Systematic Evolution of WebML Models by Coupled Transformations

Manuel Wimmer, Nathalie Moreno, and Antonio Vallecillo

Universidad de Málaga, Spain
{mw,moreno,av}@lcc.uma.es

Abstract. Model-driven Web Engineering is an effective approach for improving the development of Web applications by providing appropriate abstraction mechanisms and different viewpoints. However, maintaining existing Web models still presents some significant research challenges. In particular, maintenance and evolution tasks are based on fine-grained atomic changes, and there is no automated reconciliation support for change propagation among viewpoints. In this paper we present an approach based on coupled transformations to ease the evolution of content models and the corresponding reconciliation of dependent hypertext models. The approach is illustrated by using the well-known *Extract-Class* refactoring for WebML models.

1 Introduction

Model-driven Web Engineering (MDWE) [13] is an effective approach to Web application development that uses models, metamodels, and model transformation as key elements of the development process. It incorporates a higher level of abstraction in the specification of systems guided by the *separation of concerns principle* using *viewpoints* that allows the (semi)-automated derivation of the final implementation code from platform-independent multi-viewpoint specifications. In this sense, existing Web engineering approaches such as WebML [1] and UWE [11] to name just a few (for a survey, cf., [15]) match the MDWE principles.

Most MDWE approaches identify three key viewpoints for the design of Web applications: *content*, *hypertext*, and *presentation*. Although these viewpoints are separately specified and developed, they are not completely independent. For instance, the hypertext models reference elements defined in content models, because they describe how to navigate through the content model. Maintaining manually these references and the consistency between the different viewpoints is a cumbersome task, for which there is little automated support. Furthermore, the integration and synchronization of multi-viewpoint systems is an open issue, not only in MDWE but also in other application fields of model-driven engineering in general [7].

The maintenance and evolution of Web models in the majority of MDWE approaches is currently hampered by two main shortcomings: (i) *missing evolution support*, since changes are applied and identified at very low level of abstraction (basically as atomic changes to the model elements such us additions, deletions, and updates); (ii) *missing reconciliation support*, since the propagation of changes among viewpoints is

currently difficult and cumbersome because the reconciliation is also achieved by manually applying atomic changes.

To tackle these shortcomings, we propose to manage the evolution of content models by using *coarse-grained changes*, which are specified as *model transformations*. To reason about the impact of the coarse-grained content model changes, we specify the changes by stating not only the structural transformation of content models, but also the implications for their instances, i.e., the data of the Web application, by applying *coupled model transformations* [12] for the instance models. This approach allows describing in a precise way the semantics of the coarse-grained changes—which is a prerequisite for reasoning about the reconciliation of dependent hypertext models.

Based on the coarse-grained content model changes, we present a catalogue of reconciliation patterns for hypertext models specified as coupled model transformations. As an example, we present how hypertext models have to co-evolve when a content model evolve by an *ExtractClass* refactoring. This catalogue of reconciliation patterns is based on the core modeling elements of Web modeling languages, which have been jointly developed in the MDWEnet initiative [14,18]. For demonstrating the proposed approach, we use WebML as selected MDWE protagonist. Although the presented (coupled) transformations are specific to WebML, the used modeling concepts are shared by the majority of MDWE approaches. Thus, they results are not limited to WebML but may be also transferred to other MDWE approaches. As a spin-off, during our investigations we explored some limitations of WebML for which we propose two extensions.

This paper is structured as follows. Section 2 briefly outlines WebML and introduces the running example used throughout the paper. Then, Section 3 presents our approach and Section 4 describes the catalogue of reconciliation patterns for hypertext models when *ExtractClass* refactorings have been applied on content model. Finally, Section 5 relates our work to similar approaches and Section 6 concludes.

2 Background: WebML By-Example

WebML describes Web applications with three viewpoints: *content* (data), *hypertext* (navigation between pages), and *presentation* (look&feel). The *content model* is specified using an Entity-Relationship model (or, equivalently, a simplified UML class diagram), comprising *classes*, *attributes*, *single inheritance*, and *binary relationships* as shown in the WebML metamodel (cf. Fig. 1). The front-end is specified using the *hypertext model*, which is structured into *pages*.

Pages are the basic interface containers: they can be structured in sub-pages and comprise content units. A *content unit* is defined as a component that publishes some content in a page; the published content can be extracted dynamically from the objects specified in the content model or specified statically in the hypertext model (e.g., an entry form consisting of multiple input fields). In addition to content units, WebML provides *operation units*, defined as components for executing commands (mostly on the database). Operation units, unlike content units, do not publish content and thus are positioned outside pages. Components (content and operation units) may have *input and output parameters* (e.g., the OID of the object to display or modify, etc.). Parameter passing is expressed as a side effect of navigation (values are transported from source

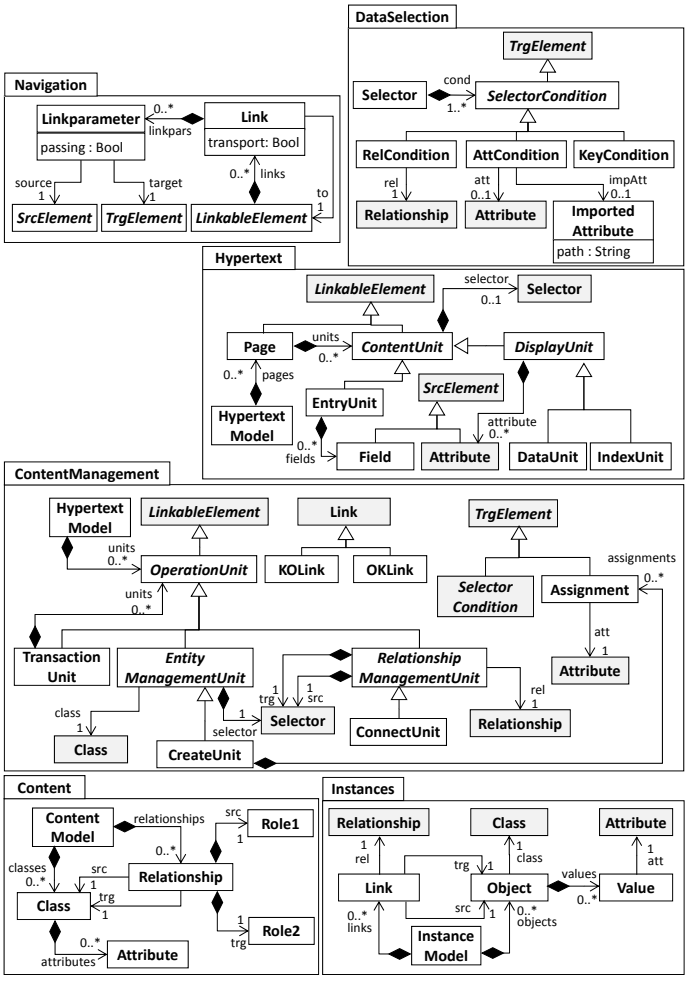

Fig. 1: Excerpt of the WebML Metamodel.

elements of link parameters to their targets): components are connected by *links*, which have a threefold purpose: enabling the user's navigation, supporting the passage of parameters, and triggering the execution of components. In particular, *OK links* and *KO links* are output links of operations, respectively followed after execution success or failure. How all these concepts are related to each other is illustrated in an excerpt of the metamodel shown in Fig. 1.

Running example: The Agenda system. At the beginning, this Web application was designed with the only goal of allowing users to maintain a simple list of contacts. Following the WebML methodology, the content and hypertext models were designed as shown in Fig. 2. Given the simplicity of our requirements, one class was enough to store the contacts' information and, based on it, the hypertext model was established, comprising a *DataUnit* and an *IndexUnit* for retrieving information from the content

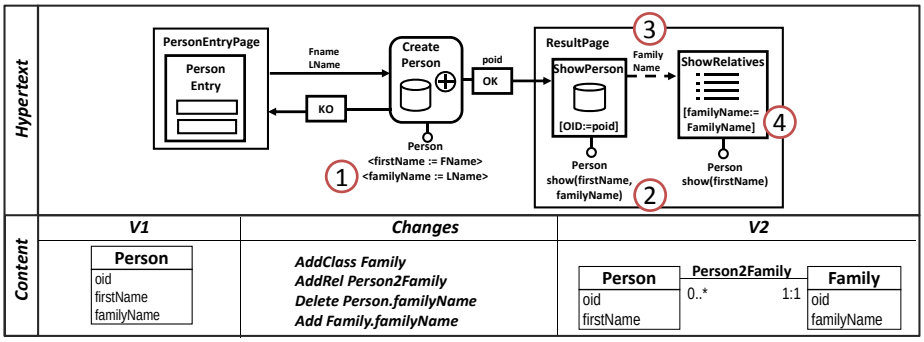

Fig. 2: Running Example: Content Model Evolution and Impact on Hypertext Model.

model as well a *CreateUnit* and an *EntryUnit* for inserting and storing information. However, the content model was later revised to add, among other changes, the *Family* class for grouping contacts based on their family ties. This meant to *extract* a class from the *Person* class, and to *move* the attribute *familyName* from *Person* to the new class.

When describing the changes as refactorings, this high-level of abstraction is the natural way in which modelers usually thinks and discuss about a system evolution. However, when these changes are detected by any of the existing model difference tools, what we obtain is a very large number of atomic changes that need to be applied to the individual model elements (*AddClass Family*, *AddRel Person2Family*, *Delete Person.familyName*, *Add Family.familyName*, etc.). Understanding and manipulating atomic changes to propagate them from one view to the rest can become a complex and brittle task. Just thinking about, e.g., the *ExtractClass* refactoring that we have previously mentioned. In order to guarantee that the hypertext model still works as before, the modeler has to adjust several elements in the hypertext model (around 22 atomic changes as we shall see later) for this small example because of four issues:

1. The *CreateUnit* has an assignment to the attribute *familyName* (cf. ① in Fig. 2) which is now no longer contained the class *Person*. In WebML, only attributes contained by the class which is referenced by the *CreateUnit* can be used in assignments. Furthermore, the *CreateUnit* is only able to produce a *Person* instance, but actually, also a *Family* instance is needed that is linked to the *Person* instance to populate the same information in the database for the given inputs of the *EntryUnit*.
2. The *DataUnit* shows two attributes, namely *firstname* and *familyname* (cf. ② in Fig. 2). However, as mentioned before, the attribute *familyName* is no longer available in the class *Person*. As for *CreateUnits*, also *DataUnits* can only use attributes which are directly contained by their referenced class.
3. The automatic transport link between the *DataUnit* and the *IndexUnit* comprises a *LinkParameter* transferring the *familyName* value (cf. ③ in Fig. 2) from the source unit to the target unit. However, this value is not accessible in the source unit.
4. A similar issue arises for the *SelectorCondition* of the *IndexUnit* which also accesses the moved attribute *familyName*.

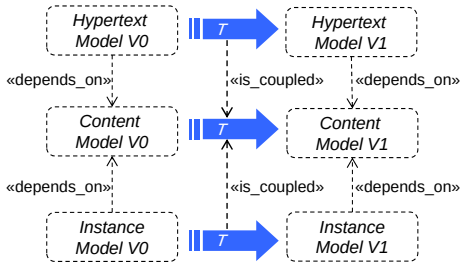

Fig. 3: Coupled Transformations for Web Model Evolution at a Glance.

3 Transformations for Web Model Evolution: An Overview

In WebML, the content model is the cornerstone around which all other views are articulated. This fact is not a particular feature of WebML, but shared by most modeling approaches for data-intensive Web applications. So, given its importance, we have focused our research on the evolution of Web application models when evolution is triggered by the content model.

Fig. 3 illustrates our proposed approach for the systematic evolution of Web models when coarse-grained content model changes are applied. While the upper area of this figure is concerned with the reconciliation of the changed content model and the initial hypertext model, the lower area is dealing with the co-evolution of the content model and its instances. So to speak, we have *initiator changes* on the content models expressed as *model transformations*, and *reconciliation changes* for the instance models and hypertext models expressed as *coupled model transformations* [12], which are transformations that involve multiple software artefacts, such that changes in one artefact trigger co-changes in other artefacts.

3.1 Coarse-grained content model changes as transformations

A transformation describing a coarse-grained change is much more than a *set of atomic changes*. In fact, its definition includes *pre-* and *post-conditions* which have to be fulfilled for an appropriate application. A natural way of implementing coarse-grained changes is by means of *in-place* transformations. As a matter of fact, the term *in-place transformations* stands for transformations rewriting a model, as opposed to producing a model from scratch which is done by *out-place* transformations.

In-place transformations can be described in many ways. Rule-based descriptions are elegant and easy to understand. Such descriptions have declarative model rewriting rules as their primitive building blocks. A rule consists of a *Left Hand Side* (LHS) pattern that is matched against a model. If a match is found, this pattern is updated, in the model, based on what is specified in the *Right Hand Side* (RHS) of the rule. Additionally, *Negative Application Condition* (NAC) patterns may be used, specifying which patterns should not be found in the model (match for non-existence) for applying the rule.

Coarse-grained changes such as refactorings are implemented by specifying its pre- and post-conditions as well as the actions that have to be executed for applying the

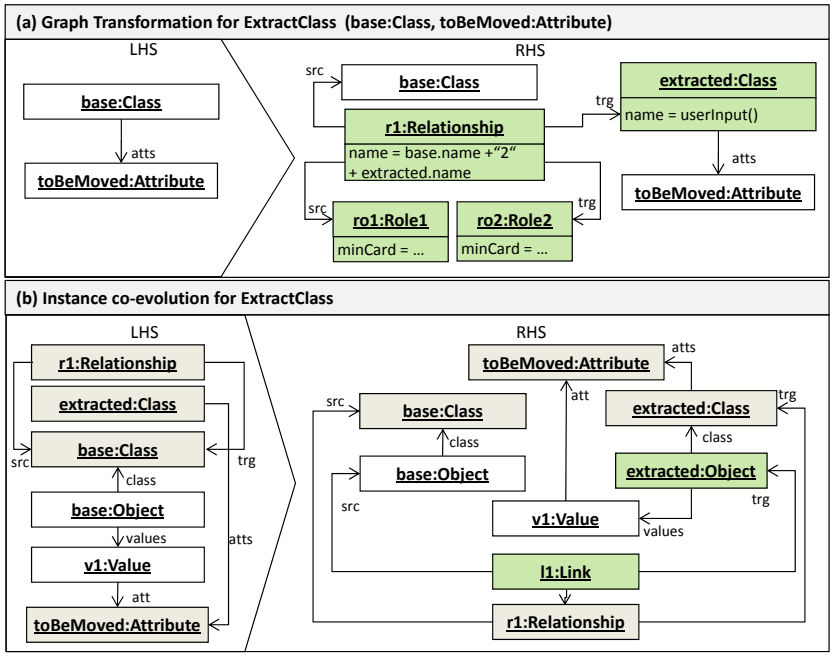

Fig. 4: Formalization of ExtractClass Refactoring

change. Most of them need also some input parameters that should be properly instantiated by the user. Let us go back to our example. The corresponding graph transformation rule for the refactoring *ExtractClass* is depicted in Fig. 4(a). Thereby, the LHS of the rule represents the pre-condition of the operation and the post-condition is specified in the RHS whereas actions that are going to carry out are implicitly defined in both sides. More precisely, the execution of a transformation rule produces the following effects: (i) all elements that only reside in the LHS are deleted; (ii) all elements that only exist in the RHS are added, and (iii) all elements that reside in both sides are preserved. To mark that an element in the RHS is equivalent to an element in the LHS, both elements must have the same identifier. Note that the *ExtractClass* refactoring requires that a class and an attribute are given as input by the user.

The graph transformation rules that describe the content model changes may be coupled with other rules which take care of the reconciliation of the existing instances for the content model (cf. Fig. 3). This coupling is a crucial aspect, because the semantics of the content model changes are described by the corresponding changes on the instances. For example, moving an attribute from one class to the other can have several meanings, depending on the intended behaviour on the instances: either the attribute is supposed to be deleted and then created (and therefore the values for the old attribute are lost, and the newly created attribute gets fresh values), or the attribute is supposed to be *moved* (and hence the values of the attribute should not be deleted but reused for the newly introduced attribute).

3.2 Instance reconciliation as coupled transformations

If instances of content models are again considered as models, transformations can be applied for their reconciliations. To represent instance models on a conceptual level, we reuse UML object diagrams for modeling *objects* (instances of classes), *values* (instances of attributes), and *links* (instances of relationships). Thus, we have included in the WebML metamodel a package for modeling instance models (Fig. 1).

Considering again the *ExtractClass* refactoring, expressing the effect at the instance level, a coupled transformation is needed. Fig. 4(b) shows the effect on the instance model as a transformation rule. For each object of the *base* class (which stands for an arbitrary class on which an *ExtractClass* refactoring has been applied), an additional object of the *extracted* class is created and linked to the *base* object. Finally, the value of the moved attribute is shifted from the *base* object to the *extracted* object.

The benefits of having a conceptual representation of the instance level evolution is twofold. First, the intend of the refactoring is concisely represented by stating the effects on the instances, thus we have the basis for reasoning on the impact of the change on the hypertext level. Second, the conceptual representation may be used to derive platform specific reconciliation rules, e.g., SQL-based migration rules for relational data, automatically.

3.3 Hypertext reconciliation as coupled transformations

It is likely that reconciliations in the hypertext models are necessary when the underlaying content model has been changed. In this sense, the hypertext model has to be reconciled to guarantee interaction requirements supported by the system before evolution.

Some effects that content model evolution implies on the hypertext model may be easily inferred by looking at broken correspondence links between hypertext and content model. Let us consider the *ExtractClass* refactoring. In the hypertext, all *Units* that reference the moved attribute (for applying any CRUD operation on it) need to be split into two in order to consider the new container of the attribute, i.e., the *Family* class. To preserve the system's initial navigation structure and behavior, added elements on the hypertext model must be properly linked by using suitable navigation links. In next section, we will explain in detail how coupled transformations are used to reconcile hypertext models with evolved content models.

4 Co-Evolution Patterns for WebML Hypertext Models

When propagating changes from content models to hypertext models, equivalence properties have to be preserved for the initial hypertext model (H) and the revised version (H') such that the observable behavior of the Web application is equivalent between H and H' from a user point of view. In particular, we have derived three equivalence properties which are directly related to the three core behavioral element types of hypertext model, namely *ContentUnit*, *OperationUnit*, and *Link* shown in Figure 1:

– **Amount of information per page**. The *content units* located in a *page* should display in total the same amount of information in H and H', i.e., the same attribute values have to be shown on the page before and after evolution for given input values.
– **Effects on the database**. Having a set of input values for a *operation unit* in H should have the same effect as having these input values for the corresponding sequence of *operation units* in H'. This means, when a *operation unit* in H is executed on the initial content model and the data is subsequently migrated to the new content model, it should lead to the same result as executing the corresponding sequence of *operation units* in H on the new content model.
– **Navigation paths**. If a node *b* is reachable from node *a* in H then node *b* has to be reachable from node *a* in H' with the same parameter values transported.

In the following, we present co-evolution patterns for reconciling hypertext models after a *ExtractClass* refactoring has been executed in the associated content model. The co-evolution patterns are described by recapturing the issue that has to be resolved in the hypertext model, the reconciliation strategy, and the corresponding graph transformation rule.

4.1 Rule 1: CreateUnit Reconciliation

Issue: A *CreateUnit* refers to a *Class* in the content model on which the *ExtractClass* refactoring has been executed. As a result, the moved attribute may be used in an assignment of the *CreateUnit*; a situation which does not represent a valid model structure in WebML. Furthermore, to preserve the operational semantics of the hypertext model, not only an instance of the base class has to be created, but also an instance of the extracted class linked to the instance of the base class is needed.

Reconciliation Strategy: In addition to the already existing *CreateUnit* for instantiating the base class, an additional *CreateUnit* for instantiating the extracted class and a *ConnectUnit* for linking instances of the base class and of the extracted class have to be introduced. Furthermore, to guarantee the same behavior as before the evolution, a *TransactionUnit* has to be introduced which contains all three operation units. This ensures that only when all three units are successfully executed, the complete information is populated in the database —which corresponds to behavior of the initial hypertext model where one *CreateUnit* is responsible for populating the complete information at once. Furthermore, the assignment of the attribute that has been moved to the extracted class has to be moved to the new *CreateUnit*.

Transformation Rule: The transformation rule[1] for co-evolving the hypertext models based on the mentioned adaptation strategy is illustrated in Fig. 5. The newly introduced elements in the hypertext model are shown in green background color. The content model elements are shown in gray background color. As is illustrated, additional *OperationUnits* connected by *OKLinks* are introduced to simulate the behavior of the single *CreateUnit* in the initial version. The *KOLink* is moved from the *CreateUnit* to the *TransactionUnit* which ensures if one single unit fails, the target of the *KOLink* is shown to the user. Finally, also the source of the initial *OKLink* is relinked to the last unit of the transaction.

[1] *LinkParameters* and *SelectorConditions* are not shown due to space limitations.

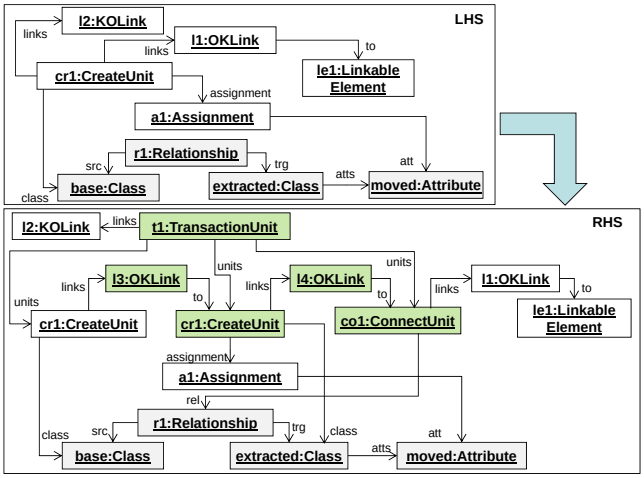

Fig. 5: Co-evolution pattern for *CreateUnits* affected by *ExtractClass* refactorings

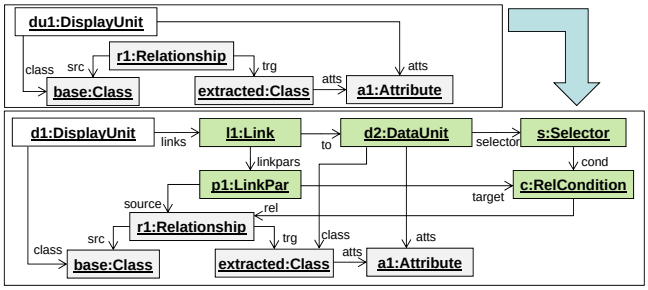

Fig. 6: Co-evolution pattern for *DisplayUnits* effected by *ExtractClass* refactorings

4.2 Rule 2: DisplayUnit Reconciliation

Issue: A *DisplayUnit* refers to a *Class* in the content model which has been effected by the *ExtractClass* refactoring and displays the attribute which has been moved to the extracted class. As for *CreateUnits*, a *DisplayUnit* can only refer to attributes which are directly contained by the referenced class.

Reconciliation Strategy: In order to display the value of the moved attribute, a *DataUnit* has to be introduced which is able to display the attribute, i.e., which refers to the extracted class. This means, also an additional *TransportLink* has to be created to navigate the relationship from the base class to the extracted class to find the appropriate instance which contains the value to display. The *DataUnit* shows the moved attribute and is included in the page containing the initial *DisplayUnit*.

Transformation Rule: The transformation rule for this strategy is shown in Fig. 6.

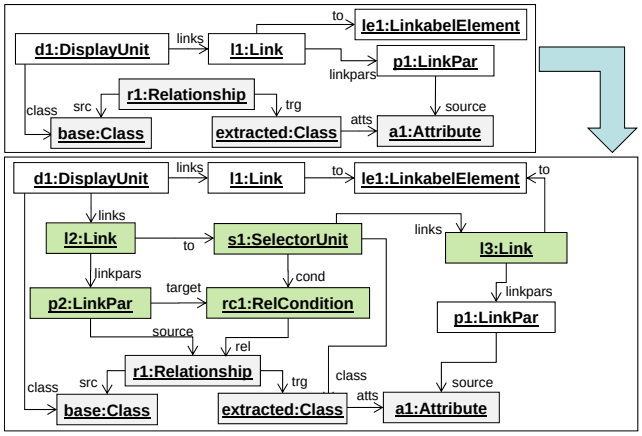

Fig. 7: Co-evolution pattern for *Source Elements* of *LinkParameters* effected by *Extract-Class* refactorings

4.3 Rule 3: LinkParameter.source Reconciliation

Issue: A *Link* may use an *Attribute* as a source element for a *LinkParameter* which is no longer accessible for the source of the *Link*, because it has been moved to the extracted class. Again, the same constraint applies that units cannot access elements outside their referenced classes.

Reconciliation Strategy: In order to transfer the necessary input for the target of the *Link*, a work-around using a so-called *SelectorUnit* is required. A *SelectorUnit* is used to access the attribute and transports the value of the attribute to the target of the *Link*, however, the processing of a *SelectorUnit* does not effect the user interface of the Web application. This additional unit is needed, because the initial source unit of the link is not able to access the moved attribute. But it is possible to access the extracted class by using the relationship between the base class and the extracted class, but it is not possible to access its features directly. Thus, the access of the moved attribute is delegated to the *SelectorUnit* which receives the extracted class instance from which it retrieves the requested attribute value.

Transformation Rule: The transformation rule for reconciling source elements of link parameter which are no longer accessible is shown in Fig. 7.

4.4 Rule 4: LinkParameter.target Reconciliation

Issue: A moved *Attribute* is used as a target element of a *LinkParameter* which is no longer accessible for the target of the *Link*, because it has been moved to the extracted class (inverse case to Rule 3). This case is typically concerned with *AttConditions* of *Selectors*, which act as target elements for *LinkParameters*.

Reconciliation Strategy: In order to the use again the target element for the *LinkParameter*, the *AttCondition* has to point to a so-called *ImportedAttribute* instead of normal *Attribute*. By using *ImportedAttributes* it is possible to access information

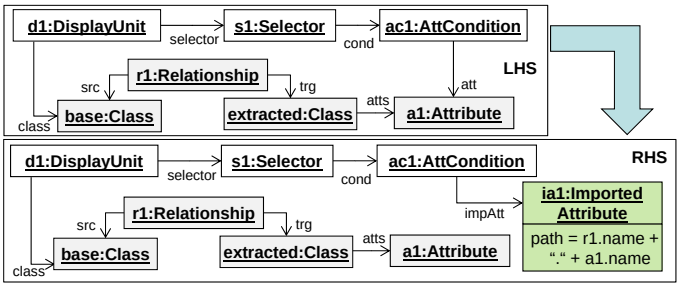

Fig. 8: Co-evolution pattern for *Target Elements* of *LinkParameters* effected by *Extract-Class* refactorings

outside the referred class. Thus, we employ this concept to access the moved *Attribute* by following the relationship from the base class to the extracted class.

Transformation Rule: As can be seen in Fig. 8, the link from the *AttCondition* to the moved *Attribute* is substituted by a link to an *ImportedAttribute*. In particular, the *path* to the moved attribute is calculated by concatenating the relationship name (*r1.name*) followed by the point operator (used to access the features of the target class) and the name of the moved attribute (*a1.name*).

4.5 Application to the Running Example

When applying the presented coupled transformation rules exhaustively (i.e., the model is rewritten until no further match can be found) on the running example, we end up with a hypertext model illustrated in Fig. 9. In particular, when the rules are applied in the order they are presented, the initial hypertext model is rewritten from left to right. First, the *CreateUnit* is rewritten by Rule 1 into a *TransactionUnit* covering the three *OperationUnits*. Second, the ShowPerson *DataUnit* is split by Rule 2 into two *DataUnits*, one visualizing the *firstName* attribute value and the other the *familyName* attribute value of the created person. Third, Rule 3 substitutes the *Link* between the ShowPerson *DataUnit* and the *IndexUnit* by one *Link* activating a *SelectorUnit* for retrieving the family instance for the transferred person instance, followed by another *Link* which is transferring the familyName attribute value from the *SelectorUnit* to the *IndexUnit*. Finally, the *AttCondition* of the *IndexUnit* is rewritten from a "standard" attribute to an *ImportedAttribute* (cf. *Person2Family.familyName*) by Rule 4.

4.6 Critical Discussion

The reconciled hypertext model allows to work with the new content model version in an equivalent way as the initial hypertext model worked with the initial content model w.r.t. the three stated properties in the beginning of this section. However, there are also some minor differences concerning the structure of the Web pages. Because it is not possible to use the notion of *ImportedAttributes* for showing attributes residing outside the classes referenced by *DisplayUnits*, some additional *DisplayUnits* have to be introduced in the hypertext model. This has an effect on the presentation models

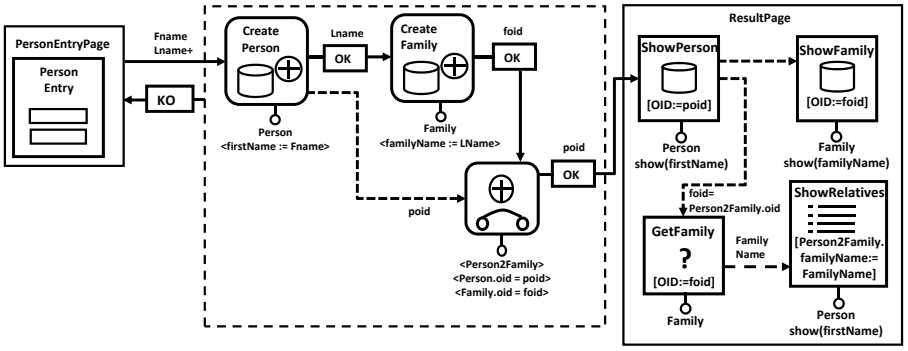

Fig. 9: Reconciled Hypertext Model of the Running Example

of the Web applications, and thus, on the user interfaces. For example, automated test may fail to access some information which is now visualized in a different place on the corresponding Web site.

As a consequence, the *ResultPage* in the reconciled hypertext model (cf. Fig. 9) is more verbose than the initial version because *ImportedAttributes* are not possible either for source elements of *LinkParameters* or for the shown attributes of *DisplayUnits*. Other Web modeling languages such as UWE [11] allow for *ImportedAttributes* for *DisplayUnits* by using some kind of expression language, similar to the one in WebML for defining *ImportedAttributes* for *SelectorConditions*.

When we assume that we have an enhanced modeling support in WebML, i.e., *ImportedAttributes* are also possible for *DisplayUnits* as well as for source elements of *LinkParameters*, the *ResultPage* would be expressible in a more concise manner following the initial page structure as shown in Fig. 10. Instead of using four units in the reconciled hypertext model, only two units—as in the initial hypertext model—are sufficient to work with the new content model version. Thus, the same structure of the Web page is guaranteed which also allows to reuse the presentation model of the initial hypertext version also for the reconciled version.

In addition, having this enhanced modeling support also leads to less complex coevolution patterns. In particular, *Rule 2* and *Rule 3* only have to substitute the links from the hypertext model elements to the moved attribute with an *ImportedAttribute*. Therefore, we propose the WebML metamodel to have also the possibility to use *ImportedAttributes* for *DisplayUnits* and for source elements of *LinkParameters*. By this not only the reconciliation rules and the resulting reconciled hypertext models are simpler, but also modeling Web applications in WebML from scratch may be enhanced by having such modeling support.

4.7 Implementation

We have implemented the presented approach by defining WebML models in the Eclipse Modeling Framework (EMF). For representing WebML models in EMF, we have developed an Ecore-based WebML metamodel. This opens the door for using transformation approaches available for EMF-based models. We selected the Eclipse Modeling

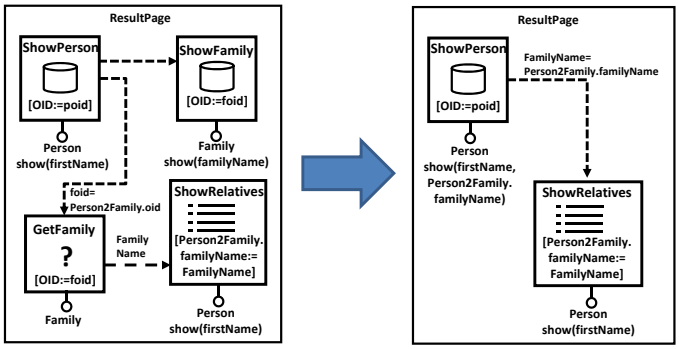

Fig. 10: Possible improvements of the *ResultPage* using *ImportedAttributes* for *DisplayUnits* and *Source Elements* of *LinkParameters*

Operation (EMO) project (*http://www.modelversioning.org/emf-modeling-operations*) which is a dedicated transformation framework for implementing and executing model refactorings. Based on EMO, we have implemented the transformations for the content models as well as the coupled transformations for the instance models and the hypertext models. EMO allows also the interactive execution of the transformations by pre-selecting model elements in the modeling editor. The execution engine of EMO completes the bindings of the model elements in case only a partial pre-binding for the transformation rule has been provided by the user. Finally, EMO also allows for user input during transformation execution, e.g., to give the name for the extracted class.

5 Related Work

With respect to the contribution of this paper, namely evolution and reconciliation support for Web models, we identify two main lines of related work: (i) model refactoring and (ii) multi-viewpoint model synchronization.

Model Refactorings. Compared to refactorings established in the field of object-orientation modeling [16,20], only some initial proposals for Web models exist. Most notable is the work of Cabot and Gómez [2] in which a catalogue of refactorings for improving the navigation between pages has been documented. The presented refactorings are defined on a high-level of abstraction considering links, pages, and navigation paths so they can be translated to any Web modeling methodology as we do. However, their approach only covers one single viewpoint and does not consider the change impact on dependent viewpoints. Mitigating this shortcoming, the work in [8] focuses on the navigation and presentation viewpoints and how they must co-evolve for propagating changes in a consistent way. In particular, they make an OOHDM dependent, fine-grained characterization of different kinds of refactorings. They combine also atomic changes to achieve more complex transformations. In contrast, our approach considers the co-evolution problem between content models and hypertext models.

Model synchronization. A large number of approaches in other disciplines address the problem of multi-viewpoint synchronization [3,5,6,7,9,10,19]. All these approaches

have in common that they consider only atomic changes when reconciling models to satisfy again given modeling language constraints. However, when structuring changes to composite ones, more appropriate reconciled models may be found. For Web applications, instance migration support for evolving databases is presented in [17], but the impact on the hypertext level is not discussed. Cicchetti et al. [4] propose evolution support for Web models going beyond instance migration. The approach uses state-based model comparison to compute the differences between two content model versions based on fine-grained atomic changes, such as adding and removing elements or modifying some of their values. Two coarse-grained change operators are considered in their work: merge/split of classes. The approach is described in detail for the beContent Web modeling language which does not employ an explicit hypertext layer, and briefly discussed for WebML, for which only the reconciliation of hypertext models in case of deletions of content model elements is discussed. Our work is orthogonal in the sense that coarse-grained changes are considered for reconciling WebML hypertext models.

6 Conclusions and Future Work

In this paper we have presented coarse-grained content model changes formalized as model transformations, which are propagated to dependent viewpoints using coupled transformations. The approach has been demonstrated by the *ExtractClass* refactoring example in the particular context of WebML.

Since the reconciliation strategies are defined for the core of WebML, which is also shared by other Web modeling languages, the results should be transferable to other Web modeling languages. In particular, we have abstracted the patterns as much as possible, e.g., by using generalized classes of the metamodel such as *LinkableElement* or *DisplayUnit* which usually have equivalent concepts in other Web modeling languages. By this, the transformations are not specific to the presented example, but are reusable for others. However, there are language concepts which may require their own reconciliation patterns which are not presented in the paper. For instance, if a *DeletionUnit* refers to a class which has been subject to the *ExtractClass* refactoring, an analogous reconciliation pattern is necessary as for the *CreateUnit* to ensure that the instances of the base class and the extracted class are deleted.

As future work we plan to extend the presented catalogue of reconciliation patterns, and identify/resolve possible conflicts between them based on graph transformation theory, in particular, using critical pairs analysis. Furthermore, our patterns aim to preserve the consistency and observable behavior of the system by fulfilling a set of equivalence properties before and after co-evolution models happen. However, coarse-grained changes can be translated to the hypertext viewpoint in different ways, i.e., producing different models where some of them are more efficient than others. In this sense, we want to improve our proposal to determine the most optimal pattern in each case by exploring quality properties such as usability and accessibility of hypertext models. Finally, we want to investigate a hybrid reconciliation approach by using in the first phase the presented approach for coarse-grained changes and in the second phase a constraint-based approach for atomic changes which could not be composed into coarse-grained changes.

Acknowledgements. This work has been partially funded by the Austrian Science Fund (FWF) under grant J 3159-N23, and by Spanish Research Project TIN2011-23795.

References

1. Acerbis, R., Bongio, A., Brambilla, M., Butti, S., Ceri, S., Fraternali, P.: Web Applications Design and Development with WebML and WebRatio 5.0. In: TOOLS'08. LNBIP, vol. 11, pp. 392–411. Springer (2008)
2. Cabot, J., Ceballos, J., Gómez, C.: On the Quality of Navigation Models with Content-Modification Operations. In: ICWE'07. LNCS, vol. 4607, pp. 59–73. Springer (2007)
3. Cicchetti, A., Ruscio, D.D.: Decoupling Web Application Concerns through Weaving Operations. Science of Computer Programming 70(1), 62–86 (2008)
4. Cicchetti, A., Ruscio, D.D., Iovino, L., Pierantonio, A.: Managing the Evolution of Data-Intensive Web Applications by Model-Driven Techniques. SoSym pp. 1–31 (2012)
5. Diskin, Z., Xiong, Y., Czarnecki, K.: Specifying Overlaps of Heterogeneous Models for Global Consistency Checking. In: MDI'10. LNCS, vol. 6627, pp. 165–179 (2010)
6. Eramo, R., Pierantonio, A., Romero, J.R., Vallecillo, A.: Change Management in Multi-Viewpoint Systems using ASP. In: WODPEC'008. IEEE (2008)
7. Finkelstein, A., Gabbay, D.M., Hunter, A., Kramer, J., Nuseibeh, B.: Inconsistency Handling in Multi-perspective Specifications. In: ESEC'93. pp. 84–99. Springer (1993)
8. Garrido, A., Rossi, G., Distante, D.: Model Refactoring in Web Applications. In: 9th International Workshop on Web Site Evolution. pp. 89–96. IEEE (2007)
9. Grundy, J., Hosking, J., Mugridge, W.B.: Inconsistency Management for Multiple-view Software Development Environments. IEEE Trans. Softw. Eng. 24(11), 960–981 (1998)
10. Hofmann, M., Pierce, B.C., Wagner, D.: Symmetric lenses. In: POPL'11. pp. 371–384. ACM (2011)
11. Koch, N., Knapp, A., Zhang, G., Baumeister, H.: UML-Based Web Engineering: An Approach Based on Standards. In: Web Engineering: Modelling and Implementing Web Applications, Human-Computer Interaction Series, vol. 12, chap. 7, pp. 157–191. Springer (2008)
12. Lämmel, R.: Coupled Software Transformations (Extended Abstract). In: First International Workshop on Software Evolution Transformations (2004)
13. Moreno, N., Romero, J.R., Vallecillo, A.: An Overview Of Model-Driven Web Engineering and the MDA. In: Web Engineering: Modelling and Implementing Web Applications, chap. 12, pp. 353–382. Springer (2008)
14. Moreno, N., Vallecillo, A.: Towards Interoperable Web Engineering Methods. JASIST 59(7), 1073–1092 (2008)
15. Schwinger, W., et al.: A Survey on Web Modeling Approaches for Ubiquitous Web Applications. IJWIS 4(3), 234–305 (2008)
16. Sunyé, G., Pollet, D., Traon, Y.L., Jézéquel, J.M.: Refactoring UML Models. In: UML'01. LNCS, vol. 2185, pp. 134–148. Springer (2001)
17. Vermolen, S.D., Wachsmuth, G., Visser, E.: Generating Database Migrations for Evolving Web Applications. In: GPCE'11. pp. 83–92. ACM (2011)
18. Wimmer, M., Schauerhuber, A., Schwinger, W., Kargl, H.: On the Integration of Web Modeling Languages. In: MDWE'07. CEUR Workshop Proceedings, vol. 261 (2007)
19. Xiong, Y., Liu, D., Hu, Z., Zhao, H., Takeichi, M., Mei, H.: Towards Automatic Model Synchronization from Model Transformations. In: Proc. of ASE'07. pp. 164–173. ACM (2007)
20. Zhang, J., Lin, Y., Gray, J.: Generic and Domain-Specific Model Refactoring using a Model Transformation Engine. In: Model-driven Software Development—Research and Practice in Software Engineering. pp. 199–217. Springer (2005)