Towards Visualization Support for the Eclipse Modeling Framework

R. Ian Bull and Margaret-Anne Storey
Department of Computer Science
University of Victoria
Victoria, British Columbia
{irbull,mstorey}@cs.uvic.ca

1. INTRODUCTION

The Eclipse Modeling Framework [1] is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification, EMF produces a set of Java Classes to represent the model, as well as a set of adapter classes that enable viewing and editing. EMF can efficiently create a working prototype for the data model along with a set of user interfaces.

In our research, we propose extending the viewers and editors EMF produces by providing the designer with the tools necessary to create more customized views for their data through Model Driven Visualization. This paper motivates the need for Model Driven Visualization and describes the process in detail. Section 2 reviews the related literature. Section 3 motivates the problem by describing a scenario actually witnessed by students and researchers at IBM Canada. Section 4 describes our approach to this problem and finally Section 5 presents some concluding remarks.

2. RELATED WORK

Visualizations are most effective when the views have been customized for specific users and their tasks. Few visualization tools provide adequate support for customization. Widget sets such as SWT and JFace [7] are highly versatile but requires strong programming skills even to create simple prototypes. Toolkits such as SHriMP [2], Bloom [10] and GSee [5] provide scripting languages in order to customize the views. While these scripting languages are an improvement, the tools still lack the ability to describe the views graphically through models. Consens and Mendelzon [4, 8] proposed a method of using Visual Query Languages to visualize and query software structures. While these tools have had some success in visualizing static structures in a software system, they have not been generalized for creating views in other domains.

3. MOTIVATION

In the Spring 2003 we conducted a series of user studies in order to test the effectiveness of Zoomable User Interfaces [6] compared to traditional tabbed interfaces for navigating business process diagrams. While the results of this study were inconclusive the experiment itself resulted in some interesting insights into the importance of having a lightweight means of customizing views. At IBM Canada in Toronto Ontario, both the first author and members of IBM User Centred Design team planned the experiment, designed business models and performed the study. Concurrently, a group of researchers at the University of Victoria, in Victoria British Columbia, customized the tools for use within the business process domain [9]. As the study was performed, initial feedback suggested that additional customization was required. Users suggested that several aspects of the tool needed changing. This included removing entities that were deemed unimportant for the tasks business modelers actually perform, introducing new abstract relationships which could be derived from the existing data, changing the interaction style and modifying the visual properties such as colour and icons.

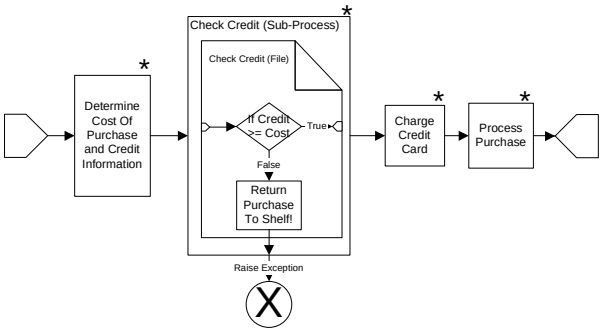

Figure 1: Business Process Flow

While none of these changes were conceptually challenging, it required constant communication between the experimenters and tool designers. Due to the geographical constraints, unclear requirements and other responsibilities each one of these requests required several days to complete. Since each of these changes are simply a transformation of the underlying model or attributes describing how the data is visualized, we propose tool support to give the experimenters the ability to make the changes directly on the model, immediately updating the prototype.

Figure 1 shows an example of a business process diagram for a purchase order. Each of the 4 boxes marked with an asterisk represent a sub-process. If the sub-process resides locally then it will be described in its own XML file. The second sub-process (Check Credit) has been expanded to show the details of how it works. Figure 2 shows the model which describe business processes in general. When the visualization tool was first designed, the `file` entities were explicitly displayed. Initial feedback from the users suggested that these `file` entities cluttered the diagram and were not required. Removing these entities for visualization purposes, so that the sub-processes directly contained further sub-processes required a change in the way the data was actually being displayed. Whenever a `file` token was reached, it had to be omitted. The

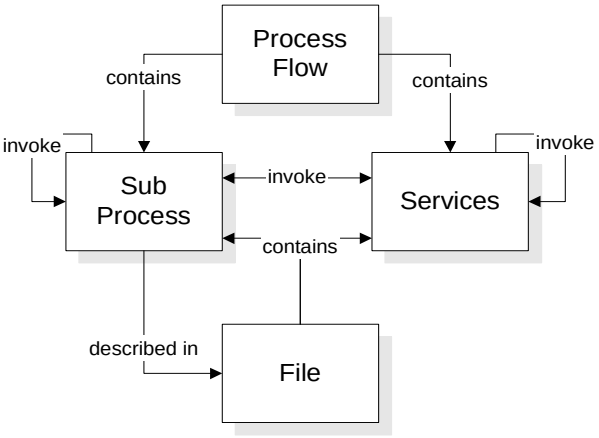

Figure 2: Business Process Model

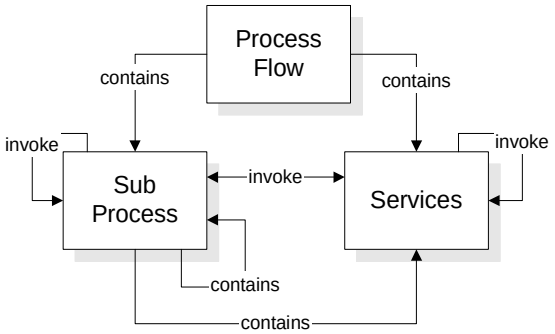

Figure 3: Modified Business Process Model

`file` tokens are still important for other process operations (such as name spaces and caching information), but for navigation purposes they hindered the users. This type of data traversal operation is common when creating visualizations and is almost always described programmatically.

4. PROPOSED WORK

In order to address the problem motivated above, we propose an integrated set of tools to help rapidly create visualizations for a given dataset. These tools could be incorporated into the Eclipse Modeling Framework so modelers would be able to model their visualizations as well as their data. The integrated tools will include a model extractor, a view recommender, a view modelling environment and code generators.

- **Schema Extractor:** Structured data always contains a model or schema, however this model is often implicit within the data itself. The schema extractor will extract the concrete schema from the data. A concrete or "as-is" schema is the model which shows only the types, relationships and constraints which actually exist in the instance data. In order to generate the implicit schemas we assume that the entities and relationships in the instance data are typed. In comparison, an explicitly specified schema shows the types, relationships and occasionally constraints which may, or are allowed to exist. By using the extracted schema the designer can create views tailored to the specific characteristics of the data they are working with.

- **View Recommender:** Often data models are highly complex and in order to create effective visualizations the designers must familiarize themselves with the structure of the data and then they can suggest the appropriate visualizations. For example, it does not make sense to use a tree style visualization if the data is not hierarchical. The view recommender searches for known patterns within both the extracted schema and the instance data and recommends views which may help to facilitate effective views of the data.

- **View Model Manipulation:** The view model manipulation stage presents the interaction designer with a graphical representation of the model, allowing them to directly design views. There will be several built-in operations for abstraction, information hiding, edge creation etc. The model will be manipulated through a process known as *Model Driven Visualization*. The list of operations which the interaction designer may perform is still being researched.

- **View Generation:** The view generation stage creates graphical views for the data by applying transformations the data as specified in the model manipulation stage, in order to view the data. The views will use the AVE (**A**dvanced **V**iews for **E**clipse) [3] or JFace Views. The AVE consists of template code, as well as a formal specification for several common information visualization techniques such as zoomable nested and flat graphs. Other views could also be created and added to the repository.

Model Driven Visualization would have helped us, as experimenters, while conducting the user study at the IBM Labs in Markham Ontario since we would have been able to modify the prototype based on user feedback. We could have imported the data directly into this toolkit, extracted the "as-build" schema and run the view recommender to find patterns (such as trees and other constraints) within the instance data. We could then have directly manipulate the model, filtering the `file` entities while maintaining all the others. The resulting schema is displayed in Figure 3. Finally we could generate a new zoomable nested graph to visualize the process diagrams, this time without the `file` entities.

5. CONCLUSION

EMF successfully creates frameworks from which programmers can begin developing applications. By adding visualization support to EMF, developers will be able to create more customized interactions for their applications through new and innovative visualization techniques. In order to add customized visualizations we proposed a process known as Model Driven Visualization.

6. REFERENCES

[1] Eclipse modeling framework. website. http://www.eclipse.org/emf.

[2] Casey Best, Margaret-Anne Storey, and Jeff Michaud. Designing a component-based framework for visualization in

software engineering and knowledge engineering. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 323–326, 2002.

[3] R. Ian Bull, Casey Best, and Margaret-Anne Storey. Advanced Widgets for Eclipse. In *Proceedings of 2nd workshop on Eclipse Technology Exchange*, pages 6–11, 2004.

[4] Mariano Consens, Alberto Mendelzon, and Arthur Ryman. Visualizing and Querying Software Structures. In *Proc of International Conference on Software Engineering*, 1992.

[5] Jean-Marie Favre. G^{see}: a Generic Software Exploration Environment. In *Proc of 9th International Workshop on Program Comprehension*, pages 233–244, Toronto, ON, May 2001.

[6] George W. Furnas and Benjamin B. Bederson. Space-Scaled Diagrams: Understanding Multiscale Interfaces. In *Proc. of Human Factors in Computing Systems CHI'95*, pages 234–241, Denver, Colorado, 1995.

[7] Erich Gamma and Kent Beck. *Contributing to Eclipse Principles, Patterns and Plug-Ins*, chapter 35, JFace – User Interface Frameworks, page 335. Addison Wesley, 2003.

[8] Alberto Mendelzon and Johannes Sametinger. Reverse Engineering by Visualizing and Querying. *Software – Concepts and Tools*, 16:170–182, 1995.

[9] Derek Rayside, Marin Litoiu, Margaret-Anne Storey, Casey Best, and Robert Lintern. Visualizing flow diagrams in websphere studio using shrimp views. *Information Systems Frontiers*, 5:161–174, 2003.

[10] Steven P. Reiss. An Overview of Bloom. In *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 2–5, 2001.