Weaving Aspect-Orientation into Web Modeling Languages

Irene Garrigós[1], Manuel Wimmer[2], and Jose-Norberto Mazón[1]

[1] WaKe Research, University of Alicante, Spain
{igarrigos,jnmazon}@dlsi.ua.es

[2] Business Informatics Group, Vienna University of Technology, Austria
wimmer@big.tuwien.ac.at

Abstract. While building Web application models from scratch is well-supported, reuse in Web application modeling is still in its infancy. A promising approach in this respect is aspect-oriented modeling to separate certain concerns from the base application, typically cross-cutting ones, and reuse them in various applications. A few Web modeling languages targeting the design phase have been already equipped with aspect-orientation. However, languages for the early phases of Web modeling lack such support, but especially these phases would tremendously benefit from aspect-orientation. Moreover, all the existing solutions are tailored to a specific modeling language. To improve this situation, we consider aspect-orientation itself as an aspect. This allows us to weave aspect-oriented language features into already existing Web modeling languages. We introduce a generic metamodel module comprising the main concepts of aspect-orientation as well as a tool-supported process to weave it into existing base languages. By having this systematic metamodel weaving process, dedicated modeling as well as design-time weaving support is provided for aspects out-of-the-box. We demonstrate our approach by aspectifying a Web requirements modeling language based on i* and applying the aspectified version of it to a case study.

Keywords: Model-driven Web Engineering, Web Requirements, Aspect-oriented Modeling, Language Engineering.

1 Introduction

Web modeling approaches are becoming mature and tailored to supporting different phases of the Web application development process. Several modeling languages exist for specifying requirements, designs, and architectures of Web applications [3, 7, 16]. While building new Web application models from scratch is well-supported by existing approaches, reuse in Web application modeling is still in its infancy. Reuse in software engineering is generally accepted and promoted to improve productiveness [11, 13]. Although a few Web modeling languages provide dedicated reuse mechanisms, e.g., cf. [15], the majority of languages often miss dedicated abstraction, specialization, and integration techniques for reusing Web models.

A promising approach in this respect is aspect-oriented modeling to separate certain aspects from the base application, typically cross-cutting concerns, and reuse them in different places of different base applications. In the context of software engineering, the principle of separation of concerns helps us to manage the complexity of the software systems that we develop by identifying and separating the different concerns involved in a given problem. The concerns that cut across other concerns (i.e. crosscutting concerns) are responsible for producing tangled representations which are difficult to understand and maintain.

Aspect-Oriented Software Development (AOSD) aims to identify and specify such crosscutting concerns in separate modules, known as aspects, in order to improve modularity and hence comprehensibility, maintainability and reusability. Some Web modeling languages targeting design modeling have been already upgraded to support some aspect-oriented language features [3, 16]. However, these solutions are language specific and furthermore, they do not consider the possibility of defining aspects at the requirements analysis phase. Reasoning in the early phases of Web application development about cross-cutting concerns as well as reuse possibilities enables a more efficient design of the Web applications which results in improved development and maintenance.

In order to solve these drawbacks, we present an approach that allows to extend a Web modeling language with aspect-oriented features in a non-intrusive and automated way. This approach also allows the definition of early aspects (i.e., at the requirements level). For this purpose, we have defined a generic metamodel where the main concepts of aspect-orientation are represented, and we have implemented a process in order to weave it into existing modeling languages' metamodels.

We demonstrate our approach by using a Web requirements modeling language based on i* (called Web Requirements Modeling Language (WRML)) and applying the aspectified version of it to a case study.

The remainder of this paper is as follows: Section 2 briefly introduces WRML that is later used to illustrate our approach. Section 3 explains our proposal by describing the aspect metamodel and detailing the composition processes. For a better understanding of the approach, a case study is presented in Section 4. Section 5 studies related work, and finally, Section 6 concludes and sketches future work.

2 Running Example: Modeling Web Requirements

In this section we briefly introduce the Web requirements modeling language (WRML) that we will use as a running example in order to illustrate our approach. It is a goal-oriented proposal to specify requirements in the context of a Web modeling method by using i* models [6].

The i* framework [20] consists of two models: the strategic dependency (SD) model to describe the dependency relationships (represented as —D—) among various actors in an organizational context, and the strategic rationale (SR) model, used to describe actor interests and concerns and how they might be addressed.

The SR model (represented as ◌) provides a detailed way of modeling internal intentional elements and relationships of each actor (◯). Intentional elements are goals (⬭), tasks (⬡), resources (▭) and softgoals (∞). Intentional relationships are means-end links (—▷—) representing alternative ways for fulfilling goals; task-decomposition links

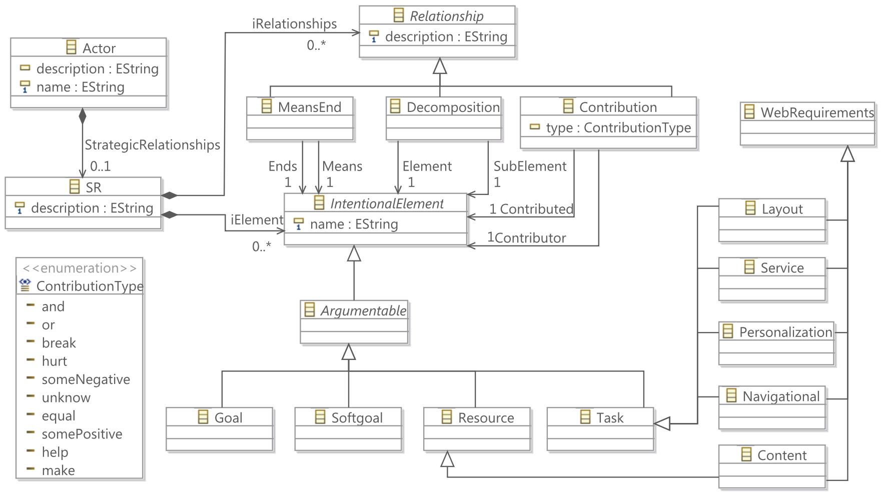

Fig. 1. Overview of our metamodel for *i** modeling in the Web domain

(—+) representing the necessary elements for a task to be performed; or contribution links ($\xrightarrow[\text{hurt}]{\text{help}}$) in order to model how an intentional element contributes to the satisfaction or fulfillment of a softgoal.

Although i* provides good mechanisms to model actors, their intentions and relationships between them, it needs to be adapted to the Web engineering domain to reflect special Web requirements that are not taken into account in traditional requirement analysis approaches, thus being able to assure the traceability to Web design. To adapt the i* framework to the Web engineering domain we used the taxonomy of Web requirements presented in [5]. As such, the i*'s "task" element, which is used to model functional requirements (FRs), was further specialized in the following Web specific requirements: service, navigational, layout, and personalization requirements. With the aim of considering Web requirements, we defined an i* metamodel extended with each of these requirements. Therefore, the core metamodel specifies metaclasses that represent the i* elements and their relationships, as well as new metaclasses according to the different kinds of Web requirements. Some requirements extend the *Task* metaclass: the *Navigational* (represented by an *N* inside the task symbol), *Service* (represented by an *S*), *Personalization* (represented by a *P*), and *Layout* (represented by an *L*). The *Content* requirement extends the *Resource* metaclass and it is represented as a *C* inside the resource symbol.

It is worth noting that non-functional requirements can be modeled by directly using the *Softgoal* metaclass. Figure 1 shows an excerpt of our metamodel for Web requirements specification. For a further explanation, we refer the reader to [1, 6]. A sample application of the i* modeling framework for the Web domain is shown in Section 4.

3 Aspect-Orientation by Aspect-Oriented Language Design

Nowadays we are facing a pantheon of (domain-specific) Web modeling languages. Extending these languages with new modeling features seems the next necessary step to make them more mature and to tackle practical needs. A promising way to tackle this need is to have reusable language modules that can be attached to existing languages. In this paper, we follow this idea for introducing aspect-orientation to existing Web modeling languages.

3.1 Approach Overview

In this section, we describe how we developed a composable metamodel module covering the core concepts of aspect-orientation. This metamodel module is composable with modeling languages defined as Ecore-based metamodels. For composing the aspect metamodel module with base modeling languages, we propose a dedicated composition process that attaches the aspect-oriented concepts to existing modeling languages and an in-place transformation for relaxing some modeling constraints of the base language to allow for language-specific modeling of aspects with respect to a given base language. By this, we can reuse existing modeling editors to specify aspects as well as validation frameworks to check them before they are composed with base models. Based on the aspect-orientation metamodel module and the systematic process for importing this module to base metamodels, we also present a generic composition process for the model level, namely composing aspects and base models. We demonstrate the approach by using WRML introduced in the previous section as a running example, but we have to stress that the approach is reusable for any other modeling language defined as an Ecore-based metamodel.

3.2 Aspect-Orientation Metamodel Module

In Figure 2, we illustrate the core concepts of aspect-orientation based on our previous work on establishing a reference model for aspect-oriented modeling languages [19]. In this paper, we solely focus on the asymmetric approach to aspect-orientation, namely aspects that are not independent but that have to be composed with base models based on explicit and well-defined join points. The symmetric case, i.e., aspects representing independent models that are composed by merging them based on, e.g., name similarity, we leave as subject to future work.

In the metamodel module shown in Figure 2, we have two kinds of concerns represented (this reflects the asymmetric approach): (i) *aspects* are defining *adaptations* described by a set of model elements and (ii) the *base* concern represents the base model that is, of course, made up of a set of model elements (cf. the top of Figure 2).

To connect aspects with base models, *weavings* (cf. the middle of Figure 2) are used that compose a set of *adaptation rules* describing the *effect* (i.e., an enhancement, a replacement, or a deletion in the base model) of an adaptation as well as the *place* where the adaptation actually takes place by using so-called *pointcuts*. A pointcut selects possible *join points* of an aspect by either statically linking directly to some model elements in the base model by using extensions of the *JoinPoint* meta-class or using an

OCL expressions (cf. meta-attribute *Pointcut.expression*) to dynamically calculate the join points in the base model by selecting elements based on certain conditions.

As join points in the context of this work we consider the container elements that will contain the elements of the assigned aspect. The main reason for this decision is the fact that EMF-based models have to comprise a valid tree with respect to the containment hierarchy, i.e., there is only one root element allowed and all other elements have to be inside a unique container.

In addition to the join points, an aspect may interact with the base model by having further cross-references. This means to substitute some aspect elements with specific model element when introducing the aspects to the base models. For modeling such replacements, we allow for template signatures for aspects (cf. Figure 3). Template signatures contain template parameters that may be bound to concrete model elements or String values. This is defined in the context of join points contained by the weaving models. In particular, join points can have template bindings that again contain a set of template parameter substitutions giving for each formal parameter a concrete value. More details on the template support is presented based on concrete modeling examples in Section 4, Figures 7-8.

Important to note is that the aspect-orientation metamodel module (shown in Figure 2) has two root classes with respect to the containment relationships. First, aspect models may be defined that describe reusable aspects in a first step, and second, weaving models may be created for introducing the aspects to several base models. By this separation, aspects are completely independent of particular base model examples and not specific to one weaving model.

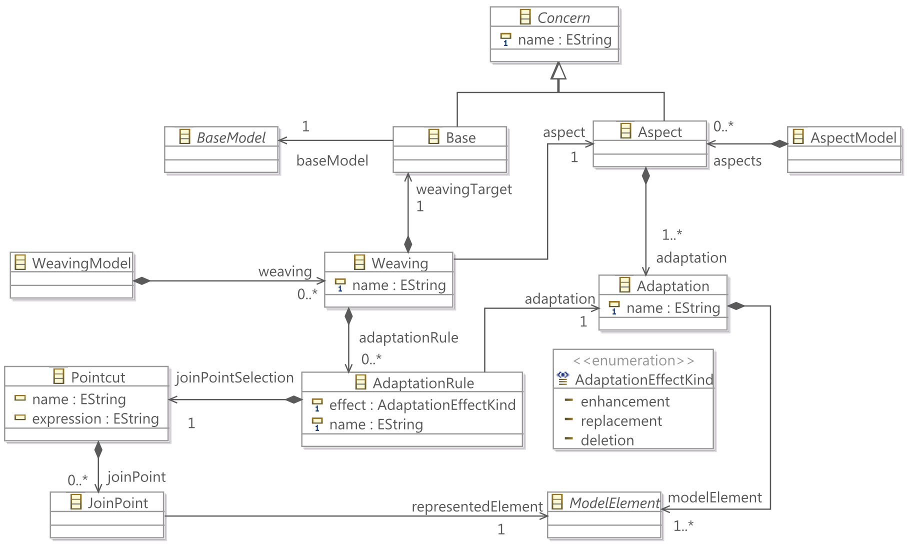

Fig. 2. Aspect-orientation metamodel module in Ecore

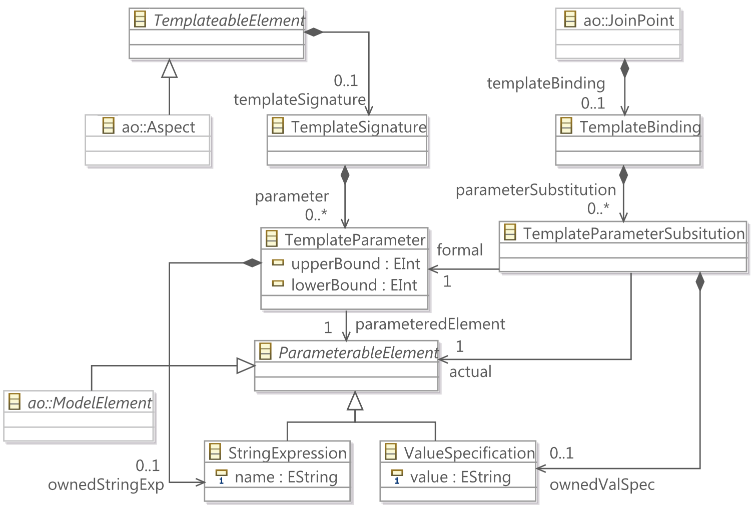

Fig. 3. Templateable aspects (elements from Figure 2 are referred by the prefix *ao*)

Having these concepts defined in the aspect-orientation metamodel module allows to represent aspects and their introduction to base models on a generic level, i.e., agnostic to any metamodel. In the following, we present how this generic aspect metamodel module can be attached to specific metamodels. As one may have already noticed, the abstract meta-classes *BaseModel* and *ModelElement* play an important role in this process.

3.3 Weaving the Aspect-Orientation Metamodel Module into Base Metamodels

To make efficient use of the aspect-oriented concepts described in the previous subsection for a particular language, we foresee a specific process to compute a dedicated metamodel that allows to define well-formed aspect models. As one can see in Figure 2, aspects are composed of a set of different model elements. Thus, also the aspects have to follow the rules of the host modeling language in an eventually relaxed form (because aspects may only represent model fragments, not all language constraints may be meaningful). Thus, we aim for a language-specific aspect language that is equipped with a specific metamodel. This specific metamodel is formed by composing the base language metamodel with the aspect-orientation metamodel module by using some specific join points. By this, we introduce aspect-orientation to languages by using aspect-orientation itself. In particular, the following two steps are necessary to achieve this composition:

- **Step 1:** Aspects have to be introduced to base models by weaving models. For this, the base models have to be identified by the weaving models by setting the *weavingTarget* reference (cf. Figure 2). Thus, it has to be specified on the metamodel level, which model element actually represents the base model in the base metamodel. Normally a root model element containing all other model elements directly or indirectly is used for this purpose in EMF, because the containment relationships span up a tree. Therefore, the language designer has to link the root meta-class of the base language metamodel with the *BaseModel* meta-class of the aspect-orientation metamodel module by using an inheritance relationship.

– **Step 2:** Model elements of the base language are used to specify the adaptation rules, i.e., what has to be introduced, updated, or deleted in the base model. In order to ensure that the elements contained in adaptation rules represent valid model fragments, every meta-class in the base metamodel has to directly or indirectly inherit from the *ModelElement* meta-class of the aspect-orientation metamodel module. By this, the adaptation rules can be populated by ordinary model elements of the base language.

Both steps may be completely automated. While for the first step, a manual selection of root meta-class candidates may be necessary if there is more than one candidate, the second point is completely automatic. All top meta-classes in the inheritance taxonomy of the base metamodel have to be calculated and then these classes get an additional super class, namely, the *ModelElement* class of the aspect-orientation metamodel module. To avoid name clashes, the classes of the aspect-orientation metamodel module are introduced in the base metamodel by using their own package that is integrated as a subpackage in the root package of the base metamodel (normally EMF metamodels have exactly one root package as they exhibit a tree-based package structure). We have implemented support for these two steps by using a parameterizable model transformation, whereas the parameter to be set is the root meta-class of the base metamodel.

Having the classes of the aspect-orientation metamodel module integrated in the base metamodel may be not enough to end up with an appropriate aspect modeling language. Some modeling constraints may be too strict for defining aspects, because aspects represent model fragments and not necessarily complete models that have to fulfill all language constraints. Here similar issues arise as defining a domain-specific transformation language that is tailored for a given modeling language [10]. Thus, we apply a relaxation process to the base metamodel before the composition with aspect-orientation metamodel module takes place to end up with a language that allows to model aspects that may be completed by weaving them into the base models in a flexible way. In particular, this relaxation process requires to reset the lower multiplicities of the references and attributes in the base metamodel to zero in order to make them all optional properties. By this, aspects can be defined as model fragments that are completed by the subsequent weaving step. Again, the relaxation process is automated by a model transformation.

Example. In Figure 4, a small excerpt of the aspectification of the WRML is shown. On the top of this figure, an excerpt of the WRML metamodel and an excerpt of the aspect-oriented metamodel module are illustrated by the two packages as well as the *Actor* meta-class representing the root element of the base metamodel. The two metamodels are merged into one metamodel whereas the *AO* package is now nested into the base language root package. Furthermore, the inheritance relationships are specified as described above and the relaxation takes place. When running this composition process

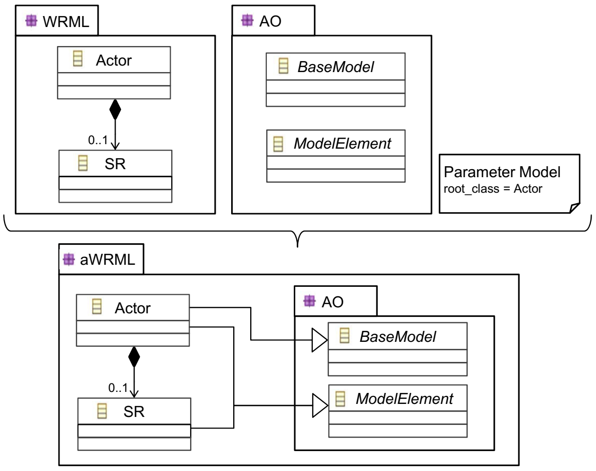

Fig. 4. Aspectification of WRML (excerpt)

on a tuple consisting of a base metamodel and the aspect metamodel, we finally end up with a metamodel that allows us to model aspects that can be checked against the newly created *aspectified* metamodel as well as it allows to define weaving links between a base model (that is an instance of the base metamodel) and aspects. In the next subsection, we show how aspect models and weaving models are specified.

3.4 Modeling Aspects and Weavings with Aspectified Metamodels

Based on the available metamodeling capabilities and out-of-the-box tool support in EMF, we can instantiate the automatically produced aspectified metamodels by standard EMF modeling editors. The particular usage of the metamodel is to model aspect models that define independently from base models certain model fragments that are later integrated with specific base models by using a weaving model. We decided to go for a separation of aspect models and weaving models to allow for aspect repositories from which a modeler may select certain aspects and integrate them to her base model using additional weaving models.

Example. To demonstrate how to use the aspectified metamodels, we present a small instance of the afore enhanced metamodel in Figure 5. On the right-hand side of the figure, a small aspect model is shown that is conform to the aspectified metamodel. The aspect only describes a small adaptation, namely to add a task to an actor by adding it to the SR element. In order to actually use this aspect, a weaving model is residing between the base model and the aspect model. In the upper half of Figure 5, the weaving model uses an explicit joinpoint defined by the modeler that points to the particular SR element that should be enhanced by the aspect, i.e., who should get the task assigned, while in the lower half of Figure 5, no extensional joinpoint is used. Instead, it is described intensionally by a OCL query that is defined by the *Pointcut* object. The OCL query

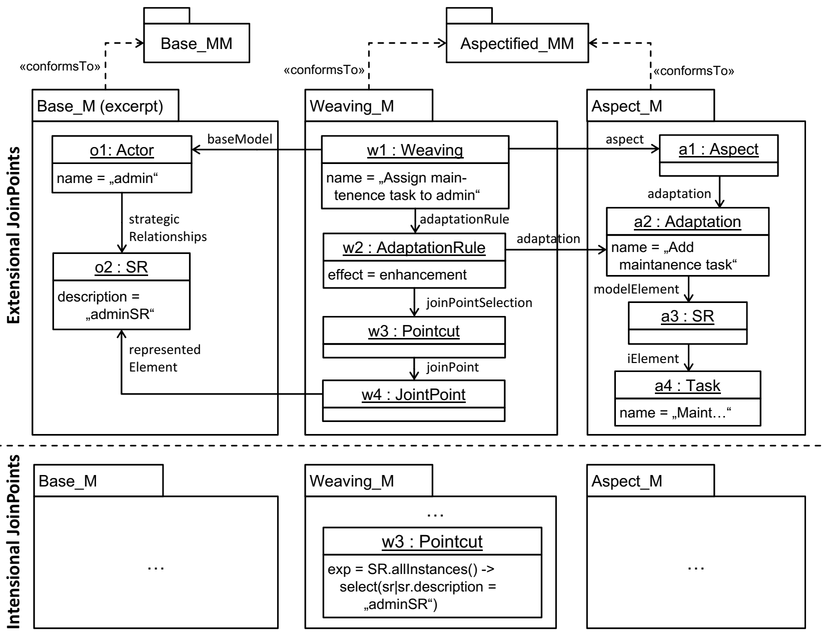

Fig. 5. Aspects and Weaving Models by Example

is evaluated by an OCL engine and all the selected elements are extended by the given aspect. One may think about the OCL query as a mechanism to compute automatically a set of joinpoints. In our example, the evaluation of the OCL query results in the same weaving semantics as has before been extensionally defined. Only the SR element with the description ”adminSR” gets the aspect attached when we assume that only one SR element with such a value exists in the model.

3.5 Model Composition Process

Having the weaving models defined between the aspect models and the base models, a composition process is activated to actually introduce the aspects to the base models by producing an enhanced model that is solely conform to the base metamodel. In order to compose models with aspects, several composition strategies are possible:

- *Arbitrary order*: Weavings are contained in one single set and are executed based on their implicit order, i.e., the order in which they are contained in the weaving model.
- *Priorities*: If aspects are building on each other, priorities have to be supported for weavings, i.e., to split the single set of weavings into a sequence of different subsets.
- *Orchestration*: For more complex weaving plans, control structures such as conditionals may be applied to orchestrate the execution of the composition.

Currently, we support no particular order for integrating the aspects to the base models. The order is implicitly given by the order of the weaving elements in the weaving model which is traversed and based on the traversal strategy, the aspect order is given. The concrete composition of one aspect to the base model is as follows: First, the aspect is retrieved from the weaving object and the first join point that adds the aspect model elements to the containment structure of the base model is executed. Of course, constraints have to be checked such as if the base model element that gets the aspect root model element has an empty slot in the used reference available. Consider for instance that the base element has already another element contained by a reference that has as upper bound cardinality 1. Then, the aspect root element cannot be added to the base model without either deleting the already referenced element or destroying the well-formedness of the base model. In such cases a warning is given and the composition process is terminated. Furthermore, not all elements have to be added but the templates of the aspect have to be substituted.

3.6 Implementation

We have implemented our approach for the Eclipse Modeling Framework (EMF). The metamodel composition process has been implemented as a Java-based transformation that takes the base metamodel and the aspect metamodel as input and produces as output the aspectified metamodel. Based on this aspectified metamodel, aspects containing adaptations as well as weavings containing adaptation rules are defined. In particular, we exploit the capabilities of EMF to load several resources within one modeling editor that allows basic inter-model reference support out-of-the-box. The model composition process takes as input the model triple (i.e., base model, aspect model, weaving model) and produces the composed model. Again this process is implemented as a model transformation.

4 Case Study

In this section, we consider the example shown in Figure 6. The figure shows the SR model of a book store. The main goal of the system actor is to manage the book sales. In order to fulfill this goal we have two different alternatives: books can be sold online or books can be sold at the store. We are going to focus on the first task *"books be sold online"*. In order to complete this task, two subtasks should be completed: *"provide book info"* (which is a navigational requirement) and *"provide recommended books"* (which is a personalization requirement).

The data is specified by means of the content requirement *"books"*. Each of the leave tasks is related with this content requirement.

In the following, we show how to model two new concerns in an application independent way by defining aspects using the aspectified metamodel of WRML. Furthermore we show how the aspects can be introduced in the base model by using weaving models. To allow for a more concise representation, we switch from the abstract syntax to a concrete syntax to visualize the models in the rest of this section. For didactic reasons, we present the base model and weave both concerns independent from each other to it. The woven model is illustrated in Figure 9.

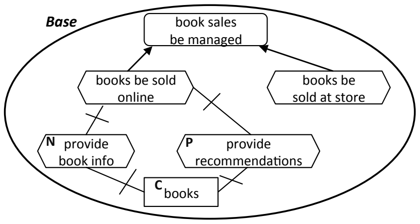

Fig. 6. Base model

4.1 The Shopping Cart Aspect

Concern Description. The first concern we want to model is a shopping cart. The tasks needed are adding a product to the cart and viewing the shopping cart content.

Aspect Model. The shopping cart aspect is shown in the right hand side of Figure 7. This concern includes a general task *"provide a shopping cart"*. It is decomposed into two subtasks to be completed, which are of different type: *"$add <x> to cart$"* (service requirement) and *"view cart content"* (navigational requirement) which are related to the content requirement *"cart"*.

The aspect has a template signature consisting of two template parameters, one is a String expression while the other is a content requirement element. Please note that the name of the first subtask is dynamically computed by including the first parameter, since we can weave this aspect with a base model of any kind of e-commerce application (i.e., not necessarily a book store). The tasks are also related with a content requirement named*"item"* which is a generic element, that will depend again on the type of application of the base model we weave the concern in.

Weaving Model. As we can see in Figure 7, the shopping cart concern is a subtree of tasks with cross-links to content requirements that is introduced in the base model by using a weaving model. The join point (jp) for the aspect is the general task *"books be sold online"* in the base model that acts as container element for the general task *"provide a shopping cart"*. Furthermore, we need for this example additional cross-links from the aspect model elements to the base model elements by binding the template

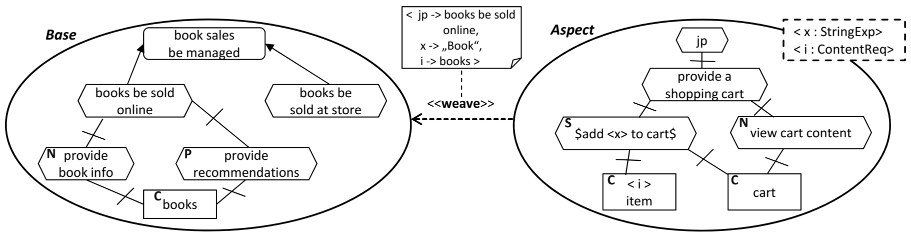

Fig. 7. Shopping cart aspect

parameters. In particular, the content requirement *"books"* has to play the role of the template *"item"* of the shopping cart aspect. Thus, when introducing the aspect to the base model, the template *"item"* has to be substituted with *"books"*. This is done by mapping the content requirement *"books"* of the base model to the template parameter *i* of the aspect. Finally, for the first parameter of the aspect, namely parameter *x*, a primitive String value has to be provided. This parameter is needed for calculating the name of an element contained in the aspect when it is introduced to the base model. Thus, the parameter is also occurring in the expression that is used to calculate the name of a navigation concern (cf. expression *"\$add <x> to cart\$"*). The rest of the elements contained in the aspect are just "simple" model elements that are woven into the base model by just copying them.

4.2 The Search Engine Aspect

Concern Description. This concern refers to a search engine that we want to reuse. In this case we have defined the search by keyword or by category. We can also refine the search by expanding our keyword list with semantic-related keywords or by listing all the possible items.

Aspect Model. The aspect model of this concern can be seen on the right hand side of Figure 8. It has a goal named *"\$<i.name> be searched\$"* . Again this concern is modeled in a generic way (i.e., independent of the target base model). This goal is decomposed into two tasks, we can *"search by keywords"* or *"search by category"*. In order to fulfill the first one, a goal *"search to be refined"* and a task *"\$search by <x> keyword\$"* (service requirement) need to be completed. The search can be refined in two ways: *"by listing all'* or *"by keyword expansion"*. Again in this case, the leave tasks are related with the needed content requirements.

For this aspect, the template signature consists of three parameters. First, a String expression is needed for computing the name of the keyword search task. Second, the content requirement playing the role of the item has to be introduced. Finally, an additional content requirement presenting the categories of the items is considered. Please note that the latter is optional, i.e., the multiplicity of this template parameter is zero-to-one. This has as consequence, that if it is bound to a specific element then the parameterized element

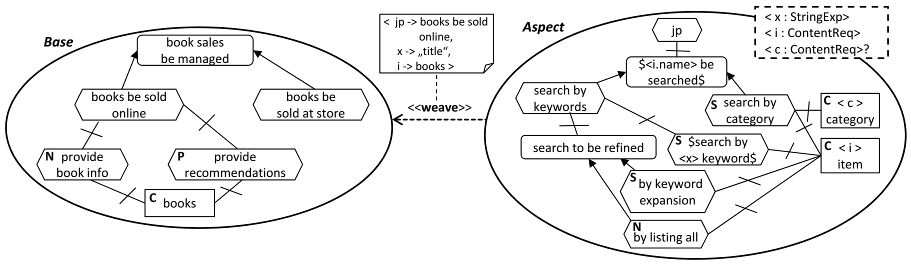

Fig. 8. Search engine aspect

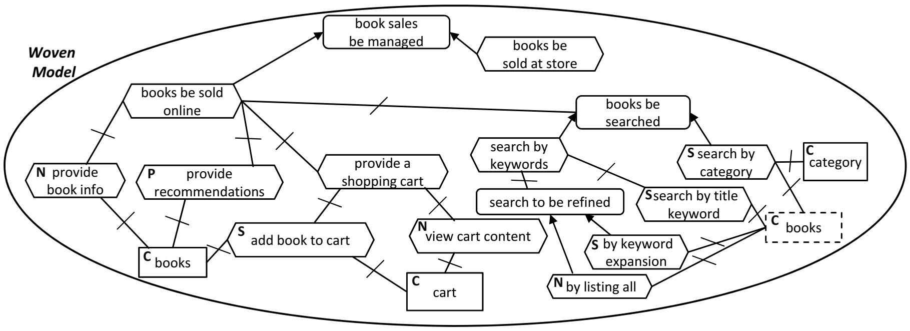

Fig. 9. Resulting model after aspect composition

is substituted by the specific element. Otherwise the content requirement "category" is introduced in the base model.

Weaving Model. As for the shopping cart concern, the join point for the aspect is the general task *"books be sold online"* in the base model. In contrast to the previous example, only one of the two content requirement parameters is mandatory to be bound in the weaving between the base and the aspect model. More specifically, the content requirement *"item"* has to be bound (that is bound to content requirement *"books"* again by using the parameter *i*), but not the content requirement *"category"* (thus it is introduced to the base model in our example, because no binding has been specified for the parameter *c*). Besides the content requirement parameters, we have again a simply typed parameter that is used in the expression ("\$search by <x> keyword\$") to calculate the name for a task similar as it has been done for the shopping cart aspect. In addition, the content requirement parameter *i* is used also in an expression for calculating the name of another task (cf. "\$<i.name> be searched\$"). Here the parameter is in the expression for accessing the name of the content requirement that is bound to the content requirement parameter by using an OCL expression.

5 Related Work

With respect to the contribution of this paper, we elaborate on aspect-oriented modeling support in the field of Model-Driven Web Engineering (MDWE).

Baumeister et al. [2] presented one of the early works in MDWE that proposed the use of aspect-orientation for modeling customization in hypertext models. In the UWE metamodel, the Aspect concept has been introduced as a sub-class of the UML meta-class Package. Furthermore an Aspect is defined to have one Pointcut and one Advice each likewise specialized from the UML meta-class Package. The approach distinguishes between static aspects woven at design time and dynamical aspects woven at runtime.

In [3], Casteleyn et al. presented the extension of the Hera-S approach with aspect-oriented concepts for modeling customization concerns in Web applications. More specifically, to separate adaptations from the hypertext models, the textual language Semantics-based Aspect-Oriented Adaptation Language (SEAL) has been designed.

OOHDM provides support to compose so-called volatile concerns with a base Web application [7]. A volatile concern introduces some additional services temporarily in Web applications for a short and determined period of time. For supporting volatile concerns, OOHDM allows to model each concern separately by defining a separate content, hypertext, and presentation models. In addition, OOHDM provides a textual language for defining the integration of the separate models with the base Web application.

For WebML, we have proposed an aspect-oriented extension called AspectWebML [16]. For realizing AspectWebML, the WebML metamodel has been extracted semi-automatically from WebRatio and then extended by similar aspect-oriented concepts as we have used in this paper for the generic aspect-orientation module. However, in [16] the aspect-oriented extension of the language has been manually achieved.

Cicchetti et al. [4] present a weaving model based approach to model Web applications using different viewpoints. While we are separating model fragments from one modeling view, they are concerned with consistency between different viewpoints. This is especially needed for Web models, because several viewpoints are used such as content, navigation, and presentation. To validate consistency between viewpoints, constraints are introduced for weaving links that allow to reason if weaving models are complete and consistent. Thus, they do not use weaving as in aspect-oriented modeling, but as a general form of correspondence links between viewpoints.

To the best of our knowledge, none of the existing Web modeling languages support aspects at early stages of the development. The only exception is the recent work presented by Urbieta et al. [18], which defines an approach for tackling crosscutting workflow behaviour in the requirements phase in the context of the WebSpec requirements modeling language [1,12]. Outside the MDWE field, there exist some approaches that consider specifying crosscutting concerns in the requirements level of a software application [8, 14, 17]. They mainly propose extensions for UML, i.e., in particular for use cases, to define aspects.

Furthermore, we are not aware of an approach that considers aspect-orientation as a generic concern that can be introduced into existing modeling languages. All the mentioned approaches have been particularly designed for a specific language. In this paper we take a different angle by semi-automatically enhancing existing languages with aspect-oriented modeling features. This has been exemplified in the paper for WRML. However, the application of our approach for re-building the aforementioned languages already providing aspect-orientation support is left as subject to future work.

6 Conclusions and Future Work

In this paper we have presented an approach to equip already existing languages with aspect-oriented modeling concepts by using aspect-orientation itself on the metamodel level. By having aspectified metamodels, aspects and weavings can be systematically modelled and validated by reusing out-of-the-box EMF support as well as a generic design-time model weaver is provided for actually introducing the aspects to the base models. We have demonstrated the approach by using WRML as an example language and showed how Web requirements can be expressed as aspects.

For future work we aim to rebuild already aspect-oriented modeling languages to see wether this is possible in a generic manner or specific adaptations are necessary. Another important point for future work is if different execution orders of given weavings may result in different woven models, i.e., if the aspects are confluent or not. For this we plan to investigate the notion of critical pairs defined for graph transformations [9]. In particular, we plan to provide a mapping to graph transformations in order to reuse the reasoning techniques currently available for determining if a set of graph transformation rules, representing the aspects and weavings, is confluent or not.

References

1. Aguilar, J.A., Garrigós, I., Mazón, J.-N., Trujillo, J.: An MDA Approach for Goal-oriented Requirement Analysis in Web Engineering. J. UCS 16(17), 2475–2494 (2010)
2. Baumeister, H., Knapp, A., Koch, N., Zhang, G.: Modelling Adaptivity with Aspects. In: Lowe, D.G., Gaedke, M. (eds.) ICWE 2005. LNCS, vol. 3579, pp. 406–416. Springer, Heidelberg (2005)
3. Casteleyn, S., Woensel, W.V., Houben, G.-J.: A semantics-based aspect-oriented approach to adaptation in web engineering. In: Hypertext, pp. 189–198 (2007)
4. Cicchetti, A., Di Ruscio, D.: Decoupling web application concerns through weaving operations. Sci. Comput. Program. 70(1), 62–86 (2008)
5. Cuaresma, M.J.E., Koch, N.: Requirements engineering for web applications - a comparative study. J. Web Eng. 2(3), 193–212 (2004)
6. Garrigós, I., Mazón, J.-N., Trujillo, J.: A Requirement Analysis Approach for Using i* in Web Engineering. In: Gaedke, M., Grossniklaus, M., Díaz, O. (eds.) ICWE 2009. LNCS, vol. 5648, pp. 151–165. Springer, Heidelberg (2009)
7. Ginzburg, J., Rossi, G., Urbieta, M., Distante, D.: Transparent Interface Composition in Web Applications. In: Baresi, L., Fraternali, P., Houben, G.-J. (eds.) ICWE 2007. LNCS, vol. 4607, pp. 152–166. Springer, Heidelberg (2007)
8. Grundy, J.C.: Aspect-Oriented Requirements Engineering for Component-Based Software Systems. In: RE, pp. 84–91 (1999)
9. Heckel, R., Küster, J.M., Taentzer, G.: Confluence of Typed Attributed Graph Transformation Systems. In: Corradini, A., Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2002. LNCS, vol. 2505, pp. 161–176. Springer, Heidelberg (2002)
10. Kühne, T., Mezei, G., Syriani, E., Vangheluwe, H., Wimmer, M.: Explicit transformation modeling. In: Ghosh, S. (ed.) MODELS 2009 Workshops. LNCS, vol. 6002, pp. 240–255. Springer, Heidelberg (2010)
11. Li, J., Gupta, A., Arvid, J., Borretzen, B., Conradi, R.: The empirical studies on quality benefits of reusing software components. In: COMPSAC, pp. 399–402 (2007)
12. Luna, E.R., Rossi, G., Garrigós, I.: WebSpec: a visual language for specifying interaction and navigation requirements in web applications. Requir. Eng. 16(4), 297–321 (2011)
13. McClure, C.: Software Reuse: A Standards-Based Guide. Wiley (2001)
14. Rashid, A., Sawyer, P., Moreira, A.M.D., Araújo, J.: Early Aspects: A Model for Aspect-Oriented Requirements Engineering. In: RE, pp. 199–202 (2002)
15. Rossi, G., Schwabe, D., Lyardet, F.: Abstraction and Reuse Mechanisms in Web Application Models. In: Mayr, H.C., Liddle, S.W., Thalheim, B. (eds.) ER Workshops 2000. LNCS, vol. 1921, pp. 76–88. Springer, Heidelberg (2000)
16. Schauerhuber, A., Wimmer, M., Schwinger, W., Kapsammer, E., Retschitzegger, W.: Aspect-Oriented Modeling of Ubiquitous Web Applications: The aspectWebML Approach. In: ECBS, pp. 569–576 (2007)

17. Suzuki, J., Yamamoto, Y.: Extending UML with Aspects: Aspect Support in the Design Phase. In: ECOOP Workshops, pp. 299–300 (1999)
18. Urbieta, M.M., Rossi, G., Gordillo, S., Schwinger, W., Retschitzegger, W., Escalona, M.J.: Identifying and Modelling Complex Workflow Requirements in Web Applications. In: Grossniklaus, M., Wimmer, M. (eds.) ICWE Workshops 2012. LNCS, vol. 7703, pp. 146–157. Springer, Heidelberg (2012)
19. Wimmer, M., Schauerhuber, A., Kappel, G., Retschitzegger, W., Schwinger, W., Kapsammer, E.: A survey on UML-based aspect-oriented design modeling. ACM Comput. Surv. 43(4), 28 (2011)
20. Yu, E.S.K.: Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. In: RE, pp. 226–235 (1997)