A Feature-based Comparison of Melanee and MetaDepth

Ralph Gerbig[1], Colin Atkinson[1], Juan de Lara[2], and Esther Guerra[2]

[1] University of Mannheim
{atkinson, gerbig}@informatik.uni-mannheim.de
[2] Universidad Autonoma de Madrid
{Juan.deLara, Esther.Guerra}@uam.es

Abstract. Melanee and MetaDepth are two deep modeling tools based on a common set of core concepts such as the orthogonal classification architecture and deep instantiation. However, at the present time, they have different foci. MetaDepth is a textual modeling tool focusing on the needs of software developers while Melanee is a graphical modeling tool focusing on modeling language creation. The question arises, therefore, as to how different/similar the tools actually are and how their features relate to each other. This paper addresses this question by comparing the two tools and investigating the extent to which the modeling features they offer to users overlap.

Keywords: Multi Level Modeling; Deep Modeling; Melanee; MetaDepth

1 Introduction

Melanee [5] and MetaDepth [12] are two of the maturest, publicly available deep modeling tools explicitly built on the notions of orthogonal classification and deep instantiation. Over time they have evolved to specialize on two different aspects of model language engineering and use. MetaDepth is primarily aimed at supporting programmers who want to create deep models using a textual syntax and write executable code against them. Melanee, on the other hand, is primarily aimed at modelers who want to create and use domain specific languages in multiple formats such as table, text and graph-based diagram. Execution of these models is not the main focus of Melanee at the moment, but constraint and action languages are under development at the time of writing.

As far as the data modeling features of the two tools are concerned, they seem to be very similar despite their focus on different target audiences. They are both based on the orthogonal classification architecture [3], and both support the core notions of clabjects [1] and potency [7]. However, they do so in different ways, with some distinct concepts and semantics. Modelers who wish to develop deep models using these concepts are therefore faced with the problem of working out precisely how the tools differ and which tool best suits their needs. The goal of this paper is therefore to clarify the similarities and differences between Melanee's and MetaDepth's underlying modeling approaches as well as the set of features they offer for users. The result of the paper is a list of deep modeling features, presented using the tools in different modeling scenarios found in the

literature, and a discussion of the relevance of these features for specific modeling goals.

The paper is structured as follows. The next section clarifies the terminology used in this paper since the two tools sometimes use different terms for the same concept. This is followed by a comparison of the tools' modeling architectures (Section 3), support for deep instantiation (Section 4), handling of attributes and edges (Section 5) and inheritance rules (Section 6). The paper then presents a small summary of the results of the comparison in a table (Section 7) and a discussion of their significance. Finally, Section 8 concludes with some closing remarks.

2 Terminology

When comparing two modeling tools that sometimes use a different vocabulary for similar concepts, it is important to define the terminology used. Hence, this chapter highlights the terminological differences in the two tools' flavors of deep modeling and explains which terminology is adopted for the remainder of the paper.

The three core terms used to refer to modeling content in MetaDepth are *Model*, *Node* and *Edge*, while Melanee uses the terms *Level*, *Entity* and *Connection* to refer to the same basic concepts. Of these, the term *Model* probably has the potential to cause the most confusion. In MetaDepth a *Model* is the container for all model elements in a single classification level, whereas in Melanee the term *Model*, or more specifically *Deep Model*, encompasses all classification levels (i.e. models). In Melanee, the term used to refer to a container for content at a given ontological Level is *Level*. To avoid confusion, in the remainder of the paper we use the term *Level* to refer to container for model content at a given ontological level, and *Model* for the union of all ontological levels.

There are also differences between Melanee and MetaDepth in the terms used to refer to level content. In particular, model elements representing classes/objects in the problem domain are named *Nodes* in MetaDepth and *Entities* in Melanee, while model elements representing links/associations are called *Edges* in MetaDepth and *Connections* in Melanee. To avoid confusion, in the rest of the paper we use the MetaDepth terms – namely, *Node* and *Edge*. Both tools regard *Nodes* and *Edges* as *Clabject*, and employ the term *potency* to describe over how many subsequent classification levels a clabject can be instantiated. Hence, these two terms (*Clabject* and *potency*) are used in the following without ambiguity.

Another difference of terminology between MetaDepth and Melanee is the use of *attribute* and *field* to refer to the same thing. Melanee uses the term *Attribute*, while MetaDepth uses *Field*. In the following, the term *Attribute* is used as it is widely used in the UML technology space. Finally, the MetaDepth literature uses a technical term, *linguistic extension*, for introducing new types into a model at levels which are not the most abstract. We also use this term with the same meaning in the rest of this paper.

3 Modeling Architecture

Figure 1 shows canonical representations from the literature of the MetaDepth architecture (Figure 1(a)) and Melanee architecture (Figure 1(b)) side-by-side. The two figures show that both approaches are based on the orthogonal classification architecture (OCA) in which one level-spanning meta-model classifies the content of the ontological levels from a linguistic (i.e. tool) point of view. In the MetaDepth version of the OCA (Figure 1(a)), the linguistic meta-model is displayed on the left of the figure labeled *Linguistic meta-model*, and the ontological levels are arranged vertically to the right and use the *@*-notation to indicate their potency. The Melanee version of the OCA (Figure 1(b)) displays the linguistic meta-model at the top of the figure labeled L_2, while the ontological levels are arranged horizontally and are labeled using the O prefix, e.g. O_0. The Melanee example additionally shows the real world which is modeled at level L_0. This real world exists in the MetaDepth architecture too but is not displayed in Figure 1(a). In both architectures, model elements are not only classified by their linguistic types but can also be classified by an ontological type residing at a higher ontological level. Multiple ontological classification is not allowed in MetaDepth but is allowed Melanee. In [11], however, a mechanism is presented for retyping a clabject with respect to several other clabjects after creation from one ontological type.

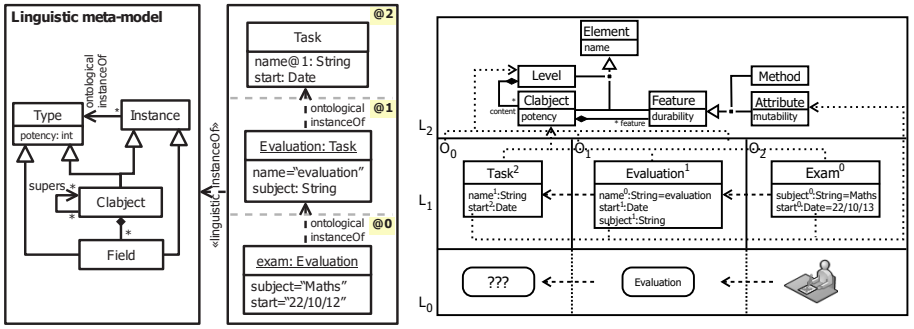


Fig. 1. MetaDepth architecture (a) from [14] and Melanee architecture (b) adapted from [6].

It can be observed that both linguistic meta-models share the concept of *Clabject* which is a concatenation of the words *Class* and *Object*. This term reflects the type instance duality of model elements residing in the middle classification levels. They are at the same time instances of their types at the ontological level above and types for their instances one ontological level below. In the MetaDepth version of the OCA, this dual nature of clabjects is explicitly modeled by defining *Clabject* as a subclass of both *Type* and *Instance*.

For one ontological type to conform to another in these architectures, certain conditions have to be fulfilled by the instances. The set of attributes of the

instance has to conform to the set of attributes defined by the ontological type, the connection cardinalities defined by the ontological type have to be satisfied by ontological instances, and the rules for potency have to be satisfied. These three criteria are the same in MetaDepth and Melanee.

4 Deep instantiation

Melanee and MetaDepth are both based on deep instantiation. MetaDepth allows a potency for levels, clabjects, and fields to be configured while Melanee allows potencies to be defined for clabjects, operations, attributes and their values. The fact that levels do not have potencies in Melanee has the advantage that an unlimited number of classification levels can be defined without thinking of the deepness of a deep model when assigning a potency to the initial classification level. However, it has the disadvantage that potencies have to be defined for each clabject in the level. Futhermore, MetaDepth allows models (i.e. levels) to be imported into other models for reuse. In Melanee only level content such as clabjects and packages can be linked.

The linguistic meta-model element, *Operation*, shown in the linguistic meta-model of Melanee in Figure 1(b) is not available in MetaDepth. Hence, in MetaDepth only attributes can be added to clabjects.

Potencies facilitate deep instantiation [8, 10] by expressing the *deepness* of a model element's instantiation tree, i.e. how many subsequent ontological levels it can influence through its classification tree depth and through the existence and values of attributes over that tree. However, there are differences in the way Melanee and MetaDepth define potency for model elements. In MetaDepth the potency is initially associated with levels as indicated in Figure 1(a) and then applied to the level content, i.e. *clabjects* and *attributes*. In other words, the default value for clabject potencies in MetaDepth is the potency of the level. Attributes inherit their default potency from their containing *clabjects*. *Clabjects* can have a potency differing from their level's potency, but this has to be explicitly stated. On the contrary, in Melanee, the potency is explicitly defined for each *clabject* and *attribute* as shown in Figure 1(b).

The rules for potency reduction during instantiation are identical for Melanee and MetaDepth. Instances always have a potency that is one lower than the potency of their type. Model elements with a negative potency cannot exist, and so, model elements with a potency value 0 cannot be instantiated. They therefore correspond to concepts such as objects and slots, when compared to the UML technical space.

Both tools allow instantiation over an arbitrary number of levels using so-called *star potency*. When instantiating a model element with star potency, the instances can have a star potency themselves or a potency with a non-negative Integer value. This is an advantage when defining frameworks in the form of a modeling language for instantiation. In such a scenario, a modeler often does not know in what environment the framework will be used and thus cannot define appropriate constraints on the instantiation of the provided types.

Strict meta-modeling [4] requires each instance to have a type exactly one level above. This may lead to *identity instantiations* where clabjects are instantiated at each ontological level without adding additional information, just for the pure reason of supporting the instantiation of further elements at the level below. To overcome this problem, MetaDepth offers a special feature exclusive to the tool called *leap potency* [13]. If a type is assigned leap potency n, then instances of the type can be created n levels below, eliminating the need to instantiate the type at intermediate levels.

Linguistic extension, the ability to add new types at any intermediate level without being classified by an ontological type, is supported by both tools.

5 Attributes and Edges

There are some significant differences between MetaDepth and Melanee when it comes to the definition of attributes and edges. In MetaDepth an attribute does not have to be present (repeated) at all intermediate levels. The only level at which an attribute is mandatory is the lowest level, where it plays the role of a slot. However, attributes can also be added to intermediate levels to hold default values. In Melanee, an attribute has to be present throughout the whole instantiation tree of a type. Melanee, in contrast to MetaDepth, allows finer-grained control of attributes by assigning a value potency (*mutability*) to them in addition to the attribute potency (*durability*). The durability defines over how many subsequent levels an attribute can exist (i.e. endure) and the mutability defines over how many levels the value of an attribute can change. Together they allow modelers to capture precise details about the ownership, visibility and changeability of attributes across classification levels, akin to features such as tags, tag values, statics variables and constraints in the UML space. For example it is possible to define that an attribute belongs only to the type facet of a clabject and thus is not derived by its instances (by assigned it durability 0), or that an attribute's default value defined at the class level cannot be changed by its instances (by assigning it mutability 0).

MetaDepth offers richer functionality than Melanee when it comes to connecting clabjects with each other. Melanee only allows edges to be represented by connection classes, where each Edge is a fully fledged clabject containing attributes, operations and connection ends identifying the connected clabjects. MetaDepth goes beyond this by also allowing edges to be represented by simple *references* which are essentially attributes whose value is a reference to another clabject.

Both approaches support *connection diversification* [2] as they allow connection ends / reference-names to be renamed at the instance level. Melanee allows edges to represent containment and aggregation as well as basic associations, whereas MetaDepth is only aware of basic associations. Also, MetaDepth does not allow clabjects to contain other clabjects like Melanee.

MetaDepth supports deep references (Figure 2) which are not available in Melanee. Deep references help when nodes need to be connected but the direct types are not known at the time of modeling. In Figure 2, for example, a *Page*

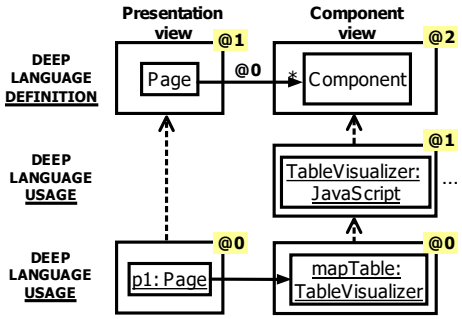


Fig. 2. Deep reference example from [13].

instance shall be connected with a *Component* instance. *Component* however is instantiated at one more level in contrast to *Page* resulting in potency 0 *Page* instances being connected to indirect potency 0 instances of *Component*. In the example *Component* is instantiated as *TableVisualizer*. The *Page* instances shall be connected with instances of this direct instance of *Component*. Deep references in MetaDepth support the creation of such references whose direct type is not currently known. By assigning *@0* to the reference between *Page* and *Component* it is declared that *Page* instances shall be connected with indirect *Component* instances of potency 0.

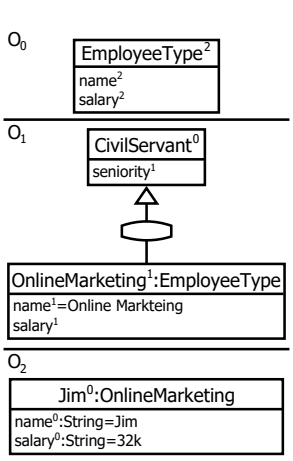
The multiplicity (cardinality) semantics of MetaDepth restricts the number of instances of a connection at the level below. Each instance then defines its own multiplicity for the following level. In Melanee, the multiplicity constraint of the type level is satisfied if the sum of the multiplicities of all connection instances is in the range of the multiplicity defined at the type level. Hence, a multiplicity constraint not only constrains the immediate level below but all following ontological classification levels. Approaches providing finer grained control of multiplicity to Melanee are also feasible as described in [2].

Enumerations can be defined in both Melanee and MetaDepth. Melanee allows an enumeration to be defined in a deep model and then used by clabjects across all classification levels. MetaDepth defines enumerations within levels which can then be used at all subsequent levels.

6 Inheritance at intermediate levels

Melanee and MetaDepth follow similar rules concerning inheritance. Subtypes must have a potency equal to or higher than their supertype. Attributes can be repeated in subtypes in order to override values defined in supertypes. Inheritance relationships can exist at all levels including intermediate levels and the most concrete level. Hence, inheritance works for both type and instance facets of clabjects, enabling a sort of prototype-based modelling. Moreover, multiple inheritance is supported by both approaches.

When it comes to typing of participants in an inheritance relationship Melanee and MetaDepth differ. Unlike MetaDepth, Melanee allows supertypes



(a)

```
1  Model O0@2{
2   Node EmployeeType{
3    name:String;
4    salary:String;
5   }
6  }
7
8  O0 O1{
9   abstract EmployeeType CivilServant{
10   seniority:String
11  }
12  EmployeeType OnlineMarketing:CivilServant{
13   name="Online Marketing";
14  }
15 }
16
17 O1 O2{
18  OnlineMarketing Jim{
19   name="Jim";
20   salary="32k";
21  }
22 }
```

(b)

Fig. 3. Inheritance example in Melane (a) and MetaDepth (b).

which do not have an ontological type. MetaDepth always forces a supertype to have an ontological type which is *compatible* with the ontological type of its subtypes. Compatible means that the ontological type of the supertype has to be equal to, or a supertype of, the ontological type of the subtype. Hence, the Melanee inheritance example shown in Figure 3(a) is not allowed in MetaDepth. To make the example valid in MetaDepth *CivilServant* would have to be assigned *EmployeeType* as its ontological type as shown in the MetaDepth version of the example in Figure 3(b).

This restriction prevents the definition of abstract supertypes with a potency of 0 if the subtypes have ontological types in MetaDepth. In Melanee this technique is commonly used to represent abstract classes, especially in the context of the power type pattern. In MetaDepth, an abstract class always has to be ontologically typed and have a potency one less than the potency of the ontological type. In most cases this new potency is different to 0 for instances residing in middle classification levels. To overcome this problem the *abstract* keyword is introduced in MetaDepth's textual syntax as shown in Figure 3(b).

The difference between MetaDepth's programming language orientation and Melanee's structural modeling orientation can be seen in the example in Figure 4. MetaDepth does not allow inheritance in the middle level of the example where the supertype (*A1*) is an ontological instance of a different ontological type, *A*, to the subtype (*B1*), which is of type *B*. The problem is that *A* and *B* are not related by inheritance at the top ontological level. MetaDepth prevents this because it is possible that a constraint written on *A* is applied to instances of *B* in such a model as shown in Figure 4. Such behavior is not easily foreseen by programmers and is thus prohibited in MetaDepth in favor of simpler and more reliable constraint definition. Melanee, however would al-

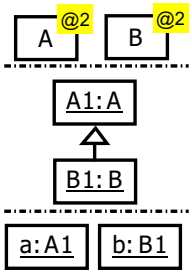


Fig. 4. Inheritance example from [13].

low such a situation since *B* could be assumed to be a subtype of *A* in a case where the sets of instances are overlapping and hence it is correct to have the same constraint also applied to instances of *B*. Melanee's reasoning services [9] could even discover an inheritance relationship between *A* and *B* based on their property sets.

In addition to pure generalizations as supported in MetaDepth, Melanee adds the ability to define generalization sets which can mark the instance set as complete/incomplete and overlapping/disjoint. Furthermore, names (a.k.a discriminants) can be given to these generalization sets.

7 Comparison Summary

In the previous sections the languages of Melanee and MetaDepth were compared in terms of their implementation of potency, their handling of attributes and edges, classification semantics, and their semantics for inheritance. The results are summarized in Table 1 and Table 2. This shows that the number of common features (highlighted cells) roughly matches the number of differences in all four of these areas.

		MetaDepth	Melanee
Potency	Potency Reduction	– Potency at instance one lower than at ontological type level – potency 0 clabjects cannot have instance	equal to MetaDepth
	Leap Potency	available	not available
	Star Potency	available	available
	Strict Meta Modeling	– leap potency – deep references	enforced
	Potency Declaration	– At level, clabject – clabjects inherited potency from level if not stated otherwise	At each clabject
	Abstract Keyword	available	no keyword, potency of 0

Table 1. Summary of the comparison part 1

		MetaDepth	Melanee
Inheritance	Potency	subtype potency equal or higher supertype	equal to MetaDepth
	Disjoint Types of super and subtype	not allowed	allowed
	Untyped supertype for typed subtype	not allowed	allowed
	Multiple Inheritance	supported	supported
	Generalization Sets	not supported	supported
	Attribute Overriding	by duplication	equal to MetaDepth
Attributes, Edges & Distributed Modeling	Durability	derived from Clabject, but can be set indivually	set independent from Clabject
	Mutability	not available	available
	Repeated at instances	not mandatory	mandatory
	Default Values at Intermediate Levels	available	available
	Association Classes	available	only available kind of connections
	References (complex attributes)	available	not available
	Cardinality	effects immediate level below	effects all levels below
	Containment & Owning	not available	available
	Association Types	plain associations only	containment, aggregation, plain association
	Connection Diversication	supported	supported
	Enumerations	supported	supported
	Operations	not available	available
	Distributed Modeling	importing of Levels (i.e. Models in MetaDepth)	linking to remote Clabjects and Packages
Classification	Clabject Conformance	– potency – attributes – edges & references	equal to MetaDepth
	Multiple Ontological Classification	dynamic retyping [11]	available
	Linguistic Extension	allowed	allowed

Table 2. Summary of the comparison part 2

8 Conclusion

This paper has presented a comparison between the Melanee and MetaDepth approaches to deep modeling and revealed that even though the two languages might look quite similar at first sight and share a lot of common ideas, they have significant differences in their detailed terminology and semantics. The next step is to extend the comparison to include a broader range of deep modeling tools in order to create a concrete set of criteria for comparing them. Once such an exhaustive set of features has been defined it can be used as the basis for empirical studies into which language features should be targeted to which user groups and for defining guidelines for choosing deep modeling tools. We are working on building transformations between the two tools to bridge their semantic differences. We hope this paper may serve as a starting point for providing criteria to compare deep modeling tools, and encourage other researchers to compare their tools to MetaDepth and Melanee, providing further comparison points.

Acknowledgements. Work partially sponsored by the Spanish MINECO (TIN-2014-52129-R), and the Madrid Region (S2013/ICE-3006).

References

1. Atkinson, C.: Meta-modelling for distributed object environments. In: Enterprise Distributed Object Computing Workshop [1997]. pp. 90–101 (1997)
2. Atkinson, C., Gerbig, R., Kühne, T.: A unifying approach to connections for multi-level modeling. In: MODELS (2015)
3. Atkinson, C., Gutheil, M., Kennel, B.: A flexible infrastructure for multilevel language engineering. Software Engineering, IEEE Transactions on 35(6) (2009)
4. Atkinson, C.: Supporting and Applying the UML Conceptual Framework, pp. 21–36. Springer Berlin Heidelberg (1999)
5. Atkinson, C., Gerbig, R.: Flexible deep modeling with melanee. In: Modellierung 2016, 2.-4. März 2016, Karlsruhe - Workshopband. pp. 117–122 (2016)
6. Atkinson, C., Gerbig, R., Fritzsche, M.: A multi-level approach to modeling language extension in the enterprise systems domain. Information Systems (2015)
7. Atkinson, C., Kühne, T.: The essence of multilevel metamodeling. In: Gogolla, M., Kobryn, C. (eds.) UML 2001. pp. 19–33. Springer Berlin Heidelberg (2001)
8. Henderson-Sellers, B., Gonzalez-Perez, C.: The rationale of powertype-based metamodelling to underpin software development methodologies. In: Proceedings of the 2Nd Asia-Pacific Conference on Conceptual Modelling. APCCM '05 (2005)
9. Kennel, B.: A Unified Framework for Multi-level Modeling. Ph.D. thesis, University of Mannheim (2012)
10. Kühne, T., Steimann, F.: Tiefe charakterisierung. In: Modellierung 2004, Proceedings zur Tagung, 23.-26. März 2004, Marburg, Proceedings. pp. 109–119 (2004)
11. de Lara, J., Guerra, E., Cuadrado, J.S.: A-posteriori typing for model-driven engineering. In: MODELS. pp. 156–165 (2015)
12. de Lara, J., Guerra, E.: Deep meta-modelling with metadepth. TOOLS'10, Springer-Verlag, Berlin, Heidelberg (2010)
13. de Lara, J., Guerra, E., Cobos, R., Moreno-Llorena, J.: Extending deep metamodelling for practical model-driven engineering. The Computer Journal (2012)
14. de Lara, J., Guerra, E., Cuadrado, J.S.: Model-driven engineering with domain-specific meta-modelling languages. Software & Systems Modeling 14(1) (2015)