Flexible Deep Modeling with Melanee

Colin Atkinson and Ralph Gerbig[1]

Abstract: Multi-level modeling has gained increasing attention in recent years as the maturity of the supporting tools has grown. One of the most advanced tools for deep modeling is Melanee [Me16] which supports modeling through deep, multi-format, multi-notation user-defined languages. "Multi-format" refers to seamlessly editing a language in several formats in parallel (e.g. diagrammatic, textual and tabular) while "multi-notation" denotes the ability to mix notations arbitrarily (i.e. to show one part of a model in a general-purpose, UML-like, notation and one part in a domain-specific notation such as BPMN). The deep modeling component of the tool, underpinned by the orthogonal classification architecture and deep instantiation, allows models to contain as many classification levels as needed to concisely represent the domain in hand. This demonstration shows the capabilities of the Melanee tool in the context of defining a language to model the structure of an enterprise.

Keywords: Domain-specific Languages, Deep Modeling, Melanee, Tool Demonstration

1 Introduction

The market for domain-specific language engineering tools is strongly segmented by the set of features which modeling tools provide. For instance most tools focus exclusively on one explicit format (i.e. textual, diagrammatic, etc.) and are usually limited to support just two classification levels at a time (i.e. type level and instance level). An exception is Jetbrains MPS which specializes in supporting multi-format editing but lacks support for working with more than just two levels. However, no commercial tool today supports multi-format, multi-notation, domain modeling across multiple classification levels. This capability is advantageous in many domains, especially model execution [AGM15], enterprise language evolution [AGF15], cyber-physical systems modeling [At14], formal security policy modeling [At12] and many more.

In this demonstration, a prototype tool called Melanee which meets this need for multi-format, multi-notation domain-specific modeling will be presented. The implemented example revolves around the creation of a language capable of modeling the structure of an enterprise and demonstrates the following features: 1. multi-notation editing, 2. multi-format editing, 3. seamless multi-level editing, 4. deep, aspect-oriented concrete syntax definition 5. and deep context-sensitive languages.

2 Deep Modeling

Deep modeling is a form of multi-level modeling supporting the creation of models with arbitrary numbers of classification levels based on the Orthogonal Classification Architecture (OCA) [AK03] and deep instantiation [AK01]. An example of a how a deep enterprise model fits into the OCA is shown in Figure 1.

The OCA is implemented as two classification stacks aligned orthogonally to one another. One of these is the linguistic stack, labeled L_2 to L_0, and the other is the ontological stack,

[1] University of Mannheim, Chair of Software Engineering, 68159 Mannheim, {atkinson, gerbig}@informatik.uni-mannheim.de

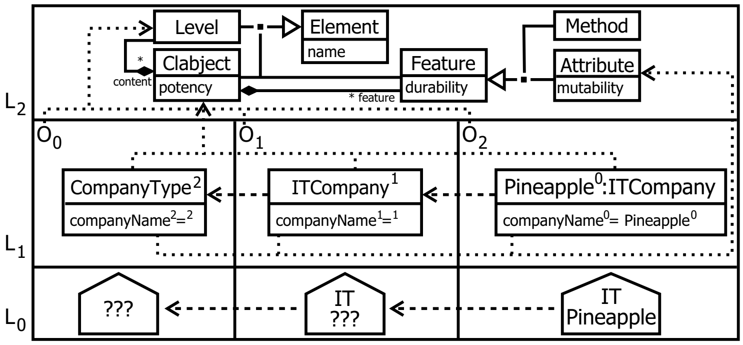


Figure 1: An illustration of the Orthogonal Classification Architecture.

labeled O_0 to O_2. Although this example shows an ontological stack with only three levels, the number of levels is not limited in general. The so called linguistic meta model, L_2, defines all constructs available in the deep modeling language for populating the ontological levels. Examples of these in Figure 1 are *Clabject* and *Feature*. *Clabject* is the superclass for entities and connections and *Feature* the superclass for *Method* and *Attribute* which characterize clabjects. Except for those in the most abstract ontological level, all clabjects have two types in a deep modeling environment — the ontological type (horizontal, dashed lines) indicating the clabject's type from the point of view of the problem domain and the linguistic type (vertical, dotted lines) used to represent a model element in the tool. The linguistic meta model is implemented as an Ecore meta model in Melanee.

Connections and entities are referred to as *Clabjects* because they play the role of types (class) and instances (object) at the same time. This becomes obvious in the middle ontological levels, e.g. *ITCompany* is an instance of *CompanyType* and a type for *Pineapple* at the same time in the example.

Deep instantiation is the capability to specify over how many subsequent levels a model element can influence other model elements. In other words, it represents the "deepness" of a model element. The concept of deep instantiation is implemented in Melanee through potencies which are attached to *Clabjects* (*potency*), *Features* (*durability*) and *Attribute* values (*mutability*). The rule for all potencies is that they have to be decreased by one when an instantiation takes place. Model elements with a potency smaller than 0 cannot exist. In Melanee, potencies are displayed next to the name of *Clabjects* and *Features*, and the *mutability* next to the value of *Attributes* as shown in Figure 1.

3 Melanee

The architecture of the Melanee tool is displayed in Figure 2. The tool is built on the *Eclipse Platform* using the *GMF*, *EMF*, *Epsilon* and *OCL* projects. The basic Melanee download is based on this technology stack. Its features are: 1. the *Linguistic Model* defining the deep modeling language 2. the diagrammatic *Deep Model Editor* featuring the default UML-like syntax for editing deep models 3. the *Visualization Search* algorithm which finds the notations and formats to visualize a model element 4. the *Emendation* service offering seamless deep model editing and 5. *Workbench Management* functionality such as storage of user preferences.

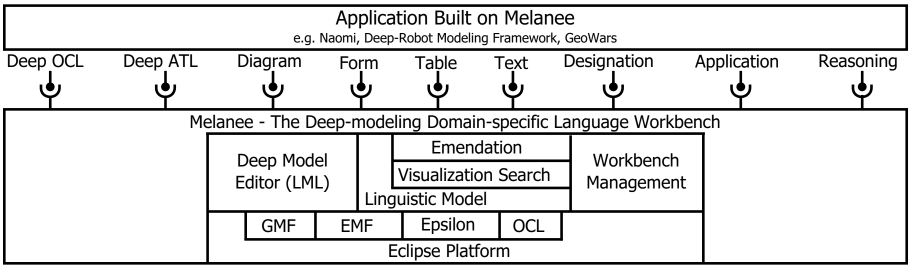


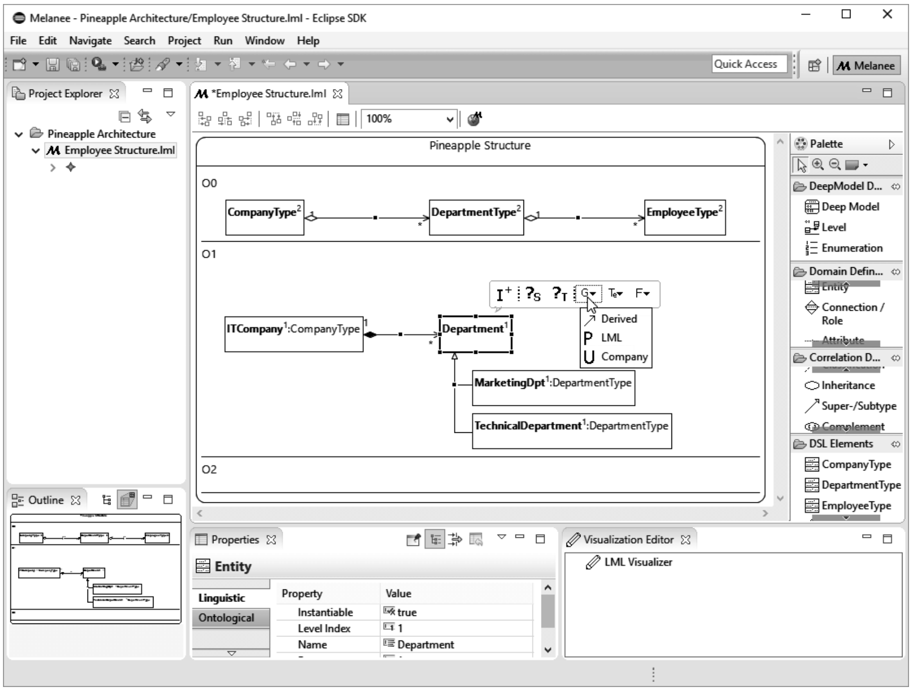
Figure 2: The Melanee architecture.

This core functionality can be extended by implementing extension points. At the time of writing the following extension points are available with reference implementations: 1. the *Deep OCL* constraint language 2. the *Deep ATL* transformation language supporting deep model to deep model transformations, ecore to deep model transformations and deep model to ecore transformations 3. *diagram*, *textual*, *tabular* and *form* supporting predefined and user-defined languages in their corresponding format 4. *designation* capabilities for the advanced naming of model elements 5. an *application* language supporting the modification of the modeling environment itself and 6. *Reasoning* services including model checking. For all of these extension points additional implementations can be provided by using standard Eclipse APIs which lowers the learning curve when extending Melanee.

The Melanee framework has been used to implement several prototypes demonstrating the capabilities of deep modeling. The orthographic software modeling environment Naomi [AGT13a] applies the deep modeling approach to store its central underlying data model and the views on this model using deep, predefined languages. Naomi also realizes transformations between the views and the central data model through the deep ATL [AGT13b] dialect. A prototype robot modeling language [At14] has been implemented which is extensible to different kinds of robots and can be used to create executable behavior by means of executing the C code produced by the textual, user-defined language feature. To demonstrate the advantages of deep modeling in the area of model execution a game, GeoWars [AGM15], has been implemented on top of the Melanee framework. In this demonstrator, a Melanee model is connected through sockets to a model execution engine. This bidirectional connection allows changes in the running game to be reflected in the Melanee model and vice versa. This prototype demonstrates amongst other things the advantages of distinct ontological levels for behavior modeling and behavior execution.

4 Building a User-Defined Language with Melanee

The Melanee workbench is shown in Figure 3. The workbench has a *Project Explorer* on the left hand side which is used for managing projects and the deep models they contain. The example shows a project to model a company called *Pineapple*. This project contains a deep model named *Employee Structure*. Below the *Project Explorer* the graphical *Outline* of the model currently opened in the diagram editor is displayed. Most of the space on the screen is occupied by the diagrammatic, deep model editor. In Figure 3 the *Employee Structure* model is opened. This screenshot shows a modeling language being defined. At this point in time the highest level, O_0 displays three types, *CompanyType*, *DepartmentType* and *EmployeeType*. These clabjects with potency two have been instantiated at the next level, O_1, into the concepts of *ITCompany* and different *Departments*. The *Palette* on the right hand side of the editor provides the tools to build a deep model. Instantiation of model elements is achieved through the *DSL Elements* section which dynamically displays the types which can be added to the currently selected model element. A toolbar offering actions for a particular model element has been opened on the *Department* clabject and the notation "drop down" for the graphical (diagrammatic) format has been selected. The user-defined *Company* notation, the built-in *LML* notation or the notation selected by the model element container (*Derived*) are available for selection. The properties view at the bottom offers capabilities to edit linguistic attributes (*Linguistic* tab page) or ontological attributes (*Ontological* tab page). The *Visualization Editor* view to the right of the *Properties* view is used to define the visualization of clabjects in all formats at run-time.



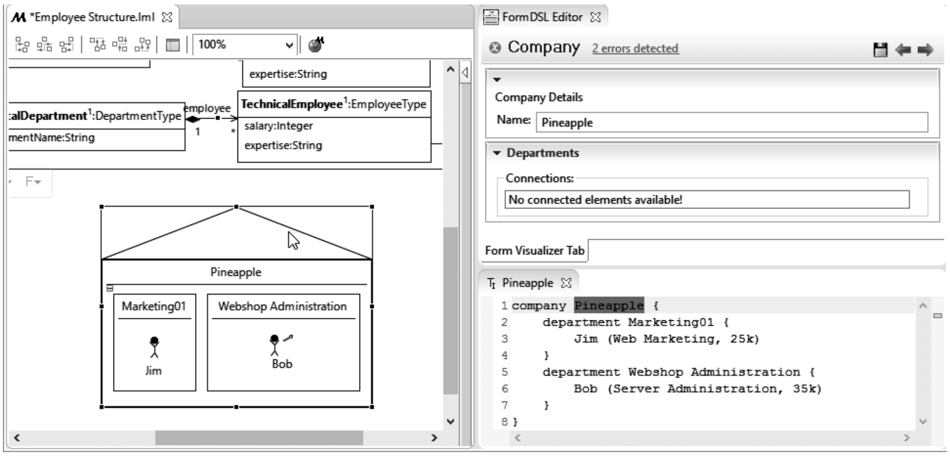
Property	Value
Instantiable	true
Level Index	1
Name	Department

Figure 3: Building up the company structure example.

The final result created during the tool demonstration is displayed in Figure 4. The user-defined modeling language is used to model a company called *Pineapple* consisting of two departments and two employees. The model is displayed in a diagrammatic format (left), form-based format (top right) and textual format (below form editor). The model can be edited in all formats without one format negatively influencing another format.

5 Conclusions

In this tool demonstration we have shortly introduced the deep modeling approach, outlined the architecture of the Melanee tool and shown how a deep modeling language spanning three ontological levels can be defined. Additionally, the created model can be edited



Name: Pineapple

No connected elements available!

```
company Pineapple {
    department Marketing01 {
        Jim (Web Marketing, 25k)
    }
    department Webshop Administration {
        Bob (Server Administration, 35k)
    }
}
```

Figure 4: Displaying *TechnicalEmployee* in three fromats.

in three different formats. Melanee is the only tool at the time of writing which provides such deep, multi-format, multi-notation, user-defined modeling capabilities.

References

[AGF15] Atkinson, Colin; Gerbig, Ralph; Fritzsche, Mathias: A multi-level approach to modeling language extension in the Enterprise Systems Domain. Information Systems, 2015.

[AGM15] Atkinson, Colin; Gerbig, Ralph; Metzger, Noah: On the Execution of Deep Models. In (Mayerhofer, Tanja; Langer, Philip; Seidewitz, Ed; Gray, Jeff, eds): Proceedings of the 1st International Workshop on International Workshop on Executable Modeling. Exe 2015, 2015.

[AGT13a] Atkinson, Colin; Gerbig, Ralph; Tunjic, Christian: A Multi-level Modeling Environment for SUM-based Software Engineering. In: Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling. VAO '13, ACM, New York, NY, USA, pp. 2:1–2:9, 2013.

[AGT13b] Atkinson, Colin; Gerbig, Ralph; Tunjic, ChristianVjekoslav: Enhancing classic transformation languages to support multi-level modeling. Software & Systems Modeling, pp. 1–22, 2013.

[AK01] Atkinson, Colin; Kühne, Thomas: The Essence of Multilevel Metamodeling. In (Gogolla, Martin; Kobryn, Cris, eds): UML 2001. Springer, 2001.

[AK03] Atkinson, Colin; Kühne, Thomas: Model-Driven Development: A Metamodeling Foundation. IEEE Softw., 20(5):36–41, September 2003.

[At12] Atkinson, Colin; Barth, Florian; Gerbig, Ralph; Freiling, Felix; Schinzel, Sebastian; Hadasch, Frank; Maedche, Alexander; Muller, Benjamin: Reducing the Incidence of Unintended, Human-Caused Information Flows in Enterprise Systems. In: 16th International Enterprise Distributed Object Computing Conference Workshops. 2012.

[At14] Atkinson, Colin; Gerbig, Ralph; Markert, Katharina; Zrianina, Mariia; Egurnov, Alexander; Kajzar, Fabian: Towards a Deep, Domain Specific Modeling Framework for Robot Applications. MORSE '14. CEUR-WS.org, pp. 4–15, 2014.

[Me16] Melanee: , Project Website. `http://www.melanee.org`, 2016.