a software system often consists of informal diagrams drawn on paper or on the whiteboard [1]. Since they represent a developer's mental model, such diagrams are better communicated with simple visual metaphors, rather than be forced to adhere to a specific architectural description language, that is rarely adopted in practice despite the success in academia [2].

A fundamental issue is that there is no automatic way to relate source code artifacts and the high-level model of a developer. For this reason, the source code and the model of a system evolve separately and tend to drift apart over time [3]. Murphy *et al.* introduced *reflexion models* to solve this problem and effectively relate source code and high-level abstractions of a system [4]. A reflexion model is a "summarization of a source model of a software system from the viewpoint of a particular high-level model".

Our vision is to investigate how visualization and visual features may enrich both the expression and construction of reflexion models, providing full-fledged *visual reflexion models*. As an initial outcome of this investigation, we show how simple interactive visualization techniques may help to construct reflexion models, understand the coverage of the models with respect to the original system, and enable an effective consistency analysis between the high-level model and the reflexion model. As an early assessment of our approach, we constructed two case studies from two well-known open-source projects, JHotDraw and ArgoUML, by extracting, analyzing and refining two respective architectural views. The contributions we make with this paper are (1) a novel visual approach for building and analyzing reflexion models, and (2) a web-based implementation of the visual reflexion model approach.

II. Reflexion Models and Related Work

The reflexion model approach unifies reverse architecting a software system model and its comparison with a high-level model, following 5 steps: (1) The developer specifies a high-level model of the system as it is present in her own mind. (2) She extracts a source model from the source-code artifacts by taking into account the structure of the repository in the file system and the textual content of the source code. (3) The developer describes a *mapping* between the source model entities and the entities in the high-level model. (4) The reflexion model is computed. (5) The reflexion model is inspected/analyzed by the developer.

The original approach by Murphy et al. has been successfully applied to very complex systems [4], such as understanding the architecture of Microsoft Excel. A developer reported that in 2 weeks he was able to understand concepts of the system that with documentation-based approaches would have taken more than 2 years. The reflexion model approach was supported by text-based tools with limited visual representation. We believe that some limitations, like the difficulty to understand which parts of the system were represented, and which parts of the system contributed to specific relations in the reflexion model, can benefit from a visual approach.

The reflexion model approach is related to "reverse architecting", described by Krikhaar as "a flavor of reverse engineering that concerns all activities for making existing (software) architectures explicit" [5]. In Riva's reverse architecting approach, the analysis starts from code and moves towards the architecture, to extract models of a software system starting from its implementation [6]. Bowman et al. call those two models respectively conceptual and concrete architecture [7]. The former represents "how developers think about the system" while the latter represents "the relationships that exist in the implemented system". Riva's approach aimed at assisting the activities of software architecture recovery trough six macro-phases, from the definition of architectural concepts to the construction and analysis of the architectural view. Mens *et al.* proposed an approach to document and co-evolve high-level structural regularities in code [8]. Among the tools developed, the *Intensional View Editor* can be used to produce *intensional views*, *i.e.,* a set of source-code entities (*e.g.,* classes) which share a common structure. An intention is an executable description that when evaluated produces the set of entities (*i.e.,* the extension) belonging to the view.

We create a bridge between these two streams of research, by extending the original reflexion model approach [4] with interactive visualization to enable "visual reflexion models".

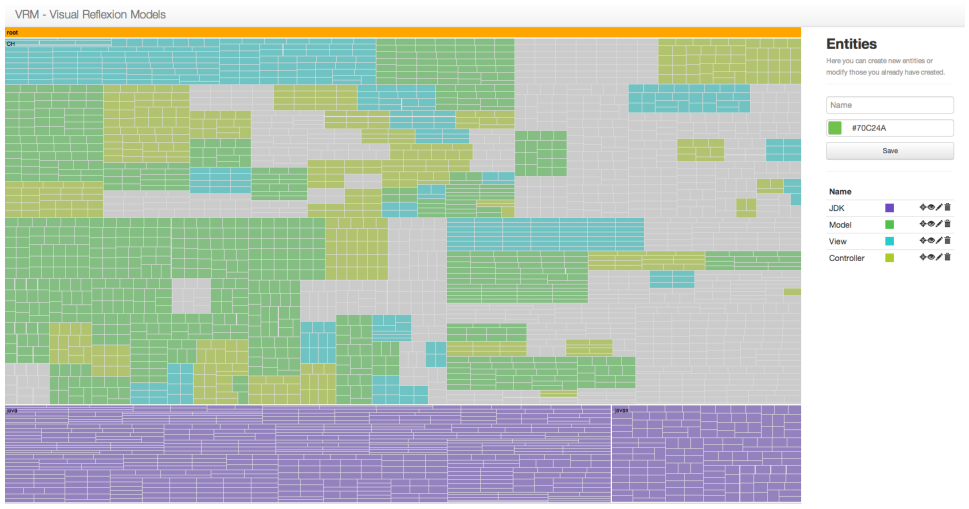

VRM - Visual Reflexion Models
root
CH
java
javax
Entities
Here you can create new entities or modify those you already have created.
Name
#70C24A
Save
Name
JDK
Model
View
Controller

Fig. 1: Mapping Construction View

III. Visual Reflexion Models

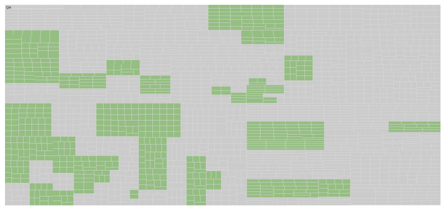
We adapted and reimplemented the original approach by integrating and providing visual support for reflexion models in a web application (see Figure 1). We use the Famix meta-model [9] to represent object-oriented systems. While the use of Famix enables detailed querying and reasoning of source models, the core contribution of this paper focuses on the visual support for the construction of mappings (step 3 of the original approach) and reflexion models analysis (step 5).

A. *Visual support for Mapping Construction*

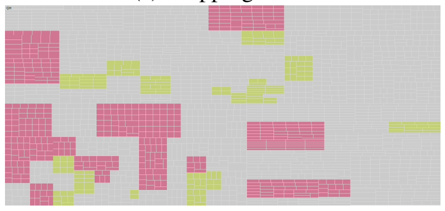
The original reflexion model approach [4] adopted regular expressions to specify mappings. We also adopt regular expressions, but we exploit the more detailed structural information of Famix to construct more specific queries that can identify entities both by name and by logical relationships (*e.g.,* all the classes contained in a namespace with a given name).

Figure 1 shows the main view for the construction of mappings, supported by visualizing the source code with a *squarified treemap*. Treemaps have become a *de facto* method to visualize large, tree-structured information spaces using nested rectangles [10]. Each branch of the tree is a rectangle, which is then sub-divided in smaller rectangles representing the various sub-branches. The area of the leaf nodes is proportional to a specified dimension of the data. We adopt the *squarified* treemaps layout algorithm by Bruls et al., which, by working on both dimensions (horizontal and vertical), tries to approximate each sub-rectangle to the shape of a square. With this algorithm a treemap acquires a more understandable layout. While the use of treemaps for software visualization is not novel (*e.g.,* [11]), in the context of mapping construction, the treemap contributes in two main aspects: (1) It gives an immediate view of which parts of the system are covered by the mappings; and (2) it helps to relate the high level mental-model entities with the source code logical structure, like namespace/package organization.

Mappings are constructed as union of queries, corresponding to disjoint sets of entities. When editing a specific mapping for a high-level entity, the view is specialized into the query view, which shows each query with a different color (Figure 2).



(a) Mapping view.



(b) Query view.

Fig. 2: Mapping and detailed query view.

The separation between these different views is fundamental to decompose the mapping construction and visualize the different components that characterize each mapping. It is also important when analyzing the reflexion model. We also devised two functionalities that further help the construction of mappings: *elision* and *focus*.

Elision hides all the entities of a mapping from the treemap. It is effective when the developer is confident of a mapping, *i.e.,* that it represents a given high-level entity (*e.g.,* after checking the covered source entities). Elision reduces the number of entities shown in the treemap,the effort of looking for entities in the source model that may need to be included in another mapping, and alleviates the effort of constructing new mappings when the previous ones have been consolidated.

Focus instead forces the treemap to show only the entities of a particular mapping. When the developer wants to refine a mapping and check if all the entities pertain to it, it may be hard to have all the system shown in the treemap. Focusing facilitates this task, since the developer can check fewer entities while keeping their logical organization in the system.

B. Visual Support for Reflexion Model Analysis

The last step exploits visualization to support analysis. Figure 3 shows an example of reflexion model: Arrows represent relations (*e.g.,* calls) between high-level model entities. When an arrow is present in both the high-level and the reflexion model, it is called *convergent* and it is solid and colored in green; when it is present in the reflexion model but not in the high-level model it is called *absent* and it is dashed and red; when it is present in the high-level model but not in the reflexion model it is called *divergent* and it is red and solid.

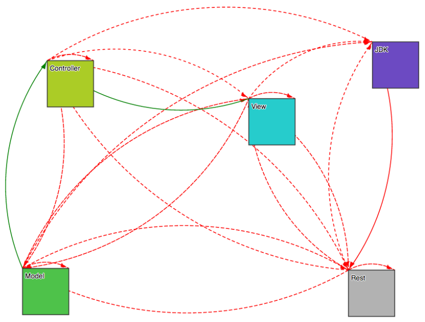


Fig. 3: A Reflexion Model.

In the original approach [4], the user could inspect the original relations in the source model as a list. We provide two ways to visually inspect the reflexion model at two levels of abstraction: the *arc-to-query* and the *query-to-entities view*.

The arc-to-query view (Figure 4) details a convergent or absent arc by showing which queries in each single entity contribute to the arc. Each box represents a query contained in a separate convex hull, representing its high-level entity.

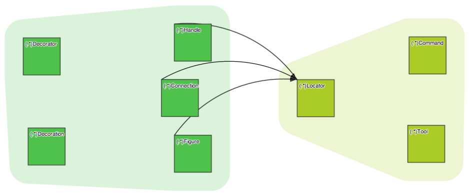
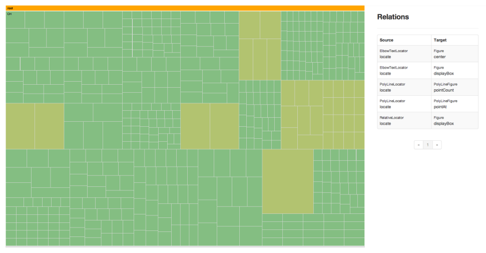


Fig. 4: Relations between mappings in the arc-to-query view.

The query-to-entities view (Figure 5) details a relation between two queries through a treemap and a list of relations between the related entities in the source model (*e.g.,* method calls). The user can select a specific relation between two entities and the view gets updated by highlighting the respective entities in the treemap.

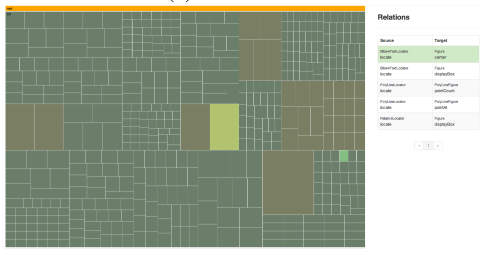


Relations

Source	Target
ElbowTextLocator locate	Figure center
ElbowTextLocator locate	Figure displayBox
PolyLineLocator locate	PolyLineFigure pointCount
PolyLineLocator locate	PolyLineFigure pointAt
RelativeLocator locate	Figure displayBox

« 1 »

(a) Default view



Relations

Source	Target
ElbowTextLocator locate	Figure center
ElbowTextLocator locate	Figure displayBox
PolyLineLocator locate	PolyLineFigure pointCount
PolyLineLocator locate	PolyLineFigure pointAt
RelativeLocator locate	Figure displayBox

« 1 »

(b) A single relation highlighted

Fig. 5: Inspection of a relation between entities in two queries.

Summing Up. We discussed the visual support to construction and analysis of reflexion models provided by our approach. We illustrated how squarified treemaps can help in the construction of mappings, and how specialized, dedicated visualizations can help to understand convergent and absent arcs in reflexion models. In the next section, we discuss two preliminary case studies constructed with our approach.

IV. Case Studies

JHotDraw. We apply our approach to construct a reflexion model of the *model-view-controller (MVC)* architectural pattern [12] of JHotDraw, a Java framework to create drawing editors. As pointed out in the official documentation of JHotDraw, it implements MVC. We considered a partial UML diagram extracted from the documentation, shown in Figure 6, as a reference for the reflexion model construction.

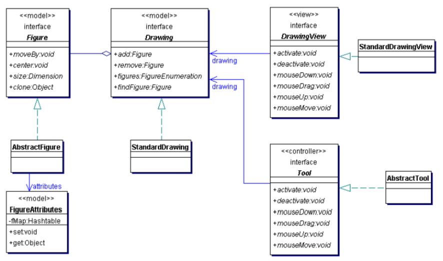


Fig. 6: A UML diagram of JHotDraw.

The diagram above is partial, so we can immediately see from the treemap view that the coverage of the system is low and so we expected the reflexion model to contain (at least) absent edges. By inspecting the packages where the classes from Figure 6 are located, we found that many subclasses do not belong to any entity in the reflexion model. We started refining the reflexion model mapping by adding such subclasses and we obtained the mapping shown in Figure 1, that shows also the classes pertaining to the Java library. All the remaining classes are essentially utility classes.

ArgoUML is an open-source round-trip modeling tool for UML. Despite an active community, the official manual of ArgoUML[1] does not contain an official description of the system architecture. We considered a possible architecture shown in Figure 7, taken from a Software Tools and Techniques course taught at the University of Auckland[2].

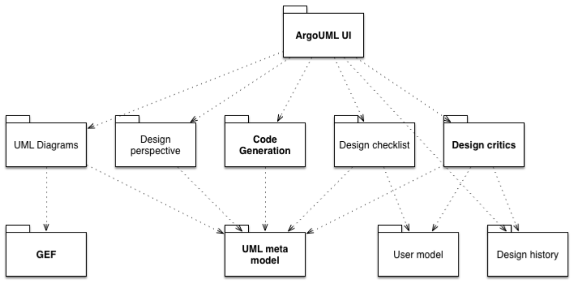


Fig. 7: High-level architecture of ArgoUML.

For space reasons, we consider only a subset of the possible entities: the ArgoUML UI, GEF, Design Critics, UML Meta-Model and Code Generation. One of the positive aspects of ArgoUML is that the architecture is related to the logical organization of classes in packages. From the documentation we discovered that each component in ArgoUML must place its UI classes in a package ending with UI. Thus, we define a single mapping for the ArgoUML UI entity matching every class contained in a package ending in UI. This matched a large portion of the system. Similar structural conventions hold for code generators and design critics. The GEF component and the meta-model are located in specific packages. Figure 8 shows the computed reflexion model of ArgoUML for the reference architecture. It mostly matches the reference architecture, except for some absent arcs. Such absences are consistent after checking the involved source code entities.

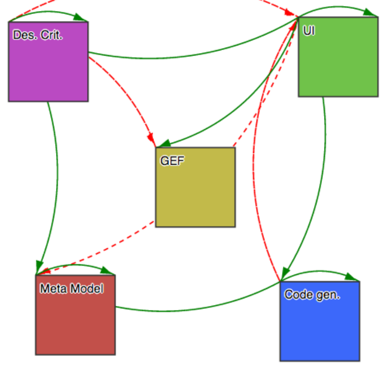


Fig. 8: Reflexion Model of ArgoUML.

V. Conclusion

We presented a preliminary approach, implemented as an interactive web application, to augment the construction and analysis of reflexion models with ad-hoc visualizations, providing the basis for *visual reflexion models*.

While our preliminary results look promising, we plan to research on two missing pieces on our long-term vision. First, we will refine the mechanisms used to analyze reflexion models to support a better workflow for refinement. Second, we plan to augment the expression of reflexion models with more than just boxes and arrows, for example by using well established visual metaphors typical of architectural description languages. Finally, we plan to fully evaluate our approach through more case studies, possibly involving industry, at a level of complexity close to the one used to assess the original reflexion model approach by Murphy *et al.* [4].

Acknowledgments. We gratefully acknowledge the financial support of the Swiss National Science foundation for the project "ESSENTIALS" (SNF Project No. 153129).

[1]See http://argouml-stats.tigris.org/documentation/manual-0.32/

[2]See http://tinyurl.com/law2r6g

References

[1] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: How and why software developers use drawings," in *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. ACM, 2007, pp. 557–566.

[2] J. Kramer, "Whither software architecture? (keynote)," in *Proceedings of the 34th International Conference on Software Engineering (ICSE 2012)*, ser. ICSE 2012. IEEE Press, 2012, pp. 963–963.

[3] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software architecture: foundations, theory, and practice*. Wiley Publishing, 2009.

[4] G. Murphy, D. Notkin, and K. Sullivan, "Software reflexion models: bridging the gap between design and implementation," *IEEE Transactions on Software Engineering*, vol. 27, no. 4, pp. 364–380, Apr 2001.

[5] R. Krikhaar, "Reverse architecting approach for complex systems," in *Proceedings of the 13th International Conference on Software Maintenance (ICSM '97)*, Oct 1997, pp. 4–11.

[6] C. Riva, "Reverse architecting: An industrial experience report," in *Proceedings of the 7th Working Conference on Reverse Engineering (WCRE'00)*. IEEE Computer Society, 2000, pp. 42–.

[7] I. T. Bowman, R. C. Holt, and N. V. Brewster, "Linux as a case study: Its extracted software architecture," in *Proceedings of the 21st International Conference on Software Engineering (ICSE '99)*, 1999, pp. 555–563.

[8] K. Mens, A. Kellens, F. Pluquet, and R. Wuyts, "The intensional view environment." in *Proceedings of the 21st International Conference on Software Maintainance (ICSM '05)*, 2005, pp. 81–84.

[9] S. Demeyer, S. Tichelaar, and S. Ducasse, "FAMIX 2.1 - The FAMOOS Information Exchange Model," Univ. of Bern, Tech. Rep., 2001.

[10] B. Johnson and B. Shneiderman, "Tree-maps: A space-filling approach to the visualization of hierarchical information structures," in *Proceedings of the 2nd IEEE Conference on Visualization (VIS '91)*. IEEE Computer Society Press, 1991, pp. 284–291.

[11] J. J. Van Wijk and H. van de Wetering, "Cushion treemaps: Visualization of hierarchical information," in *Proceedings of the 1999 IEEE Symposium on Information Visualization (INFOVIS '99)*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 73–78.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson, 1994.