Visualizing Software Models

Summary of dissertation for the degree of Master in Information Systems and Computer Engineering

Miguel Gil

1Instituto Superior Técnico, Universidade Técnica de Lisboa, Portugal
miguel.gil@tecnico.ulisboa.com

ABSTRACT

Software development is a complex and difficult task that requires the investment of significant resources and carries major risks. Model-driven development approaches can help these tasks, but they generate thousands of objects and relationships which are difficult to analyse without losing the context, and mostly by proprietary tools. We propose a visualizer for the data model of these applications/information systems that makes use of visualization techniques with the proper filters to allow a user to navigate between objects and abstraction levels without losing the context and/or focus. We used the Design Science Research Methodology to conduct our research. This proposal was demonstrated in a real life company that uses a tool with model-driven-development concepts to develop Information Systems and it helped to identify the objects with higher importance in the evaluated applications. We evaluated it with feedback gathered from interviews and results from the application of the visualizer in real applications. The artifact’s usability was also measured through user tests and it has shown that it has a nice usability.

Keywords

MDD, visualization, models, rapid development tools, visualization techniques, abstraction

1. INTRODUCTION

Software development is a complex and difficult task that requires the investment of significant resources and carries major risks. According to its proponents, model-driven software development approaches are improving the way we build software [1]. The software developed through these approaches generates a large amount of modules and database entries, with hundreds of tables which have around 100.000 different attributes.

The field of visualization is getting mature and the visualization of data models can make possible to researchers, analysts and engineers to get an insight of this data model in an effective and efficient way [2].

One way to make the analysis of such complex model instances is to go through the visualization of these data models in different abstraction levels with the capacity of navigating through them.

Our proposal consists in a set of state-of-the-art visual representations, like graphics, combined in a tool for visualization that will allow the user to navigate through data models raising or lowering the level of abstraction according to their needs. This proposal was tested through its addition to an existing Rapid Development tool, Genio, and it helped to identify the objects with higher importance in the evaluated applications and gave the users a more simple way to analyse the dependencies in the application’s definitions.

We tested with several users, and evaluated the chosen visualization techniques through observation and interviews/surveys and performed a critic analysis between them. In the evaluation, we focused on the data model analysis and reasoning in order to guarantee that the visualizations used are, in fact, helpful to visualize the relationships between objects in the data models of these applications generated automatically through the definition of the models and from the MDD approach. We also focused in the usability of our artefact.

1.1. Problem

The growing adoption of Model-Driven Development (MDD) [3] [4] methods, where models are the primary artifact of the development process, is shifting the focus of existing software engineering methods from code to models. In this context, the correctness of such models plays an important role in the quality of the final software system. Unfortunately, there are currently few tools that support the verification of software models [5].

MDD of applications brings many advantages concerning the development of highly complex information systems, which can be seen in development time, software quality, team’s knowledge management, maintenance and evolution of the system, among others.
However, the state-of-the-art of models with the capacity of automatically generating systems are a mixture of technological concepts that are better understood by programmers.

Effective performance of many software engineering tasks requires knowledge of how the system works. Gaining the desired knowledge by studying or statically analysing the source code can be difficult [6].

Software artifacts in industrial applications can consist of thousands of code files, making difficult to manage and analyse the system’s complexity. Moreover, in complex systems, the number of models’ instances that describe, say, a full ERP, have around 100.000 different attributes.

It is important that analysts and developers can extract the most relevant objects of a system from complex models and navigate through the model.

There are also more challenges related to the use of MDD [7]:

- Understanding and managing the interrelations among the multiple representations of software development artifacts representing different views or levels of abstraction of the same concept;
- Comparing and merging different versions of models, e.g., visualizing the differences in a usable way.

2. RELATED WORK

In this chapter we present a literary review of the topics related to the problem. We start by introducing the most important concepts referred in this document, followed by the analysis of the tools that make used of these concepts, with more focus on the way they visually represent their objects.

Subsequently, we’ll introduce a taxonomy of visualization techniques from which we have chosen the right ones for our proposal.

2.1. MDE/MDD/MDA

Model driven engineering (MDE) combines process and analysis with architecture [6].

In MDE, models are the prime artifacts, and developing high-quality systems depends on developing high-quality models and performing transformations that preserve quality or even improve it [7].

In Figure 2, we present our view of the MDE scope.

As seen in Figure 2, MDE can be viewed as a superset of Model Driven Development (MDD), which can be a superset of Model Driven Architecture (MDA).

MDD can be mistaken for MDE because the principles are the same, but MDE goes beyond of the pure development activities and encompasses other model-based tasks of a complete software engineering process (e.g. the model-based evolution of the system).

MDD is simply the notion that we can construct a model of a system that we can then transform into the real thing [8].

MDA is a framework for software development, defined by the OMG. Key to MDA is the importance of models in the software development process [9]. Within MDA the software development process is driven by the activity of modelling the software system. It’s the particular vision of MDD to OMG.
Central to MDD (and Model Driven Architectures) is the notion of creating different models at different levels of abstraction and then linking them together to form an implementation. Some of these models will exist independent of software platforms, while others will be specific to particular platforms.

The primary goal of MDD is to raise the level of abstraction at which developers operate and, in doing so, reduce both the amount of developer effort and the complexity of the software artifacts that the developers use [10].

2.2. Rapid Development tools

There are many tools that let us to make use of concepts like MDD. These tools allow us to build software through the automatic generation of the code from models. We’ll call these tools rapid development tools since they allow the rapid development of applications. In this section we present a critical analysis of the visualizations allowed by some of these tools, more specifically, Artech’s Genexus, Quidgest’s Genio and the Outsystems Platform.

2.2.1. Genexus

Genexus\(^1\) is an intelligent tool developed by Artech aimed at assisting analysts and users throughout the lifecycle of applications [11].

The design and prototype are done and tested on a Windows environment. Once the prototype is fully approved by users, the database and the application programs for the production environment are generated and/or maintained in a fully automatic way.

Genexus have some views that allow the user to see the entities of an application, as shown in fig. 2:

![Fig. 2 - Examples of Genexus Views](image_url)

1. Business Process
2. Form
3. Relationships between tables
4. Table

The problem we see with these visualizations are the lack of navigation and interactivity between them. For example, for a user be able to see the relations between the tables in an application (3) he has to create a diagram and then drag and drop the tables from the list of tables to the diagram. It is difficult to navigate through these views without losing the context or the focus. And there is no view that allow the user to have a general overview of the application with all the entities.

2.2.2. Genio

Genio\(^2\) is an integrated development platform, created by Quidgest, to manage dynamic metamodels of functional specifications and automatic code generation.

*It is a platform for rapid development of comprehensive information systems, combining model-based development with automatic code generation in different programming languages such as MVC, PhoneGap, Web services, HTML 5 or C# [12].*

---

2. [www.quidgest.pt/q_genioPT.asp](http://www.quidgest.pt/q_genioPT.asp)
Genio doesn’t possess a visual way (per say) to analyse the data and navigate through it. The most similar to representations of the models created by Genio is a view that can show the relations between some entities of an application, as shown in fig. 3:

This visualization, by itself, does not give the user much information about the application and it should the combined with other views in order to be more useful.

2.2.3. Outsystems Platform

The OutSystems Platform\(^3\) is a high-productivity solution for the agile development and management of custom enterprise web and mobile applications [13]. It allows development at the business logic level, with the platform generating the conventional code needed for deployment on the available technology.

Like Genexus, the Outsystems Platform also has some views that can help the user understand the application, as shown in fig. 4:

![Fig. 4 - Examples of Outsystems views (1 - Business Processes, 2 - Tables and their relationships)](image)

Once more, this visualizations are difficult to navigate and interact and they don’t give a general overview of an application.

2.3. Software visualization tools

Software visualization [14] refers to the visualization of information of and related to software systems and their development process by means of static, interactive or animated visual representations of their structure, execution, behavior and evolution.

---

\(^3\) [www.outsystems.com/platform/](http://www.outsystems.com/platform/)
Most visualization tools that we found are more concerned in showing the data that is in an application, instead of showing the relationships between the models that describe the definitions of the application: the data model. Other examples are tools with the focus on the application code.

We found some visualization tools that allow the user to see the relationships between the tables in a relational database and that allow to identify some dependencies, but these visualizations don’t seem to allow navigation between them nor they show implicit relationships between elements. It’s like they’re made to a user that already knows what he wants to see, instead of discovering through navigation.

Some of these tools are Tioga and Tioga-2 [15], Polaris [16] and Shrimp views [17].

2.4. Visualization techniques

There were some attempts at constituting taxonomies to differentiate the kind of visualization. We have chosen the taxonomy constructed by Shneiderman [18].

In this taxonomy are defined seven data types:

- **1D** - Linear data arranged in a sequential manner, e.g., alphabetical list of names, source code;
- **2D** - Planar or map data, e.g., photographs, maps, newspapers layouts;
- **3D** - Data with volume and with relationships with each other, e.g., molecules, buildings;
- **Temporal** - Data with start time, finish time with or without time overlaps, e.g., medical records, video editing;
- **Multi-dimensional** - Data that has multiples attributes that translate into points in an n-dimensional space, e.g., records in relational and statistical databases;
- **Tree/Hierarchical** - Items that are linked through hierarchy in a tree structure, e.g., computer directories, business organizations;
- **Network** - Items linked through a graph structure, e.g., World Wide Web.

From these types, we’ll chose the most adequate to the kind of data that we want to visualize.

3. PROPOSAL

We identified some criteria for good visualizations [19] and matched them with the type of information that we wanted to visualize. What we want to visualize are different levels of abstraction of the components of an application developed with a rapid development tool, i.e., its objects. These objects can be menus, tables and forms and the solution must use graphical representations, i.e., visualization techniques applied in this context to show the relationships between those objects by various levels of abstraction.

We also want to be able to see the most relevant objects of the application with few or without any interaction. The most relevant objects will be the ones with more dependencies, i.e., the ones that have relationships with more objects.

It is also important that we can visualize the evolution of the application over time in order to check and monitor the changes of the model and to identify the objects with more changes.

So, the criteria that we have identified for our solution are:

- **Ease of use** – the visualizer must be user friendly in order to make the users to want to use it;
- **Portability** – we want a visualizer that can be used with different rapid development tools;
- **Quantity of information that can be easily viewed** – we want to be able to show a great amount of data, so we will need to create filters according to the quantity of data to show in each moment;
- **Simplicity** – The information must be seen in a simple way in order to be easily understood. Also, tasks performed with the proposed solution must be done in the least amount of time and with the least effort possible;
- **Navigation and interaction** – The visualizer must provide a way to navigate through different levels of abstraction;
- **Context keeping** – In some situation it can be useful for the users to keep the context between visualizations;
- **Filters** – The visualizer must allow the user to filter the data according to their needs.

The first problem addressed is the management and analysis of the systems complexity. It is intended to extract from complex models the most relevant objects of the system without any additional information or interaction.
The second problem is concerned with the visualization of the models and the navigation between models inside and out of the same abstraction layer. Bi-dimensional interfaces that allow to visualize and manipulate data can also be used to improve the perception of the models.

**In order to try to solve these problems, we propose the development of a model’s visualizer** that will allow to:

- Facilitate the complex model’s visualization through 2D visualization techniques;
- Extract from complex models the most relevant objects of the system without any additional information or interaction;
- Visualize the models and the navigation between models inside and out of the same abstraction layer;

We proposed the following architecture for the visualizer:

![Diagram of the proposal](image)

**Fig. 5– Architecture of the proposal**

The visualizer decomposes the application in its more important objects. These objects will be present in the **Objects Decomposition** component in a way that the user can identify the most important ones without any or little additional interaction. For example, if a table has many connections with other objects, then, if there is an error with that table, all the related ones will be spreading that error.

Its **Time** component will allow users to see the changes to the objects through time, giving them an insight of how the application evolved since it began being developed. The user will know which objects were created, updated and deleted from the application and when these operations occurred.

The **Relationships** component will help visualizing the dependencies of objects in the data model of an application, giving the possibility of navigating through it discovering other connections.

The **Grouping** component will help to identify groups of models that relate in some context, i.e., how they are grouped relatively to some other components.

This architecture makes use of different visualization algorithms, each one with the characteristics needed regarding the kind of information that we want to see.

This stratified architecture also allows users to navigate freely through the abstraction levels and through focus.

4. **DEMONSTRATION**

Our demonstration was made in Quidgest⁴, which is a consultancy and software engineering company, with headquarters in Lisbon, Portugal, implementing a disruptive strategy for the IT sector, based on the creation and improvement of automatic code generation processes. Genio, Quidgest’s platform for automatic code generation, enables deployment in a wide range of platforms (as mentioned in previous sections) and seamlessly integrates with a range of existing technologies.

We tested our developed tool with several real applications from Quidgest in order to realize if our proposed tool performed well with different applications. This was extremely important because there are some applications bigger than other and our tool must perform well disregarding the size of the application.

The next figure shows the four components of our artifact – the models’ visualizer – applied to a real application developed by Quidgest:

---

⁴ [www.quidgest.pt/](http://www.quidgest.pt/)
As seen shown in fig. 6, our tool makes use of different visualization techniques to represent each view described in the previous section. It’s possible to navigate between those views in order to explore the application’s objects.

5. EVALUATION

In order to evaluate the feasibility of our proposal [20] and demonstration we performed tests of a quantitative nature, namely usability tests, with 10 users. These tests were made at the users’ workplace, as it is the environment where they would be using the tool in normal circumstances. All the users performed the tests with the same data, i.e., using the visualizer with the same real application with real data, as we aimed to simulate tasks that they would perform with the proposed tool. The fact that all the users performed the tests with the same application allowed us to measure some usability metrics like: time taken to complete a task, number of errors and number of mouse clicks.

After the tests conclusion, we summarized the metrics measured. The following table resumes the observed data and, for each task, we calculated the average, standard deviation, maximum and minimum values observed for each metric (clicks, errors and time taken):

<table>
<thead>
<tr>
<th>Task</th>
<th>Mouse clicks</th>
<th></th>
<th></th>
<th></th>
<th>Errors</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Time (seconds)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\bar{x}$</td>
<td>$\sigma$</td>
<td>Max</td>
<td>Min</td>
<td>$\bar{x}$</td>
<td>$\sigma$</td>
<td>Max</td>
<td>Min</td>
<td>$\bar{x}$</td>
<td>$\sigma$</td>
<td>Max</td>
<td>Min</td>
<td></td>
</tr>
<tr>
<td>Task 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0,1</td>
<td>0,3</td>
<td>1</td>
<td>0</td>
<td>1,3</td>
<td>0,64</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Task 2</td>
<td>10,1</td>
<td>1,87</td>
<td>13</td>
<td>7</td>
<td>1,9</td>
<td>0,94</td>
<td>3</td>
<td>0</td>
<td>72,9</td>
<td>16,91</td>
<td>96</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Task 3</td>
<td>5,2</td>
<td>1,17</td>
<td>7</td>
<td>4</td>
<td>0,9</td>
<td>0,83</td>
<td>2</td>
<td>0</td>
<td>36,7</td>
<td>6,97</td>
<td>50</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Task 4</td>
<td>3,1</td>
<td>1,51</td>
<td>6</td>
<td>2</td>
<td>0,5</td>
<td>0,67</td>
<td>2</td>
<td>0</td>
<td>31,3</td>
<td>5,25</td>
<td>40</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>Task 5</td>
<td>6,6</td>
<td>1,91</td>
<td>10</td>
<td>3</td>
<td>0,4</td>
<td>0,49</td>
<td>1</td>
<td>0</td>
<td>30,7</td>
<td>8,26</td>
<td>45</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 - Usability tests results
As it can be seen in table 1, after task 1, the time spent to perform the tasks and the number of errors decrease in average, along the test. So, we believe that when the users star using the tool more often they will be using it more efficiently. Appendix C has the metrics measures for each task.

After the tests, we asked the users to answer a small survey in order to gives some feedback on how they felt using the visualizer.

The survey had a combination of 10 standard questions based on the work of John Broke – The System Usability Scale [21]. This had the purpose of evaluating the usability of the visualizer, as well as giving us some insight of its utility. Table 2 shows the results of the SUS evaluation:

<table>
<thead>
<tr>
<th>SUS Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bar{x} )</td>
</tr>
<tr>
<td>( \sigma )</td>
</tr>
<tr>
<td>Max</td>
</tr>
<tr>
<td>Min</td>
</tr>
</tbody>
</table>

Table 2 - System Usability Scale score

As seen in the previous table, the average score obtained was 81.75 which is a good indicator of the visualizer’s usability. Even the users that had more difficulty performing some tasks, gave the impression that the difficulties were justifiable by being the first time they were seeing the visualizer. The standard deviation of the SUS score was 7.67, which isn’t much, showing that there wasn’t a great difference between the values.

6. CONCLUSION

Our research is expected to contribute to the implementation of several visualization technique prototypes that can be applied to the data model of an application that is developed through a tool that uses MDD concepts.

Those prototypes are navigable, interactive and compose a visualizer that allows the user to navigate through the application’s data model, between different levels of abstraction and without losing the context and/or focus. We tested our artefact with 10 users and the results have shown that our prototype had a nice usability and can actually help.

We expect that this visualizer will improve the way the dependencies between objects in a data model are seen and evaluated in order to identify the components that implement a given part of the application and that can be reused in other applications that share that same behaviour. We also expect the use of our artifact can help developers and analysts in analysing how the system really works and how they can improve it or detect errors.

References


