Model Weaving and Pedagogy
Mapping Abstraction Levels in Instructional Design Languages

Esteban Loiseau, Pierre Laforcade and Sébastien Iksal
Laboratoire d'Informatique de l'Université du Maine,
Avenue Olivier Messiaen, Le Mans, France

Keywords: Model Weaving, MetaModeling, Model Transformation, Instructional Design.

Abstract: The GraphiT project aims at providing teachers with an operationalizable instructional design language. To provide compatibility with existing e-learning platforms, the produced learning scenarios have to comply with a platform specific metamodel. The issue addressed in this paper is about mapping high level pedagogical elements captured from teachers' practices to low-level platform features. We propose to use model weaving to specify the mappings between different abstraction levels. Weaving models can then be used to generate model transformations that will effectively perform modifications on the model at runtime.

1 INTRODUCTION

GraphiT is a research project funded by the French research agency. Its main goal is to study the possibilities and limits of the pedagogical expressiveness of operationalizable visual instructional design languages. The operationalizable property relates to the possibility to exploit learning scenarios in Learning Management Systems(Chou and Liu, 2005). The original approach of the project is to propose a platform dependant architecture to guarantee compatibility. The challenge is about providing a pedagogically relevant modeling language without adding features to the targeted platform. We propose to address it by building a multi-layered language where the first layer includes platform specific features and each subsequent layer is a higher abstraction level. The project methodology consists in exploring how Model Driven Engineering and more particularly Domain Specific Modeling(Kelly and Tolvanen, 2008) techniques and tools can be relevant and useful to achieve our goal. From a DSM point of view, the abstract syntax of the language will be specified using a metamodel in the Ecore format, the concrete syntax will make use of the Graphical Modeling Framework from the Eclipse Modeling Project.

2 RESEARCH CONTEXT

2.1 Visual Instructional Design Languages

Educational Modeling Languages (EML), were the first generation of languages to allow formalization of learning scenarios. IMS-LD (Botturi and Stubbs, 2008) is the *de facto* standard of this domain, but has two main issues: it is mainly a machine understandable language (XML format) thus cannot be directly used by teachers, and it is currently not compatible with the main e-learning platforms (Moodle, Ganesha...). *Visual Instructional Design Languages* (VIDL) address the first issue by providing teachers with a graphical notation (for example Flexo (Dodero et al., 2010)) but the platform full compatibility issue has yet to be tackled.

Previous works tried to achieve LMS compatibility through model transformations (Burgos et al., 2007)(Abdallah et al., 2008) but suffered semantic loss in the process. This issue appeared because the instructional design language was designed independently of the operationalizing system. By designing our instructional design language with platform compatibility in mind in the first place, we want to avoid these semantic losses.

In this context, we aim at specifying a new VIDL based on teachers needs and practices while providing Learning Management System compatibility. With regards to the first point, we gathered information

Loiseau E., Laforcade P. and Iksal S..
Model Weaving and Pedagogy - Mapping Abstraction Levels in Instructional Design Languages.
DOI: 10.5220/0005003600810086
In *Proceedings of the 9th International Conference on Software Paradigm Trends* (ICSOFT-PT-2014), pages 81-86
ISBN: 978-989-758-037-6

from interviews and surveys from local teachers community. They expressed the need for a graphical tool and language to design their learning scenarios, and being able to import it on the Moodle platform.

2.2 Moodle

Moodle is one of the most wide spread Learning Management System (LMS) in academic institutions. Teachers use this platform to set up fully distant courses or as an additional medium to teach in classrooms. Moodle is also the main e-learning platform deployed in our university making it the priority target of our system.

To achieve platform compatibility we rely on a platform specific instructional design metamodel (Abedmouleh et al., 2012). This metamodel includes every pedagogically relevant concept found through an identification process including database and user interfaces analysis. An API to import scenarios was developed, based on the XSD schema extracted from the previous metamodel. The API concretely parses the XML based scenario and populates the database according to it.

2.3 A Need for Mapping

The teachers we interviewed expressed some difficulties trying to set-up their previously designed learning scenarios on the LMS. Each teachers community will often have its own "pedagogical language", as well as the platform. This "translation" issue can be tackled by mapping each concept from one language to one or many concepts from the other. In this study we focus on one community (local teachers) and one platform (Moodle).

In fact, platform features can be seen as *tools* and each tool can be used to implement many pedagogical activities that teachers are used to work with. This first level of abstraction (pedagogical activity from platform tool) was identified in a previous study (Loiseau and Laforcade, 2013). This study identified the relevance of one specific approach to model this abstraction: extending the platform metamodel to include pedagogical activities from teachers practices.

In our current work we aim at generalizing this abstraction technique to several levels. The multiplicity of these abstraction layers makes the mapping task even more complex. The more obvious way to map from one level to another is using model transformation, but considering the number of transformations to write and the complexity of each mapping, we had to find a solution to reduce development costs.

2.4 Mapping examples

To illustrate our approach, we will use two examples of mapping, simplified versions of real case pedagogical elements.

The first one is a pedagogical activity (a specific use of one or many platform features): to complete the *Write A Report* activity, students have to write a text relating a real world activity they experienced, depending on the course context. Several Moodle tools could support this activity, but each one has specific constraints, for example:

- Does the text have to be written online?
- Is it a group work (collaborative)?
- Does the text have to be written in one time or is it more like a log (iterative)?

These properties depend on the pedagogical activity the teacher wants to set up, and the appropriate tool to use depends on these properties (**conditional mapping**). The decision table 1, explain how to choose the right tool. Note that two options rely on the same feature: *assignment*. The difference is ate the settings of these tools, on the platform when creating an assignment you have to set many parameters including whether students have to upload a file or write a text online to complete the assignment. These parameter settings have to be taken into account when designing the mapping (**parameter settings**).

The second example is the *Debate*, it is a pedagogical pattern that will be mapped as following:

- Debate
  - Label (platform tool)
  - Exchange (pedagogical activity)
    - Chat (if synchronous)
    - Forum (if not)

The *Debate* will be mapped to an *Exchange* activity, which in turn will be mapped to a *Chat* or a *Forum* depending on the synchronous property. This example illustrates once again the conditional mapping, but also highlights the need to **mix abstraction levels**: *Debate* is also mapped to a *Label*, an element from an even lower level.

3 MODEL WEAVING

Model weaving can be defined as "the operation for setting fine-grained relationships between models or metamodels and executing operations on them based on the semantics of the weaving associations specifically defined for the considered application domain."(Di Ruscio, 2007)

Table 1: Write A Report mapping decision table (/ means either yes or no).

	Journal	Wiki	Assignment (file upload)	Assignment (online text)
Online	Y	Y	N	Y
Collaborative	N	Y	/	N
Iterative	Y	/	/	N

Model weaving produce a model, called **weaving model**, linking several other models, called **woven models**. The semantic of the **weaving associations**, also referred as links or mappings, depends on the application domain, but every type of weaving association as to be defined in a **weaving metamodel**.

The model weaving task can either be achieved manually, by a domain expert through a weaving model editor (or model weaver), or automatically using model matching algorithms.

In an example model weaving use case, the user will weave two metamodels (source and target), then execute a High Order Transformation on the weaving model to produce a model transformation. This transformation can then be applied to a source model to obtain a model conforming to the targeted metamodel. In this case, the weaving model captures the transformation semantics.

3.1 Weave to Map

In our use case, users will design their learning scenario at a specific level of abstraction then run a model transformation to add the corresponding elements from a lower level to the global model. Running these transformations at each level will refine the model to the point where every element is linked to elements from the lowest level (platform tools).

The weaving model will capture the mappings between elements from different abstraction layers, keeping track of what become what. This model will be defined together with pedagogical engineers and teachers from the targeted community to ensure its relevance regarding the pedagogical aspects.

Applied to our instructional design issues, the model weaving process described in 3 can be used to automate the generation of model transformations, thus reducing the cost induced by their development.

4 MODEL WEAVERS

In the following subsections, we present two frameworks to achieve model weaving. The first one was developed in an academic research context and the second one is an adhoc use of several existing languages and tools from an Eclipse foundation project.

4.1 Atlas Model Weaver

Atlas Model Weaver (AMW) (Didonet Del Fabro and Valduriez, 2009) is a model weaving framework developed by the AtlanMod research team. It is part of the ATLAS Model Management Architecture (AMMA) platform.

The framework provides a reflexive weaving model editor, a base weaving metamodel and numerous use cases and examples: matching transformations, semi-automatic generation of weaving models...

In a typical model weaving application, users will extend the basic weaving metamodel with its own weaving associations semantic in KM3 (*Kernel Meta Meta Model*) files; write a High Order Transformation (HOT) in ATL(*ATL Transformation Language*); define the weaving model with a domain expert and execute the HOT to generate the final ATL model transformation.

AMW seems to be an appropriate model weaving framework for our use case, as it fulfills every requirements we had. Unfortunately the project is not supported anymore and several of its plugins depend on outdated components of the Eclipse platform and old versions of the Eclipse Modeling Framework.

4.2 Using Eclipse Epsilon to Achieve Model Weaving

Epsilon (Paige et al., 2009) is a project supported by the Eclipse foundation. It features multiple languages and tools to perform specific tasks on models: model migration, model merging, Model-to-Model and Model-to-Text transformations, model validation, model comparison... Epsilon has a wide and active community of users and developers and has a deep compatibility with other frameworks from the Eclipse Modeling Project.

Combining several languages and tools from the Epsilon project, we were able to recreate a model weaving toolchain similar to the AMW one while benefiting from up to date compatibility with Eclipse platform and modeling plugins.

One of the tools provided by Epsilon is ModeLink, it is a 3-pane EMF-compatible model editor. The main feature of ModeLink is its ability to

capture links between different models using EMF cross-source references. In combination with Exeed, a reflective customizable model editor from Epsilon, ModeLink can be used to weave heterogeneous models. Setting up the woven models on the left and right panes and the weaving model in the center, one can simply obtain a weaving model editor with no constraints on the weaving metamodel.

As writing High Order Transformation is a critical and complex task, we chose to take a different approach on generating the final model transformations. Using Epsilon model-to-text transformation language *EGL* (Rose et al., 2008) (Epsilon Generation Language) one can generate any text file, including model transformations. *EGL* is a template based language, similar to PHP or ASP, relying on the multi-purpose language EOL (Epsilon Object Language). Considering that each transformation has a similar structure, in our case, writing a template is an easier task: write an example model-to-model transformation manually, keep the fixed section in the template and get variable ones from the input weaving model.

Our application differs from traditional use cases of model transformation: the transformation has to be run while the model is being edited, the source and target models are the same and the source elements should be preserved by the transformation. To avoid unnecessary copying of source elements in transformation rules, the simplest way to go was to implement rule body as an operation and run it as an *EOL* program modifying the currently edited model.

5 MODEL WEAVING APPLIED

We propose to use the technical framework presented in 4.2, to provide users with a ”mapping editor” (or weaving editor) and a ”mapping execution” solution. Figure 2 summarize the architecture of our proposition. Unlike the learning scenario editor, our weaving editor does not directly target teachers as main users, as it requires computer skills to understand how the system operates.

To clarify the use of this mapping solution, here is a typical use scenario: prior to distributing the learning scenario editor to teachers, a pedagogical engineer along with a teacher from the targeted community define their own mapping semantics using the **weaving editor**. Once done, a computer engineer will run the **High Order Transformation** to generate **Model Transformations** based on the weaving model. The engineer will then integrate the transformations in the learning scenario editor. As explained before, the generated model transformations will be launched inside the learning scenario editor, at runtime, to operate the mappings previously defined in the weaving model.

5.1 Weaving Metamodel

In order to define mappings for our instructional design language, we must first specify a weaving metamodel. The weaving metamodel has to be generic enough to capture every mapping we already know (fulfilling the requirements presented in 2.4) and potential evolutions and additions. The figure 1 illustrate our weaving metamodel proposition.

The *Weaving Model* is the root of the model and contains many *Bindings*. Each *Binding* represents the mapping of one element from the language to one or many *Targets*. Each *Target* may have a *Condition* controlling whether it must be added to the model or not. These *Conditions* relate to the value of the source element attributes and can be composed with logical operators. Attributes of the *Target* element can be set to constant values (*SetAttr*) or retrieved from source attributes (*BindAttr*). Again, setting of these attributes can be subject to conditions. Note that, since we are weaving metamodels, the weaving metamodel only references EClasses.

5.2 Weaving Model Example

Figure 3 is a screenshot of a ModeLink editor showing the mappings of the examples from section 2.4. These mappings were modeled using the metamodel presented in 5.1. The central pane shows the weaving model, with custom labels defined with annotations in the weaving metamodel. The left and right panels (not shown here) display the metamodel of the instructional design language. By right-clicking on the Weaving Model element, users can easily add more bindings and set the source or target elements and attributes by dragging and droping them from the left and right panes.

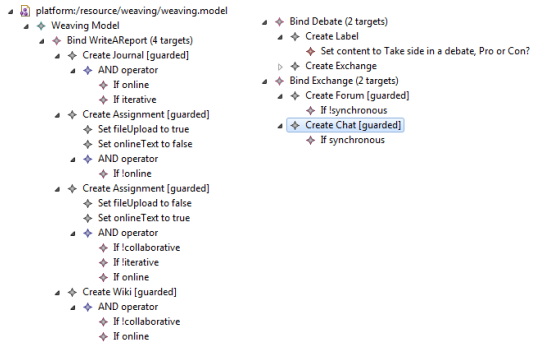

Figure 3: Weaving model example.

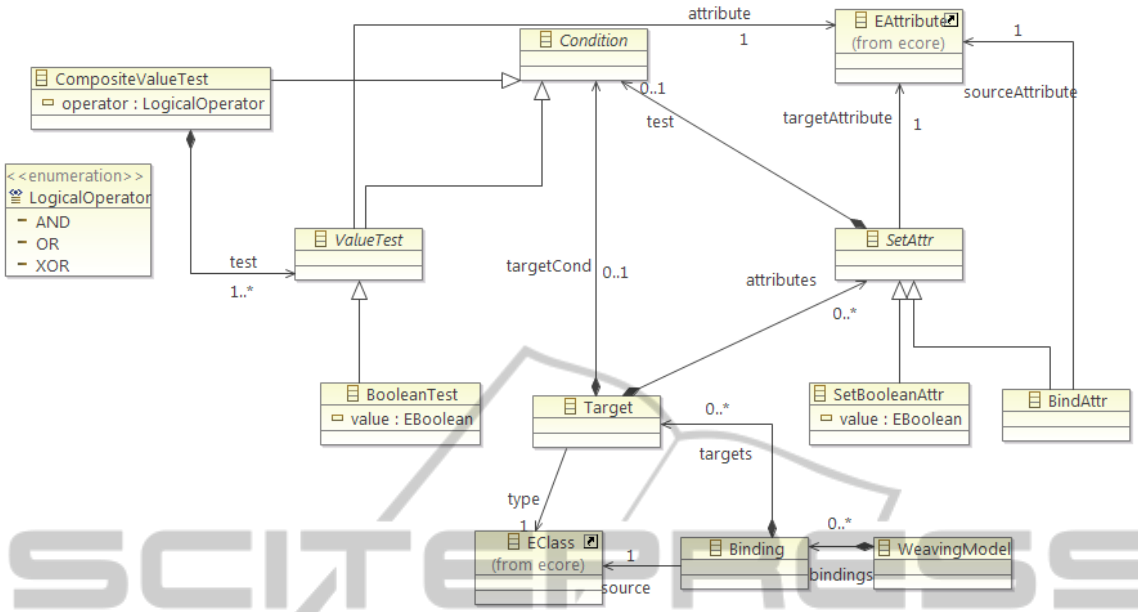

Figure 1: Weaving metamodel.

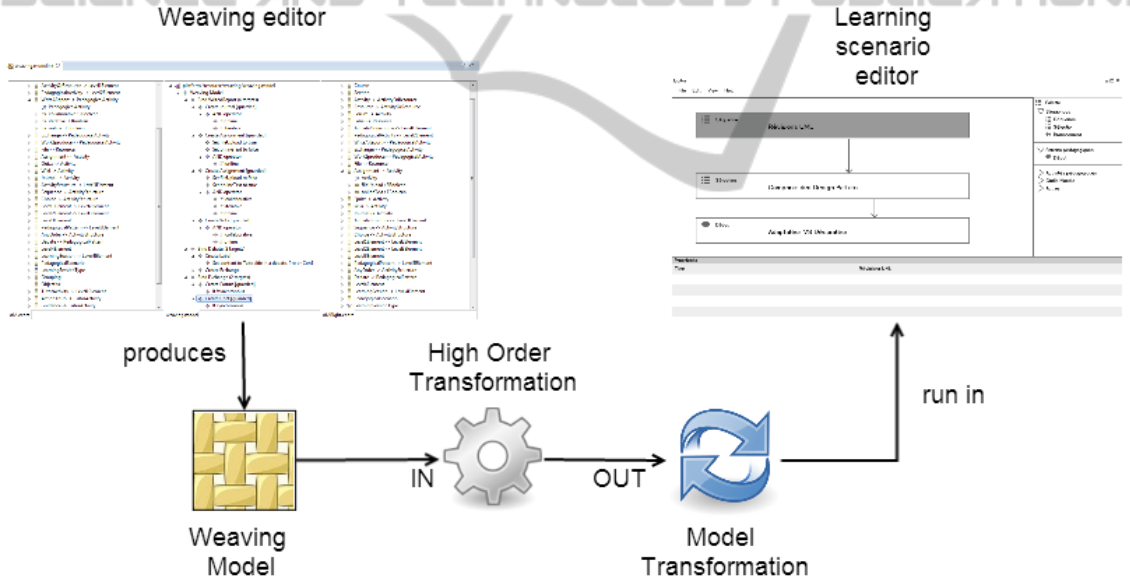

Figure 2: Global architecture.

5.3 Generative Template

The generative template, expressed in EGL, replaces the High Order Transformation as used in the AMW framework. Code between [% tags is executable and will be replace by the result of its evaluation in the generated transformation. An excerpt of the template is presented in figure 4.

6 CONCLUSION

We proposed a generic weaving metamodel, a high order model-to-text transformation, and a specific weaving model to capture mappings of pedagogical elements from an LMS centered instructional design language. We used an innovative model weaving environment making use of existing open-source languages and tools. We validated the expressiveness of our weaving metamodel by testing it on real mapping

```
[%
import "./hotOperations.eol";
var sourceClassName:String = binding.source.getName();
var sourceVarName:String = sourceClassName.firstToLowerCase();
%]

operation source![%=sourceClassName%] addMapping(element:source!Level3Element) {
    if(element.isKindOf(source!Level1Element)) {
        self.mapedL1Elements.add(element);
    }
    else if(element.isKindOf(source!Level2Element)) {
        self.mapedL2Elements.add(element);
    }
    else {
        self.mapedL3Elements.add(element);
    }
}

operation source![%=sourceClassName%] bind[%=sourceClassName%]() {

    [% for(targetElement:Target in binding.targets) {
        var cond:Boolean = targetElement.targetCond.isDefined();
        var targetClassName:String = targetElement.type.getName();
        var targetVarName:String = targetClassName.firstToLowerCase();
        if(cond) {%]
            if([%=targetElement.targetCond.formatCondition()%]){
        [%}%]
                var [%=targetVarName%]:[%=targetClassName%] = new source![%=targetClassName%];
                self.addMapping([%=targetVarName%]);
        [%if(cond){%]
            }
        [%}
    }%]
}
```

Figure 4: Generative template.

provided by teachers or from literature: from a textual description of how to implement a particular activity or pedagogical pattern on Moodle, we tried to design an equivalent weaving model using our solution. The few missing mapping features were then added to the weaving metamodel.

6.1 Further Work

The learning scenario editor has to be tweaked in order to automatically run the generated transformations, also high level elements still remain in the model, breaking compatibility with the Moodle import API. To circumvent this last issue, we will have to write a global ”cleaning” transformation that will delete these elements. Another issue is about structure blocks: teachers can structure their learning scenario using sequences of activities but since mapping transformations are run just-in-time and the high level elements are kept, the global transformation will also have to restore the general structure of the scenario, replacing the sequences with their Moodle counterparts (sections, indented content etc.). Unfortunatly we could not yet try the model weaver in real conditions with real users, as they lack interest until a suitable version of the learning scenario editor is released.

REFERENCES

Abdallah, F., Toffolon, C., and Warin, B. (2008). Models transformation to implement a project-based collaborative learning (pbcl) scenario : Moodle case study. In *8th IEEE International Conference on Advanced Learning Technologies (ICALT 08)*, Santander (Spain).

Abedmouleh, A., Oubahssi, L., Laforcade, P., and Choquet, C. . (2012). Expressing the implicit instructional design language embedded in an lms: motivations and process. In *Computers and Advanced Technology in Education*, Naples (Italie).

Botturi, L. and Stubbs, S. (2008). *Handbook of Visual Languages for Instructional Design: Theories and Practices*. Gale virtual reference library. Information Science Reference.

Burgos, D., Tattersall, C., Dougiamas, M., Vogten, H., and Koper, R. (2007). A first step mapping ims learning design and moodle. *Journal of Universal Computer Science*, 13(7):924–931.

Chou, S.-W. and Liu, C.-H. (2005). Learning effectiveness in a web-based virtual learning environment: a learner control perspective. *Journal of Computer Assisted Learning*, 21(1):65–76.

Di Ruscio, D. (2007). *Specification of Model Transformation and Weaving in Model Driven Engineering*. PhD thesis, University of L'Aquila.

Didonet Del Fabro, M. and Valduriez, P. (2009). Towards the efficient development of model transformations using model weaving and matching transformations. *Software & Systems Modeling*, 8(3):305–324.

Dodero, J. M., Álvaro Martínez del Val, and Torres, J. (2010). An extensible approach to visually editing adaptive learning activities and designs based on services. *Journal of Visual Languages & Computing*, 21(6):332–346. Special Issue on Visual Instructional Design Languages.

Kelly, S. and Tolvanen, J. (2008). *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley.

Loiseau, E. and Laforcade, P. (2013). Specification of learning management system-centered graphical instructional design languages - a dsm experimentation about the moodle platform. In *ICSOFT'13*, Reykjavik (Iceland).

Paige, R. F., Kolovos, D. S., Rose, L. M., Drivalos, N., and Polack, F. A. C. (2009). The design of a conceptual framework and technical infrastructure for model management language engineering. In *Proceedings of the 2009 14th IEEE International Conference on Engineering of Complex Computer Systems*, ICECCS '09, pages 162–171, Washington, DC, USA. IEEE Computer Society.

Rose, L., Paige, R., Kolovos, D., and Polack, F. (2008). The epsilon generation language. In Schieferdecker, I. and Hartman, A., editors, *Model Driven Architecture – Foundations and Applications*, volume 5095 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg.