A GENERIC MODEL FOR CONNECTING MODELS IN A MULTILEVEL MODELLING FRAMEWORK

Jan Pettersen Nytun
Agder University College
Faculty of Engineering, Agder University College
Grooseveien 36, N-4876 Grimstad, Norway

Keywords: Megamodel, multilevel modeling, metamodeling, model weaving, model border.

Abstract: In science and elsewhere models are weaved together forming complex knowledge structures. This article presents a generic way of connecting models with model borders both vertically and horizontally in a multilevel modelling framework. One model can be connected vertically to several models allowing a model element to be an instance of several metaclasses and different views can then be managed in an integrated way. Models at the same level can also be connected by defining the correspondence between model elements. The idea behind the approach is to break model architectures down to elementary building blocks so that all parts that might be of interest become explicit and accessible.

1 INTRODUCTION

In this article some of the ideas behind a metamodelling framework called Semantic Integration World Animation (Siwa) is presented; this framework is under development at Agder University College and it is meant for learning and experimentation; it is an offspring of the SMILE project (Nytun et al., 2004) which is more directed towards integration of existing language technologies.

MOF metamodel (OMG Editor, 2003) architectures have a pyramid structure, while a Siwa architecture is like a directed graph with models as nodes. The graph is not cyclic except maybe for the top models (e.g. level `M3` in the UML metamodel architecture).

OMG has issued a request for revision of MOF 2.0 (OMG Editor, 2006a), some of MOFs restrictions are becoming increasingly burdensome. MOF does not allow properties to have an independent existence and multiple classification is not possible - Siwa can be used without these restrictions. It will also be possible to specify architectures that are not complete, e.g. that a metamodel is missing; this opens up for data analysis, reasoning about models and in some cases the framework might automatically suggest a metamodel.

Some models are static structures and some are executable models. We call the executable models *semantic engines*, some sematic engines are presented but they are not the main issue in this article.

A model is to a large extent defined by the role it plays in relation to what it models; two basic roles are defined by Thomas Kühne (Kühne, 2005): *token* and *type*. A token model captures the singular aspects, while a type model captures the universal aspects of what it models. A class `Building` might capture the universal property that buildings have owners. An object that models one specific building is a token model for this building, e.g. it might capture the name of the owner. The focus of this article is a technique for connecting models, the following *model configurations* are to be supported:

Vertical (type model) This is the type model role which spans two model levels (some would call this the `instanceOf`-relation). Several models can be type models for the same model instance; this is not supported by MOF.
There are variations of this relationship, e.g. a model instance might actually have been instantiated from the model or the model is describing only some aspects of the model instances.

Horizontal (token model) We see the need for a model-to-model relationship which do not span a level border, but is between two models that are considered to be on the same level. Several models can in different way and with different level of granularity and detail model the same thing; these models are related with this relationship.

Pettersen Nytun J. (2006).
A GENERIC MODEL FOR CONNECTING MODELS IN A MULTILEVEL MODELLING FRAMEWORK.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 302-311
DOI: 10.5220/0001322303020311

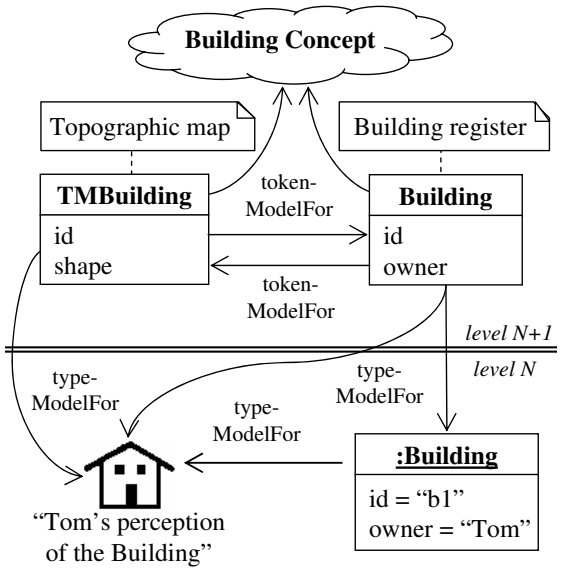

Figure 1: Example of token and type model.

Fig. 1 demonstrates, as we understand it, both token and type model roles (the UML notation has been used in an ad hoc fashion).

The transitive property of the token model role can also be seen in Fig. 1: the `TMBuilding` class is a token model for class `Building` which is a token model for "the concept of a building", consequently `TMBuilding` is also a token model for "the concept of a building". It seems to be a growing agreement (Kühne, 2005; Favre, 2004a) that a metamodel is a type model for another model which again is a type model for its model instance. These models are forming a stack structure where you don't have the same transitivity as for the token model role. Subclassing is a transitive relation and should not span a level border (Kühne, 2005; Favre, 2004a).

UML has no diagram type that truly spans several (metamodel) levels; UML *object diagrams* shows *instance specifications* (instances of metaclass `InstanceSpecification`) and also classes are allowed; an object diagram is placed on the model level (`M1`). As the name indicates, an instance specification is a specification and it might in fact specify properties of several instances at the model instance level (`M0`). According to this understanding, an object diagram is correctly placed on `M1` because an instance specification is not "truly" in a horizontal relation to an instance on `M0`. In our view, if a model is in a horizontal relation to another model, then both models are modelling "exactly the same specific thing" even if the number of details and precision might be different; the models should consequently be placed on the same level since they model the same thing.

The idea behind our approach is to break model architectures down to elementary building blocks so that all parts that might be of interest become explicit and accessible; the framework should of course allow the user to view an architecture at different levels of granularity and with different concrete syntaxes that hide the underling complexity.

Using the example in Fig. 1: object `:Building` is a structure that contains a slot called `id` with value `"b1"` and a slot called `owner` with value `"Tom"`; we consider `Building`, `id` and `owner` to be symbols that forms an *upper border* to the type model containing class `Building`; we actually have two borders (or border sides), one for each models being connected, this allows different number of symbols at the borders and it allows symbols to have different names.

Sec. 2 presents our multilevel (meta-)modelling framework and explains how models are connected. In Sec. 3 we mention some related work. Sec. 4 presents an example of how Siwa can be used to do testing of data consistency. We summarize and describe some research directions in Sec. 5.

2 THE SIWA APPROACH

Lately the term megamodel has been used to name a model of MDE itself, e.g. by Bézivin and Favre (J. Bézivin, 2004; Favre, 2005). Such a megamodel describes the concepts of MDE - concepts like model, metamodel and transformation. Fig. 2 shows our preliminary megamodel which has been inspired by Favre (Favre, 2005).

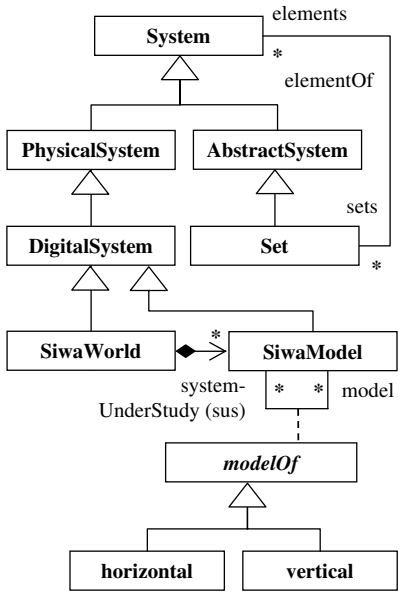

Figure 2: Megamodel.

As we can see from Fig. 2 basically everything is a *system*. In (Marcos Didonet Del Fabro, 2005b) a system is described as a group of interacting, interrelated, or interdependent elements that form a complex whole.

Abstract systems can only be described since they are not to be found in the *concrete*; *physical systems* are concrete and manifested "in reality". We consider a *computer system* to be a special type of *physical system* since they are manifested in computer hardware.

The Siwa framework supports the notion of levels like you find it in metamodelling architectures defined by OMG. Atkinson and Kühne (Atkinson and Kühne, 2001) have earlier used the term *multilevel metamodelling*, we prefer the term *multilevel modelling framework* since an arbitrary number of levels will be supported including metamodel levels. We call a multilevel metamodel architecture defined in the framework for a *Siwa world*. Fig. 2 defines a Siwa world as a special type of digital system. A Siwa world is composed of Siwa models which also are considered to be special types of digital systems. The `modelOf` relation comes in the two generic types: vertical and horizontal.

A `Siwa model` can be a model for an abstract or a physical system which is not part of a Siwa world. This possible relation is not depicted in Fig. 2 since it can not be explicitly represented in the framework.

Favre presents briefly the notion of *static* and *dynamic system* in (Favre, 2005), a Siwa world also has these two aspects which we call: Model All Types with Extent Realization (MATER) and Play Activations and Transformations with Extent Realizations (PATER). The focus of this article is MATER and in the following subsections MATER is presented with the help of UML notation and examples. PATER is touched in Sec. 4 when some sematic engines are described.

In Subsec. 2.1 we present how to represent the internal structure of a Siwa model, in Subsec. 2.2 we describe how Siwa models can be connected to constitute a Siwa world (a multilevel model architecture).

2.1 Representing One Model

Fig. 3 presents the part of MATER that is used when one model is to be represented, it is meant to be used on all the levels of a Siwa world. There are several similarities between this part of MATER and MOF (OMG Editor, 2003) as an instance model, e.g. an object can be represented as an instance of `Structure` containing instances of `Slot` with values representing properties of the object and `Descriptor` can be used for type information as described later. Links between objects can be represented as instances of `Link`. The property/owner associations can be used to represent relations between sets; since there is no reference to a `Descriptor`, instantiation of these associations might in some cases lead to ambiguous situations when it comes to finding their description on the level above. Fig. 3 has two types of symbols:

Descriptor This symbol type is used to indicate classification and it is used to establish a vertical relation, e.g. structure describing a building called `b1` might have a descriptor called `Building`.

Identifier An instance of `Identifier` is labeling a part of the model being defined and functions as an identifier for this structure, e.g. an identifier `b1` might reference structure that describes a building with that id. Another example would be an identifier `Building` referencing a structure that describes a class `Building`. Several identifiers might reference the same structure; in some cases this means that there are synonyms.

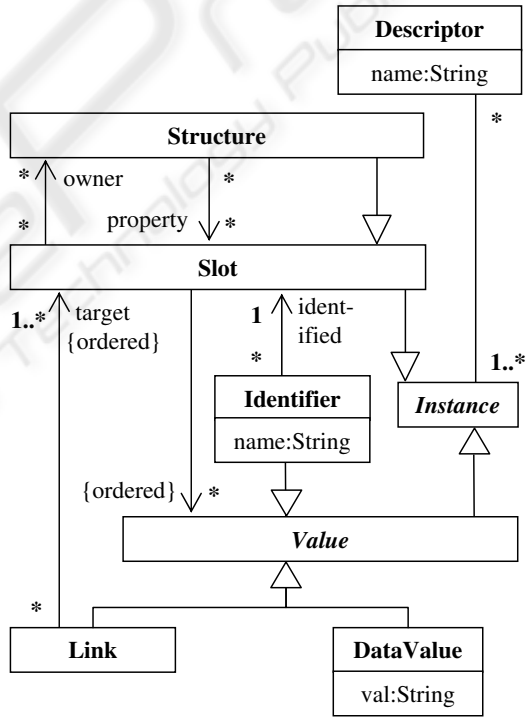

Figure 3: Part of MATER: the internals of a Siwa model.

For us as humans the symbols are typically telling what a model is about, from the "framework point of view" only what has been formalized and represented in MATER "does matter".

An example of how an object of type `Building` can be represented is given in Fig. 5. In Fig. 5(a) the object is shown in UML notation, Fig. 5(b) shows how MATER can be instantiated to represent the same. Fig. 5(c) is showing an overview of the

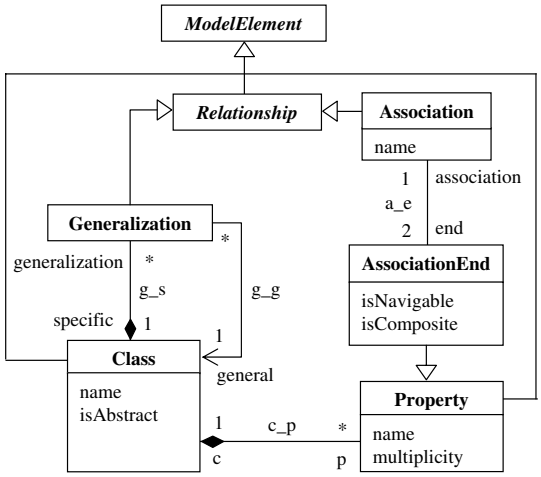

Figure 4: The top metamodel of the examples.

`Building`-object in an ad hoc notation where the structure is hidden except for the symbols (two *borders* are shown, marked `U` and `L`, this concept will be explained below). Fig. 4(a) shows the top model which is called Class-MM. It describes important object-oriented concepts like: abstract and concrete class, property, multiplicity, association and generalization.

Fig. 6(a) shows a simple class called `Building` with a property called `id`; Fig. 6(b) describes how Class-MM can be instantiated to get the class and then Fig. 6(c) demonstrates how MATER can be used to describe the class (the `:Property`-object is not shown). As we can see from the figure the number of model elements is huge - it correspond approximately to the number one would get if the UML metamodel was instantiated.

In the object-oriented literature, and also in this article, the difference between a *class as a set* (something abstract) and the *description of a class* is often confused[1]. As a consequence of being abstract: it is not possible to ”point to” a class and say ”there it is”, but it might be possible to point to the instances of a class and also to a description of a class[2]. MATER does not have *class* as a built-in construction - there is no model element in Fig. 3 called `Class`, but it is possible to describe classes (e.g. Fig. 6(c)).

Seeing a class as an object is not in conflict with the UML metamodel architecture since a UML model can

[1]In our view reification is merely to establish a descriptions of a concept.

[2]The terms *abstract class* and *concrete class* used in object-oriented programming is something else, in that context a concrete class means that there are objects that are direct instances of the class which is not the case for abstract classes.

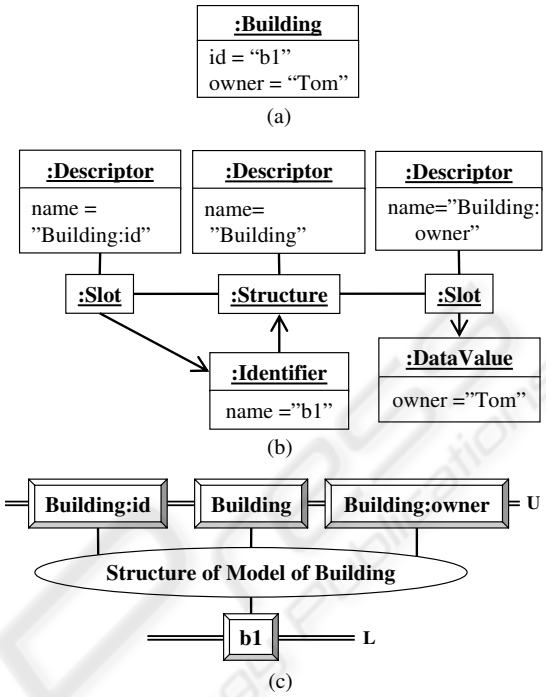

Figure 5: Example of how to represent an object in MATER.

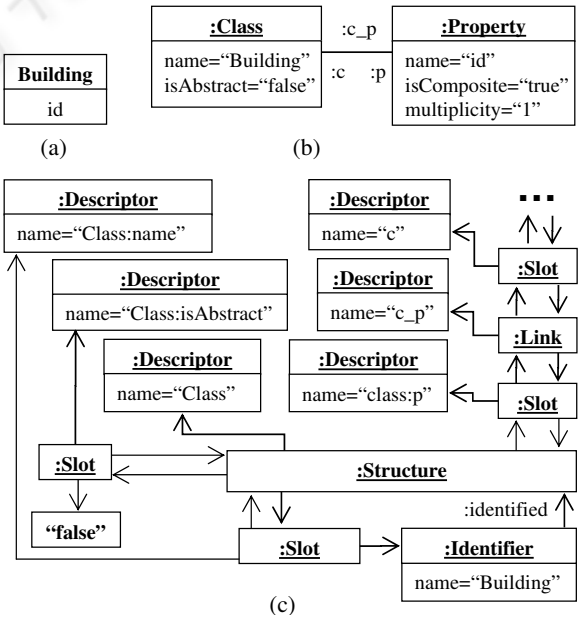

Figure 6: A part of the description of class `Building`.

be seen as composed of objects instantiated from the UML metamodel (see (Atkinson and Kühne, 2002) for more on this class/object nature); the UML metamodel can again be seen as composed of objects instantiated from MOF and MOF can be seen as composed of objects instantiated from itself.
A class `Building` in a UML class diagram is an instance of class `Class` of the UML metamodel, we understand that `Building` will be a class when we read about the semantics of class `Class` (OMG Editor, 2006b): *A class is a type that has objects as its instances...The instances of a class are objects.* From this description we understand that a whole UML metamodel architecture can be depicted as an object diagram, which is known from the literature (e.g. (Nytun et al., 2004) and (Martin Gogolla, 2005)).

MATER offers several ways to model the same thing and it is not ”strongly” constrained - this is deliberate and opens for experimentations. It is not discussed in the article but it will be possible to configure the framework with the help of some OCL-like language, e.g. enforce *strict metamodelling* (Atkinson and Kühne, 2000).

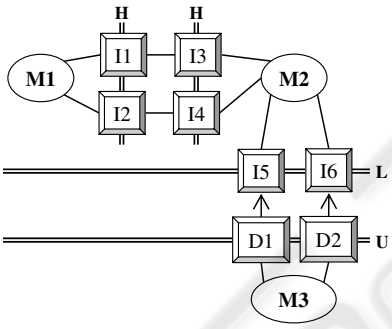

Figure 7: Sketch of connected models.

2.2 Connecting Models

Fig. 8 extends MATER and adds the possibility to connect models. A Siwa model can contain borders; two models are connected by connecting two borders, one from each model. When the models to connect are on the same level both borders will be of type `HorizontalBorder` (Fig. 8(c)), when the models are on different levels the border on the lower level is of type `UpperBorder` (Fig. 8(b)) and the border of the upper model is of type `LowerBorder`.

Fig. 7 offers a sketch where a model called `M3` is connected to a model `M2` that resides on a level above. `M3` has an instance of `UpperBorder` (marked with letter U) containing two instances of `Descriptor` called `D1` and `D2`; these two symbols are connected to instances of `Identifier`, `I5` and `I6` respectively;

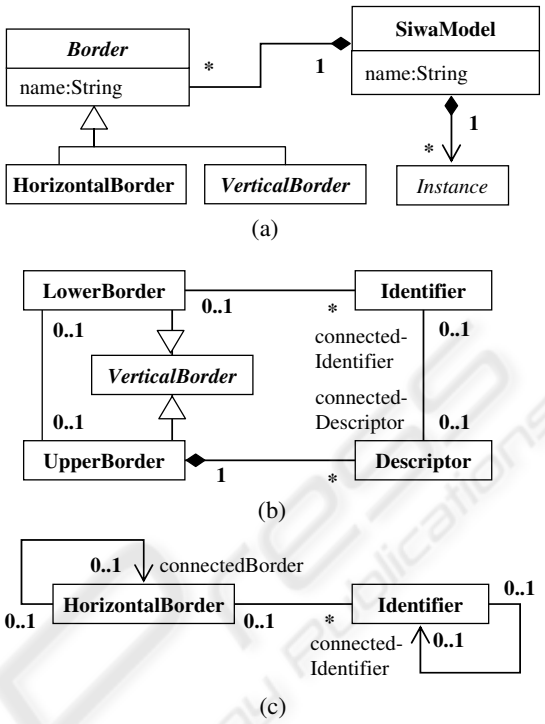

Figure 8: Part of MATER: connecting Siwa models.

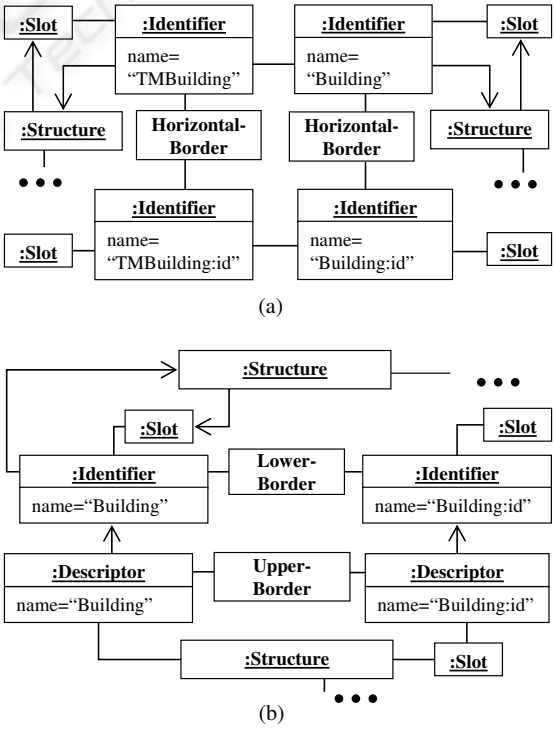

Figure 9: The example (incomplete) of Fig. 1 in MATER.

M2 has an instance of LowerBorder (marked with letter L) containing both I5 and I6. Fig. 7 is also demonstrating how model M1 and M2 residing on same level are connected by two instances of HorizontalBorder (marked with letter H).

The vertical association in Fig. 8(b) has multiplicity 0..1 on the Identifier side, this means that incomplete architectures, as claimed in the introduction, are possible. The claim that a model can have several metamodels is justified by allowing several upper borders for one and the same model.

Fig. 9 shows in more detail an example (same example as in Fig. 1) of how MATER can be instantiated to connect models, (a) shows how TMBuilding and Building residing on the same level are connected, while (b) shows how Building and :Building residing on different levels are connected.

This way of connecting models is an extension to what we have presented before; (Nytun et al., 2004) presents a solution where two models would share a common border instead of having one border for each model to be connected; also the connecting of models on the same level is new.

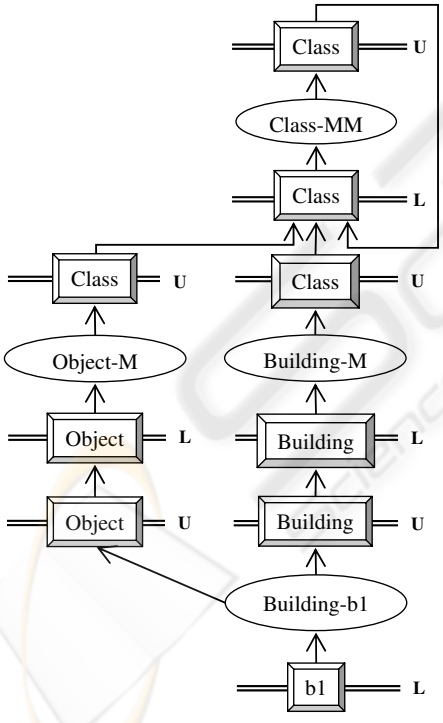

Figure 10: Architecture of building example (TMBuilding is not included).

In Fig. 10 some of the models described above are put together to form a multilevel architecture. The only model that has not been mentioned before is the one named Object-M; as can be seen in Fig. 11 it

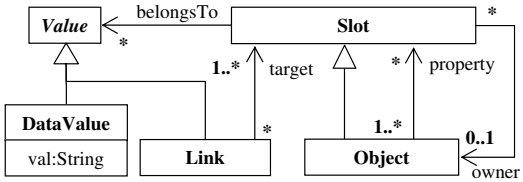

Figure 11: Model Object-M in detail.

simply defines an object as a structure having slots with values and links to other objects. The instance of Structure representing the building object has both Building and Object as descriptor.

Model Object-M is introduced to demonstrate that a model can have several type models. Object-M could be used to define a more general XML format than if Building-M was used.

3 RELATED WORK

Today there is much interest in the use of metamodels, e.g. in MDA (OMG, 2003), MDE (Favre, 2004b), LDD (Fowler, 2005), DSL (Greenfield et al., 2004).

The part of Siwa presented in this article, which is mainly the static part, can be used as a starting point in all the mentioned fields.

There are several other meta-modelling frameworks, to mention a few: MetaEdit+ (Metacase, 2006), Coral (Marcus Alanen, 2004), XMF (Tony Clark, 2004), EMF (EMF, 2006) and MPS (MPS, 2006).

Rondo (S. Melnik, 2003) is a programming platform for generic model management and it includes high-level operators used to manipulate models and mappings between models. AMW (Marcos Didonet Del Fabro, 2005a) goes further and allows extensible mappings. AMW (Marcos Didonet Del Fabro, 2005a) is a generic model weaver that allows the specification of correspondences between model elements from different models - models are in this way connected with a model, e.g. correspondence *Equals* might be established between the two id attributes of classes TMBuilding and Building of Fig. 1.

Our approach has similarities with the aforementioned works, but we have not found a framework that allows models and model levels two be connected so freely as our approach does, e.g. the weaving of models described above can be achieved simply by introducing another model with borders to the models to be weaved; this new model will describe the structure of the correspondences, semantic engines can then be defined to handle the semantics of the correspondences; a somewhat related example is given below.

4 LEGACY DATA CONSISTENCY AS EXAMPLE

Our article (Nytun and Jensen, 2003) focused on the consistency problems that occur when previously uncoordinated, but semantically overlapping data sources are being integrated. The paper presented techniques for modelling consistency requirements using OCL and other UML modelling elements. The paper also considered the automatic checking of consistency in the context of one of the modelling techniques. This section presents an outline of how Siwa can be applied to implement one of these techniques and how automatic testing of consistency can be performed.

4.1 Consistency Modelling and Testing

Fig. 12 shows an integration of two legacy models, where one is a description of apartments (class `Apartment`) and the other a description of buildings (class `Building`). The consistency requirements are as follows:

1. The number of apartments that is given as a property in class `Building` should be equal to the number of apartments with the same building id (attribute `bId`).
2. One building should have at least one apartment, and an apartment should belong to exactly one building.

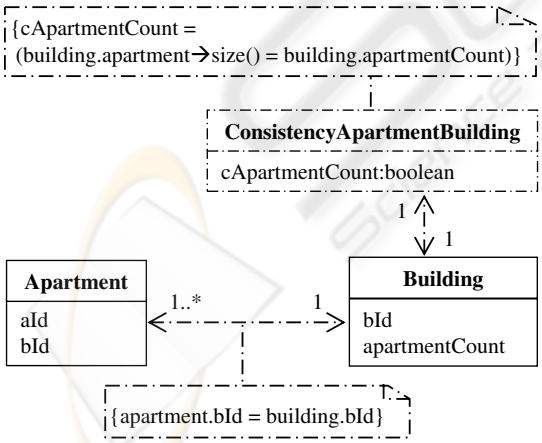

Figure 12: Consistency between `Apartment` and `Building`.

The elements with dash-dotted line style in Fig. 12 constitute what we call a *consistency model*, this model is manually made by the user. When the consistency model is established, consistency testing of legacy data can be performed automatically. In our case there is one legacy data source with information about buildings and one about apartments. Consistency testing results in a report revealing which legacy data that do not fulfil the consistency requirements.
The association (Fig. 12) between `Apartment` and `Building`, including the attached invariant expressed in OCL, constitute consistency requirement two. When testing is performed on the legacy data, a link is created between an `Apartment` and a `Building` instance if the invariant is fulfilled; if the multiplicity on the association is broken, this is reported in the consistency report.
Consistency requirement one is specified with help of class `ConsistencyApartmentBuilding`, property `cApartmentCount` and its attached invariant. During testing instances of type `ConsistencyApartmentBuilding` are created and linked to `Building` instances; slot `cApartmentCount` will be set to the value that fulfils the invariant; if the value is `false` then a consistency violation has occurred and will be reported. Note that links between `Building` and `Apartment` instances are traversed when the values of `cApartmentCount` slots are set. From this example we can understand that standard OCL-statements are used as production rules when the consistency model is being automatically instantiated.

4.2 Implementation in SIWA

In Siwa the consistency model can be seen as an instance of the declarative domain specific language described by the metamodel given in Fig. 13.

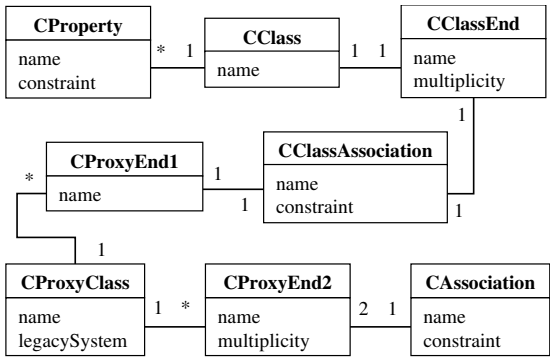

Figure 13: Consistency modelling metamodel.

In brief: The legacy classes `Building` and `Apartment` have *proxy classes* to represent them in the consistency model; a proxy class is an instance of `CProxyClass`. Class `ConsistencyApartmentBuilding` (Fig. 13) is an instance of `CClass`; its property `cApartmentCount` is an

instance of `CProperty`, where the value of slot `constraint` is the text:

cApartmentCount = (building.apartment→size() = building.apartmentCount)

The association between `ConsistencyApartmentBuilding` and `Building` is represented as an instance of `CClassAssociation` going between the building proxy class and `ConsistencyApartmentBuilding`.
The association between `Building` and `Apartment` is represented as an instance of `CAssociation` going between the two proxy classes; the value of slot `constraint` for this instance is the text:

apartment.bId = building.bId

Fig. 14 gives an overview of the complete architecture. The lowest level can be seen as one contiguous model composed of legacy data and a consistency model instance. The legacy models are on the other hand not changed - the borders towards the consistency model can be extracted automatically. From a model management point of view this is considered an advantage since it gives few models to manage (remember that the lowest level can be produced automatically at will).

The format of this article does not give room for presenting a complete picture of how Siwa solves the problem at hand, but below is some information about how the consistency model instance is automatically created by a semantic engine.
A semantic engine is a Siwa model that exhibits behavior. A special type of semantic engines can be attached to borders, they are called *border engines* and are typically involved in instantiation. For simplicity we can assumed that such engines are programmed in Java since this is our implementation language. The *metamodeller* has made a border engine and attached it to border `L4` (Fig. 14). Border `U4` and `L6` is created by the engine when the modeller decides to make a consistency model. The engine is also attaching a premade border engine at `L6`, it is this engine that automatically produces the consistency model instance and the consistency report when triggered. This last engine is an adapted implementation of the algorithm presented in (Nytun and Jensen, 2003).

This way of establishing the semantics can be seen as a specific way of implementing *deep characterization*; some see deep characterization as a natural part of metamodelling (Kühne, 2005).

5 SUMMARY AND RESEARCH DIRECTIONS

The MATER model presented in this article represent our understanding of what we mean by a multilevel model architecture; we have tried to make an explicit representation of all elements that constitutes such an architecture. Complex and advanced concepts can then be built in a natural way by combining these defined building blocks.

MATER can be seen as a metamodel by itself, but we choose to see it as the physical carrier for multilevel modelling architectures. This view allow us to specify top models as we see needed, e.g. a top model that gives properties separate existence.

The following is a list of features that in our view makes Siwa a promising and unique framework:

- It is not strongly coupled to the instantiation found in its implementation language, this allows a model to have several type models each offering different and useful information about the model.
- It is extremely generic which makes it adaptable to many different modeling needs, e.g. it might allow separate existence of properties.
- It is possible to have incomplete architectures, e.g. XML documents might be loaded for analysis, and then a model might be produced automatically (T. Gjøsæter and J. P. Nytun and A. Prinz and M. Snaprud and M. S. Tveit, 2006). This is typically not possible in other frameworks due to their strong coupling to instantiation in the selected implementation language.

Parts of our framework are already implemented in the Eclipse framework (d'Anjou et al., 2004). The first prototype is implemented by defining the MATER model as a UML model in Eclipse and from this we create an EMF Model; this looks like a trick since we end up with having all the Siwa model levels at one EMF level (Prinz et al., 2006), but it gives us a jump-start and it automatically produces a lot of useful code.

ACKNOWLEDGEMENTS

Thanks to Andreas Prinz, Vladimir Oleshchuk, Christian S. Jensen, Birger Møller-Pedersen and Arne Maus for continuous inspiration and valuable discussions.

REFERENCES

Atkinson, C. and Kühne, T. (2000). Strict Profiles: Why and How. In *UML 2000 - The Unified Modeling Language, Advancing the Standard*, volume 1939 of *Lecture Notes in Computer Science*. Springer.

Atkinson, C. and Kühne, T. (2001). The Essence of Multilevel Metamodeling. In *UML 2001 - The Unified Modeling Language: Modeling Languages and Applications*, volume 2185 of *Lecture Notes in Computer Science*. Springer.

Atkinson, C. and Kühne, T. (2002). Rearchitecting the UML infrastructure. *ACM Transactions on Computer Systems (TOCS),*, 12(4):290–321.

d'Anjou, J., Fairbrother, S., Kehn, D., Kellermann, J., and McCarthy, P. (2004). *The Java Developer's Guide to Eclipse*. Addison-Wesley.

EMF (2006). EMF, the Eclipse Modelling Framework: Available at: `http://www.eclipse.org/emf`.

Favre, J. (2004a). Foundations of meta-pyramids: Languages vs metamodels. Available at: `http://www.citeseer.ist.psu.edu /722867.html`.

Favre, J. (2004b). Foundations of model (driven) (reverse) engineering - episode i: Story of the fidus papyrus and the solarus. Available at: `http://www.citeseer.ist.psu.edu/ favre04foundations.html`.

Favre, J.-M. (2005). Megamodelling and etymology. In *Dagstuhl Seminar 05161 on Transformation Techniques in Software Engineering*. Available at: `http://www-adele.imag.fr/ jmfavre`.

Fowler, M. (2005). Language workbenches: The killer-app for domain specific languages? Available at: `http://www.martinfowler.com/articles/ languageWorkbench.html`.

Greenfield, J., Keith Short, w. c. b. S. C., and Kent, S. (2004). *Software Factories: Assembling Applications with Patterns, Frameworks, Models & Tools*. John Wiley & Sons.

J. Bézivin, F. Jouault, P. V. (2004). On the need of megamodels. In *OOPSLA, 2004*.

Kühne, T. (2005). What is a model? In Bézivin, J. and Heckel, R., editors, *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings. Available at: `http://drops.dagstuhl.de/opus /volltexte/2005/23`.

Marcos Didonet Del Fabro, Jean Bézivin, F. J. E. B. G. G. (2005a). AMW: a generic model weaver. In *Proceedings of the 1re Journe sur l'Ingnierie Dirige par les Modles (IDM05)*. Available at: `http://www.sciences.univ-nantes.fr /lina/atl/publications/`.

Marcos Didonet Del Fabro, Jean Bézivin, F. J. P. V. (2005b). Applying generic model management to data mapping. In *Proceedings of the Journes Bases de Donnes Avances (BDA05)*. Available at: `http://www.sciences.univ-nantes.fr /lina/atl/publications/`.

Marcus Alanen, I. P. (2004). The Coral Modelling Framework. In Koskimies, K., Kuzniarz, L., Lilius, J., and Porres, I., editors, *Proc. of the 2nd Nordic Workshop on the Unified Modeling Language NWUML'2004*. Turku Centre for Computer Science, Finland.

Martin Gogolla, Jean-Marie Favre, F. B. (2005). On squeezing m0, m1, m2, and m3 into a single object diagram. In *Workshop on Tool Support for OCL and Related Formalisms - Needs and Trends OCL at Models 2005*. Available at: `http://www-adele.imag.fr/ jmfavre`

Metacase (2006). MetaEdit+. Available at: `http://www.metacase.com/`

MPS (2006). Meta programming system. Available at: http://www.jetbrains.com/mps/.

Nytun, J. P. and Jensen, C. S. (2003). Modeling and Testing Legacy Data Consistency Requirements. In *UML 2003 - The Unified Modeling Language: Modeling Languages and Applications*, volume 2863 of *Lecture Notes in Computer Science*, pages 341–355. Springer.

Nytun, J. P., Prinz, A., and Kunert, A. (2004). Representation of levels and instantiation in a metamodelling environment. *NWUML 2004*.

OMG (2003). *Model Driven Architecture Guide, Version 1.0.1*. Object Management Group. omg/03-06-01.

OMG Editor (2003). Revised Submission to OMG RFP ad/2003-04-07: Meta Object Facility (MOF) 2.0 Core Proposal. Available at `http://www.omg.org/docs/formal /06-01-01.pdf`.

OMG Editor (2006a). MOF Support for Semantic Structures, OMG RFP ad/2006-06-03. Available at: `http://www.omg.org/docs/ad /06-06-03.pdf`.

OMG Editor (2006b). *UML 2.0 Infrastructure Specification, OMG Document formal/05-07-05* . OMG Document. Available at: `http://www.omg.org`.

Prinz, A., Nytun, J. P., Chen, L., and Wei, S. (2006). Integration of MATER and EMF. In *Proc. of the 4th Nordic Workshop on the Unified Modeling Language NWUML'2006*. Available at: `http://osys.grm.hia.no/osys/archive /conferences/nwuml_06`.

S. Melnik, E. Rahm, P. A. B. (2003). Rondo: A programming platform for generic model management. In *In: SIGMOD*, pages 193–204.

T. Gjøsæter and J. P. Nytun and A. Prinz and M. Snaprud and M. S. Tveit (2006). Modelling accessibility constraints. In *Proc. of ICCHP*.

Tony Clark, Andy Evans, P. S. J. W. (2004). *Applied Metamodelling. A Foundation for Language Driven Development.* Xactium. Available at: `http://www.xactium.com`.

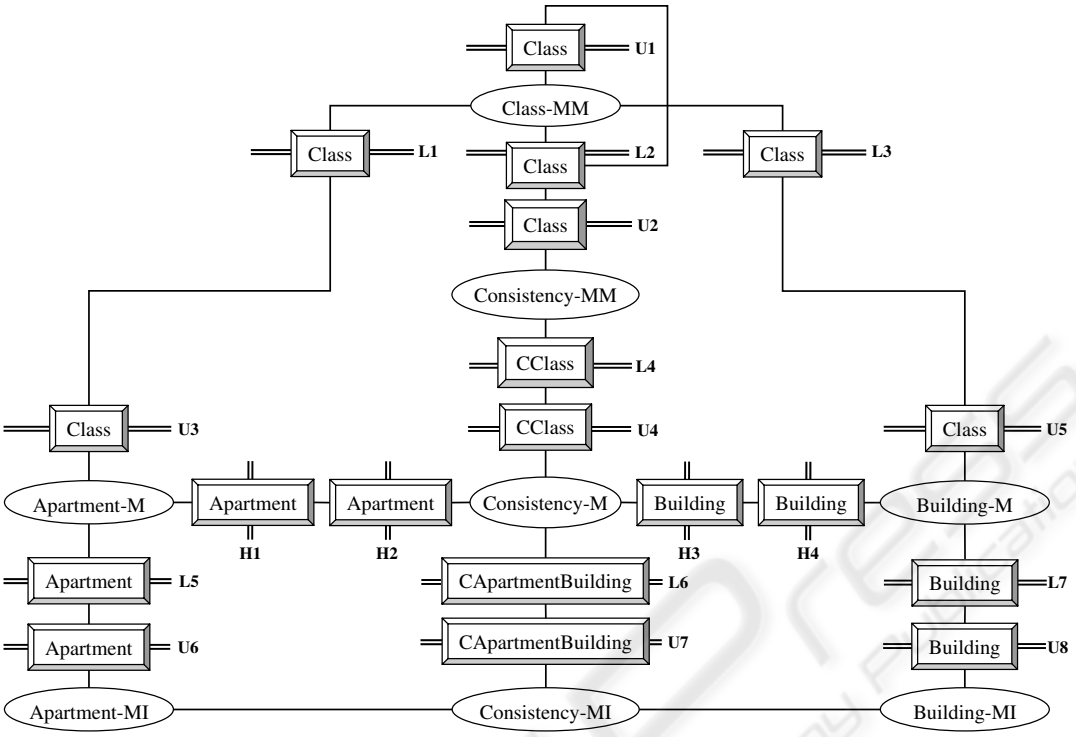

Figure 14: Architecture example: consistency modelling and testing.