Generating Visualization-based Analysis Scenarios from Maintenance Task Descriptions

Salima Hassaine, Karim Dhambri, Houari Sahraoui, Pierre Poulin
{hassaisa|dhambrik|sahraouh|poulin}@iro.umontreal.ca

Abstract

Software visualization is an efficient and flexible tool to inspect and analyze software data at various levels of detail. However, software analysts typically do not have a sufficient background in visualization and cognitive science to select efficient representations and parameters without the help of visualization experts. To overcome this problem, we propose an approach to generate software analysis tasks that use visualization. To this end, we use taxonomies of low-level analytic tasks, high-level interactive tasks, and perceptual rules to design an assistant that proposes analysis scenarios.

1 Introduction

Software visualization offers powerful tools to foster a better understanding of software quality. It exploits the natural pattern recognition ability of the human brain and the knowledge of the expert to analyze data. It also shows new perspectives and raises new interrogations that might not easily be met otherwise. A number of visualization systems exist for complex software analysis [5, 7, 8, 3]. However, each of these systems requires that the user explicitly specifies the visualization parameters. Building effective visualization requires understanding some perception rules from cognitive sciences. Unfortunately, software specialists typically do not have the necessary background in visualization, and therefore they often seek assistance from visualization experts to help them display their data and use efficiently the available tools.

This paper presents an approach which enables the user to specify an analysis task in terms of software code and metrics, and data analysis techniques, and then transforms this specification into a strategy for interactive visualization. This strategy assists the user in carrying on his task using a specific visualization tool. The approach also generates for each task a mapping between metrics and graphical representations based on perceptual rules.

The rest of the paper is structured as follows. Section 2 presents an overview of the approach. Sections 3 and 4 describe the two models involved in our approach. Section 5 shows the transformation mechanisms between the two models. Finally, Section 6 discusses some conclusions and identifies future work directions.

2 Overview

We propose a task-driven approach for software visualization in a way that supports user-defined analysis goals. This approach encompasses the entire process from the analysis task description to the generation of a sequence of interactive actions supported by a specific software visualization tool.

Figure 1 illustrates our approach which consists of the following main components. **Analysis Task (AT) Model**: This model offers a language to rigorously describe in terms of user goals the scenario for an analysis task, based on a known analytic task taxonomy [1]. This model is independent from visualization. **Interactive Visual Task (IVT) Model**: This model describes the interactive visual tasks that a software visualization application should support to be compatible with our approach. **Analysis Task to Interactive Visual Task Transformation**: This mechanism transforms an analysis task described using the Analysis Task Model and a tool specification that instantiates the Interactive Visual Task Model into an interactive visual task scenario. It is based on a set of perceptual rules. **Transformation Session**: A transformation session is specific to a particular tool. It instantiates the above mechanism, taking into account the tool specification.

3 Analysis Task Model

This section describes the Analysis Task model. We start by presenting the basic operators that can be per-

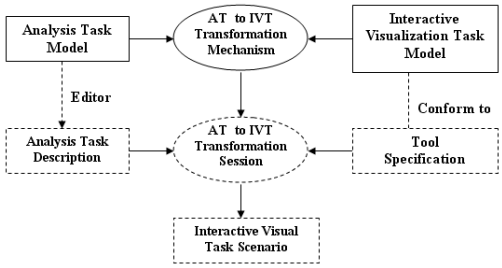

Figure 1. Overview of the transformation process.

formed to explore the software code. Then we give the analysis description model in terms of goal-oriented modeling formalism.

In [1], Amar *et al.* propose a low-level and domain-independent taxonomy of operators that a user might perform on a data set. Building on this taxonomy, we have derived a set of operators for the specific purpose of code-based data inspection. More specifically, we characterized each operator by the parameters required for its execution and the scope of its application. An operator is then described as a tuple $< operation, parameters >$ where *operation* is an action to perform on the data using the specified *parameters*. Parameters include code entity or set of entities (x or X), attributes (a), conditions (c), text labels (l), relationships (r), and properties (p) such as class name, class code, class position with respect to the package architecture, etc. Each operator has a scope: global or local. A global operator is applied to a large set of code entities (classes and interfaces), while a local operator is applied on a reduced subset of entities. A sub-set of these operators are shown in Table 1.

Up-to-now, we have defined a set of basic operators that can be used in an analysis task. The next step is to define a model that describes a complete analysis task. To this end, we defined a goal-driven model using the principles of [2]. According to this model, an analysis task consists of a goal, i.e., the purpose of the analysis, that can be refined into sub-goals. Each sub-goal is described by a list of operators defined in Table 1 and eventually control structures (conditionals and iterators).

4 Interactive Visual Task Model

This model is based on the *Task by Data Type Taxonomy* proposed by Shneiderman [9]. This taxonomy presents seven high-level interactive tasks that an information visualization application should support, such as *overview, zoom, filter, details-on-demand,* etc. These interactive tasks are task-domain information actions that users might want to perform in a visual environment.

Inspired by this taxonomy, we propose a model to describe the interactive tasks (call them interactors) that a visualization tool should support. These interactors are used to perform an analysis task (as described in Section 3) in a visual environment. Figure 2 illustrates our proposed Interactive Task (IVT) Model. According to this model, a visual environment is represented as a *scene*. A scene contains a collection of graphical representations used for displaying code entities. The scene also includes a mapping module that describes how the properties of these entities are mapped onto the visual features of their representation. We also define two types of interactors: local and global. Local interactors are applied on specific graphical representations, while the global interactors are used on the whole scene. The visual features of the graphical representations may change according to the interactor applied on them. Therefore, a history module is important to keep track of the interesting actions and the different states of the scene to support an "undo" functionality.

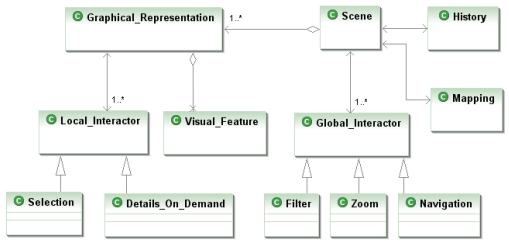

Figure 2. Interactive Visual Task Model.

The various interactors in the model are: (1) **Overview** to gain an overview of the entire collection of code entities, (2) **Zoom** to zoom in on entities of interest, (3) **Filter** to put emphasis on a sub-collection having certain properties, and filter out uninteresting entities, (4) **Details-on-demand** to select a particular entity and get details when needed, (5) **Selection** to select un subset of entities of the collection, and (6) **Navigation** to move around within the collection.

To be usable with our approach, a visualization tool must instantiate the IVT model. As an illustration, we used the software visualization tool VERSO [4].

VERSO uses $2\frac{1}{2}$D representations for software visualization, *i.e.,* 3D entities placed on a 2D space as shown in Figure 3. The visual metaphor of VERSO is constituted of two graphical elements: layout managers and graphical representations. The layout man-

Operator	Scope	Description
Locate$\langle X, c_1, ..., c_n \rangle$	global	Find code entities in X that satisfies conditions $c_1, ..., c_n$.
Select$\langle X, c_1, ..., c_n \rangle$	global	Find and memorize the subset of X that satisfies conditions $c_1, ..., c_n$.
Retrieve Value$\langle x, a \rangle$	local	Find the value of attribute a for the entity x.
Find Extremum$\langle X, a \rangle$	global	Find entities in X having an extremely high value for attribute a.
Characterize Distribution$\langle X, a \rangle$	global	Characterize the value distribution of attribute a over X.
Cluster$\langle X, a_1, ..., a_n \rangle$	global	Find clusters of similar values for attributes $a_1, ..., a_n$ in X.
Inspect$\langle x, p_1, ..., p_n \rangle$	local	Obtain detailed information about properties $p_1, ..., p_n$ for an entity x.
Decide$\langle c, op \rangle$	N/A	The user decides to perform operator op conditionally to c.

Table 1. Analysis operators.

ager arranges all the elements of a software system on a plane, according to their architectural properties. Software entities are graphically represented as 3D objects. The source code metrics are mapped to the visual features of the representation of the software entity. Each representation (a 3D box) has five visual features: the **color** that varies from blue to red, the **height** that varies from small to high, the **twist**,3D box orientation, that varies from straight to highly twisted (an angle of 90 degree), the **analog clock** texture, for which the metric value is mapped to the orientation of the clock pointer, between angles 0 and 330 degrees, and the **square** texture, for which the metric value is mapped to the number of squares (this texture is more effective for discrete data having low values).

VERSO offers the features defined in the IVT model. The user can apply **Zoom-In** and **Zoom-Out** on the

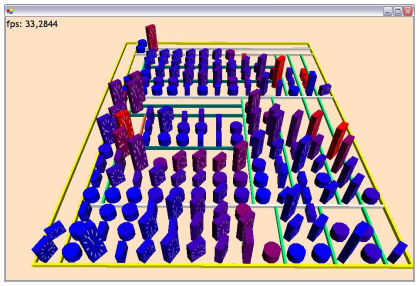

Figure 3. Overview of software system *EMMA* in VERSO.

scene using the mouse wheel. VERSO offers two types of **filters**. The statistic filter deals with the distribution of the metric values using the box plots. The relationship filter, when applied on a particular code entity, related code entities retain their original colors, while for all other unrelated code entities, the saturation of their color is reduced. It possible to obtain, for a particular entity, **details on demand**. These include metric values or characteristics of the source code. Finally, it is possible to **select** already visited code entities or those that we may want to come back to by changing the color of the top of their 3D box representation.

5 AT to IVT Transformation

For a given transformation session, the user provides an analysis task description as input. The transformation mechanism first analyzes the input to extract the attribute properties needed to correctly apply our guidelines. Depending on the scope of the operators and their measurement scale, the corresponding attributes will then be mapped onto appropriate visual features. Finally, a tool-specific scenario is generated using the operators from the task description, taking into account the attribute mapping and the various interactors necessary to achieve the task.

Several mappings are possible between attributes and visual features, each having a cost (efficiency). Selecting the most appropriate mapping among all alternatives for a given situation usually requires considerable knowledge of visual perception principles, and of the data itself. In this context we address the mapping selection as a valued-constraint satisfaction problem. Formally speaking, we define a 3-tuple $\langle X, D, C \rangle$ where $X = \{X_1, X_2, ..., X_n\}$ is a finite set of variables (metrics to be visualize), D is the domain of the variables in X, defined as the available graphical features, and C is a finite set of soft constraints. A soft constraint $f(variable, scope, visualfeature) \in C$ is a function that evaluates the effectiveness of a visual feature for a given metric considering the scope of the operator to apply to this metric (see Table 1). To define the values of f, we use the mapping costs proposed in [6]. In this work a cost is associated to each combination attribute scale-feature. For example, a quantitative attribute mapped to the length costs 1 units, whereas, an ordinal attribute costs 7 for the same feature. We extended this cost function by integrating the scope of the operation. For example, when a texture is associated to an ordinal attribute, the cost is 4 for an operation that requires a

Operator	Interactor	Generated VERSO Description
Locate$\langle X, c_1, ..., c_n \rangle$	Overview	Among X, locate classes whose $c_1, ..., c_n$
Select$\langle X, c_1, ..., c_n \rangle$	Overview	Among X, select classes whose $c_1, ..., c_n$
Retrieve Value$\langle x, a \rangle$	Zoom Details-on-demand	Apply the `Zoom-In` if necessary `Display list of properties` for class x Read the value for metric a
Find Extremum$\langle X, a \rangle$	Overview Filter Selection	Apply the `Zoom-Out` if necessary Apply the statistical filter on a for X Using the `Iterator`, `Select` the classes as long as the metric a is judged high enough Apply the `Statistical Filter` on a for X Remove the Statistical Filter
Characterize Distribution$\langle X, a \rangle$	Overview	Apply the `Zoom-Out` if necessary Apply the `Statistical Filter` on a for X
Cluster$\langle X, a_1, ..., a_n \rangle$	Overview Filter Selection	Apply the `Zoom-Out` if necessary Find clusters in X sharing similar values for attributes $a_1, ..., a_n$ `Select` classes in every cluster
Correlate$\langle X, a_1, ..., a_n \rangle$	Overview	Apply the `Zoom-Out` if necessary Find if $a_1, ..., a_n$ are correlated in X
Inspect$\langle x, p \rangle$	Zoom Details-on-demand	Apply the `Zoom-In` if necessary `Display source code` of x Inspect p
Decide$\langle c, op \rangle$	N/A	if c, generated description of operator op

Table 2. Operator-to-interactor mappings.

local scope. It is 8, when the scope is global.

After mapping the attributes, the next step is to transform the operators of Table 1. Table 2 presents the task mapping (operators to interactors) and the corresponding scenario fragment in VERSO. The mapping is one-to-many because, an operator of the task is transformed into a sequence of interactors in the IVT model and by extension in VERSO.

6 Conclusion

In this paper, we proposed an approach to transform an analysis task description into an interactive visual task for a visualization tool. We defined a language to let the analyst formally specify a maintenance task in terms of software metrics and data analysis operators. We have tested our approach on a number of anomaly detection tasks on large software programs. Our experience showed that analysts do not need any visualization-specific knowledge to perform the tasks. Up-to-now, the considered tasks have involved a small number of metrics (usually 4 or 5). In the near future, we plan to use our approach to generate scenarios for tasks that involve a larger number of code metrics and characteristics. With this perspective, an exhaustive evaluation of all possible mappings is too time-consuming. To solve that problem, we are developing a CSP solver based on a metaheuristic search.

Another future research direction is to refine our mapping constraint base, taking into consideration the distribution of the metrics data to be visualized. Indeed, in addition to the discrete/continuous aspect and the global/local one, the distribution brings valuable information to chose the best graphical attribute for a specific metric. Finally, we plan to experiment with other visualization tools to generalize the applicability of our approach.

References

[1] R. Amar, J. Eagan, and J. Stasko. Low-level components of analytic activity in information visualization. In *INFOVIS*, 2005.

[2] S. Card, T. Moran, and A. Newell. *The psychology of human-computer interaction.* Lawrence Erlbaum Associates, 1983.

[3] S. Ducasse and M. Lanza. The class blueprint: Visually supporting the understanding of classes. *IEEE Trans. Softw. Eng.*, 2005.

[4] G. Langelier, H. Sahraoui, and P. Poulin. Visualization-based analysis of quality for large-scale software systems. In *ASE*, 2005.

[5] G. Lommerse, F. Nossin, L. Voinea, and A. Telea. The visual code navigator: An interactive toolset for source code investigation. In *Proc. IEEE Symp. Information Visualization*, 2005.

[6] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 1986.

[7] A. Marcus, L. Feng, and J. Maletic. 3d representations for software visualization. In *SOFTVIS*, 2003.

[8] S. P. Reiss and M. Renieris. Chapter 11: The bloom software visualization system. In *Software Visualization: From Theory to Practice*. 2003.

[9] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. *IEEE Symp. on Visual Languages*, 1996.