Preface

This review of the software suite MetaEdit+ is the base for the class project of using rules with MetaEdit+. This is an overview of the motivations for the use and creation of MetaEdit+, and also an overview of the internal structure of its components.

April 2008

Willer Travassos
260232775
CS 763, Winter Term 2008

Table of Contents

Understanding the underlying structure of MetaEdit+

Travassos, W.[1]

McGill University, Montreal QC H3A 1A1, Canada,
willer.travassos@mail.mcgill.ca,
WWW home page: http://www.cs.mcgill.ca

Abstract. This synthesis is about the process of learning and understanding the underlying structures, and implementation method used in the creation of the CASE/CAME software MetaEdit+. Through this paper there will also be a short talk on MetaEdit+'s general architecture, and how it works, and the meta-meta-model behind MetaEdit+. Once the review of MetaEdit+ is presented there will be also a conclusion section with the pros and cons related to its use.

1 Introduction: Motivation to create MetaEdit+

The software suite called MetaEdit+ is a tool that was developed in order to tone down and/or solve common problems that affected the development of software and systems. MetaEdit+ uses the CASE (Computer Aided Systems Engineering), and CAME (Computer Aided Method Engineering) methods to tackle these development problems, such as, those which affect productivity and product quality. An example of this type of problem is unfair time scheduling.

Also, the creators of MetaEdit+ were trying to cover problems that existed in other Domain Specific Modeling (DSM) solutions at the time of its creation. They tried to extra features that enhanced MetaEdit+'s capabilities. MetaEdit+ provides a programmer with a multi-user, multi-method/behavior, multi-tool environment that allows fast creation, easy management, integration, and re-use of models and meta-models.

2 Underlying Ideas

The main problems with DSM solutions that influenced the underlying ideas used in the development of MetaEdit+ were the following:

- Lack of model integration and consistency check, i.e., in order for different programmers to work on the creation of meta-models there should be a way of storing and maintaining changes to these meta-models
- Lack of multi-user access to created objects, and models, i.e., more than one programmer should be capable to work on the same meta-model/model at the same time

- Lack of varied object representation (be it graphical, tabular, formulary, and etc)
- To be a true meta-modeling environment, tools, systems and methods should be developed using the same meta-modeling language. MetaEdit+ uses Graph-Object-Property-Port-Role-Relationship (GOPPRR) to implement this idea
- Lack of reporting techniques and limited information retrieval, i.e., DSM solutions at the time did not have a simple tool that could easily gather data for reports to be shown to stakeholders or used in presentations

In order to provide all these features, the developers of MetaEdit+ incorporated several principles to its design. They are: Object orientation, Conceptual Modeling, Data, Representation, and Level Independence. Object orientation enables the re-use and interoperability between the MetaEdit+ components. Conceptual Modeling allows a programmer to add behavior representation of an object being modeled. Data Independence provides MetaEdit+ with a level of abstraction between the data stored in a computer, and the operations performed on such data, i.e., different tools can operate on data without having knowledge of how it is stored. Representation Independence, means that models, objects, relationships and etc exist independently of its possible representation (textual, tabular, and etc). Finally, Level Independence means that data used by MetaEdit+ is handled similarly, even if they are different from each other.

3 General Architecture

The architecture of MetaEdit+ can be divided in three parts: the MetaEngine, the Object Repository, and the Tools that are part of the software environment. The MetaEdit +'s architecture is centered around the MetaEngine, with the Tools acting as an interface on the client side, and with the Object Repository, which is on a server side, acting as storage for model and meta-model data.

The MetaEngine handles operation on the conceptual data stored in the Object Repository. These operations are commands generated by a programmer through the environment tools. This style of architecture was chosen to allow an easier integration of new tools in the MetaEdit+'s environment and a simplified manner of data/code manipulation.

The environment tools that form the user interface are divided in five categories that are related to their functionality. There are Management tools, which are responsible for basic software functionality. Editing tools which are used by a user, to create, manipulate, and delete models and its components. Editing tools provide different representations (graphical or form based) for their components, so that users can use the visual representation that is better suited to them.

Method Management tools are similar to the Editing tools, but instead of operating on objects in a model or meta-model, they work with the behaviors of these objects. Retrieving tools are used to retrieve data from the Object Repository, and Annotation tools are tools that provide commenting functionality similar to the one found in programming languages.

The Object Repository is a server that holds all data that is created with MetaEdit+. It also stores information about users, concurrent access to objects, visual representation of an object, and output specifications of created meta-models. All this information is stored as concepts in the GOPPRR language, which is meta-meta-model in MetaEdit+.

The name GOPPRR is an amalgamation of the types of objects that can be created within MetaEdit+, i.e., graphs, objects representing real world data, properties of objects, and their connection ports, object roles, and relationships.

The GOPPRR language uses object orientation so that a programmer can use its properties (e.g., polymorphism) to create complex objects and behaviors, in a faster manner, and simplicity in their re-use. Also, GOPPRR uses method integration to maintain consistency in the representation of objects and its components across different models that use them.

4 Conclusion

MetaEdit+ is a tool that tries to provide a Domain Specific Modeling environment which is easy and simple to create models and meta-models. It is simple and easy to install, and it supports all main platforms (namely, Windows, Linux, and Mac). It also provides great usability examples, and great support with its help library.

Unfortunately, there are not a lot of publications about MetaEdit+ in the academic world. It is closed source, so it can be hard to figure out how the software processes API messages, and how user actions are handled. Also, the API documentation is simplistic, and there is not a document explaining error messages generated from API calls. MetaEdit+ is very expensive, and those who might want to fully use its capabilities will have to spend 17000 euros plus a 20

References

1. Kelly, S., Lyytinen, K., and Rossi, M.: MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. Proceedings of CAiSE'96, 8th Intl. Conference on Advanced Information Systems Engineering, Lecture Notes in Computer Science 1080. Springer-Verlag, pp. 1-21, 1996
2. Juha-Pekka Tolvanen, Risto Pohjonen, Steven Kelly: SAdvanced Tooling for Domain-Specific Modeling: MetaEdit+. 7th OOPSLA workshop on Domain-Specific Modeling, pp. 1-8, 2007
3. MetaEdit+ 4.5 User's Manuals. MetaCase, March 2008. (http://www.metacase.com/support/45/manuals/index.html)