Addressing New Concerns in Model-Driven Web Engineering Approaches

Nathalie Moreno[1], Santiago Meliá[2], Nora Koch[3,4], and Antonio Vallecillo[1]

[1] Universidad de Málaga, Spain
[2] Universidad de Alicante, Spain
[3] Luwig-Maximilians-Universität München, Germany
[4] Cirquent GmbH, Germany
vergara@lcc.uma.es, santi@dlsi.ua.es,
kochn@pst.ifi.lmu.de, av@lcc.uma.es

Abstract. In the last few years, almost all model-driven Web Engineering approaches have evolved in response to the new challenges of Web systems design, which are due to new requirements and implementation technologies in the Web domain. The evolution implies the extension and adaptation of current approaches, in terms of new models, transformations and processes in order to incorporate new concerns or aspects. Such changes in a methodology are a risky and error-prone process. In this paper, we analyze different alternatives to address the evolution and in particular, the addition of a new concern in a Model-Driven Web Engineering approach: (a) extending the original method with an additional modeling concern, (b) merging the original proposal with another approach covering the specific concern and, (c) finally, we propose a an interoperable and architectural-centric approach that aims to reduce the impact of adding a new concern. We discuss the main advantages and drawbacks of each alternative.

Keywords: Separation of Concerns, Web Engineering, Model-Driven Software Development, Metamodeling, Model Transformations.

1 Introduction

Model-driven development (MDD) is an approach to software development that uses models, metamodels and model transformation as key elements of the development process [2]. It incorporates a higher level of abstraction in the specification of systems guided by the *separation of concerns principle* and allows the (semi)-automated derivation of the final implementation code. In this sense, most of the existing Web engineering approaches match the MDD philosophy, because they address the development of Web applications using separate models to describe the different concerns that constitute Web systems. Furthermore, they provide model compilers that permit automatic generation of system implementations from high-level models.

Traditionally, the majority of Web engineering proposals have considered the content, navigation and presentation models, as the most relevant concerns in the design of a Web application. However, due to new requirements and implementation

J. Bailey et al. (Eds.): WISE 2008, LNCS 5175, pp. 426–442, 2008.

technologies in the Web domain, Web engineering approaches have to evolve. Following a model-driven approach, the evolution may imply: (a) the definition of new models and modeling elements that capture additional requirements; (b) the redefinition of the metamodel for handling these additional features; (c) the adaptation of the development process to incorporate the new concern and the information it represents; (d) the adaptation of the modeling process and code generation tools that support the method.

To avoid the need for these changes, some approaches have studied the practical viability of merging their design methods with others [10,11]. Indeed, one of the main advantages of using MDD in the Web application domain is the possibility of establishing and exploiting the synergies existing among the approaches using simple model transformations. In this paper we will discuss three different alternatives for addressing the evolution of Web engineering methods: (1) extending the original method with an additional modeling concern, (2) merging the original proposal with another approach covering the specific concern and, (3) finally, we propose an interoperable and architecture-centric process that aims to reduce the impact of adding a new concern. For the last approach we propose to use the (Web Engineering Interoperability) WEI common metamodel [11,18] which encapsulates the concepts of almost all Web engineering approaches and the architecture models of the (Web Software Architecture) WebSA approach [9]. The three alternatives will be illustrated by introducing a new concern to model the business process operations performed within Web applications.

The remainder of this document is structured as follows. Section 2 classifies the concerns involved in the design of a Web application based on its dependency relationships with other concerns and the impact that addressing these relationships would have on a method. Section 3 analyzes three ways to address the evolution of a Web engineering method. In Section 4 the main advantages and drawbacks of each alternative are discussed. Then, Section 5 relates our work with other similar studies. Finally, Section 6 draws some conclusions and outlines some future research activities.

2 Classification of Concerns

Adding a new concern to a Web engineering method, with a well known and structured separation of concerns, models and code generation process, is not a trivial task. The complexity of this process will depend largely on the nature of the concern considered and the relation it has with the current ones. Based on these considerations, we distinguish the following categories of concerns:

Dependent concern. This concern has one or more dependency relationships with others. Dependency relationships establish an order relation when defining the system models, since they force the designer to define firstly the independent concerns and subsequently the other concerns that depend on these. In addition, this kind of concern involves managing consistency at model level between related concerns. Modifying some modeling elements of a concern may have a knock on effect causing changes in other concerns that depend on it. For example, both in the UML-based Web Engineering (UWE) [6] and the OO-H [7] methods the navigation model is derived in

part from the content or conceptual model respectively (i.e. there is a dependency relationship between elements of both models). Therefore, when a class of the content/conceptual model is deleted, all navigational classes or relations that were defined as a view on elements of that model must also be deleted. Nowadays, the addition of a dependent concern to a methodology is done by hand. This process involves the definition of an extension of the existing metamodel that addresses it and also the specification of possible relations between the new metaclasses and the existing ones.

Replacement concern. This is a concern that replaces another previously defined for the same method but it offers a new viewpoint to address the modeling requirements. When the new concern represents a total change with respect to the previous one, its corresponding metamodel is replaced by the new one. In other cases, the original metamodel is only subject to certain modifications. In fact, a replacement concern may also be a dependent concern that has to maintain consistency with other concerns at the metamodel level, i.e., respecting the relations that the concern being replaced had with the others. Presentation is an illustrative example of replacement concern. The advent of the Web 2.0 has shown that existing Web methods require more expressive models for addressing the user interface of a system due to traditional mechanisms are now inadequate [8].

Orthogonal concern. It represents a new concern that models a feature of a system which is completely independent of all the others. In this case an independent metamodel package and corresponding transformations are defined. They capture the aspects of the new concern without affecting the other packages of the metamodel. In this way, not only the design time but also the development time can be optimized since developers can work simultaneously, each one on a different concern. Software architecture is, for example, an orthogonal concern to navigation and presentation.

3 Addressing a New Concern

Business processes have gained a lot of importance in Web applications. Addition of business processes to modern Web applications entails new challenges to the development of Web applications. Following the previous classification, the business process concern can be considered a dependent concern because it has a strict dependency relationship with the content model. Hence, current Web engineering methods have to propose extensions that include appropriate modeling concepts specifically tailored to cope with this kind of requirements and appropriate horizontal and vertical transformations. Horizontal transformations are those between models at the same level of abstraction; vertical transformations “refine” system models, adding information and therefore making them closer to the technology or implementation platform. In this section we present three different ways to address a Web method extension, illustrated by a running example.

The example models a very simple music portal Web application that allows users to search music albums and songs by name. The search result is presented as a list of matching albums/songs that provides links to a detail page for each album. The album

detail pages show the title of the album, the name of the artist, the list of songs and the album's price. For the sake of reducing complexity, in this example each album has only one artist. The new concern, introducing *modeling of business process operations*, is illustrated by the functional requirement to allow registered users to buy albums, which then can be downloaded as archive files containing MP3s. The following list gives a short informal description of the requirements.

- A user becomes a registered user by logging in. Unregistered users can register with a username and a freely chosen password.
- If the user has already bought the album then a download link is shown. Otherwise, there will be a link for buying the album. Only full albums can be downloaded.
- Each registered user has a credit account that is used to buy albums. The credit account can be recharged by credit card payment.
- The links for logging in or out, for registering and to the user's account page are always shown. This also holds for the album search box.

3.1 Extending UWE with an Additional Modeling Concern

Similar to other Web engineering methods, UWE [6,16] addresses the different concerns of a Web application by the construction of different models for the content, the navigation structure, and the presentation. The distinguishing feature of UWE is the use of standards, in particular the Unified Modeling Language (UML [15]). It is UML2 compliant since modeling with UWE is based on a UML2 profile, which is defined on a lightweight extension of the UML metamodel (see [6] for more details about the UWE method and notation). UWE evolves, in the same way other Web engineering approaches do, integrating modeling facilities for additional concerns. In fact, UWE recently integrated techniques for modeling business processes, which are driven by user actions. In the following we show UWE models by example and explain how the new concern is added to the UWE approach [7].

A content model built with UML classes models the content of Web applications in UWE. In our running example the content is modeled by classes such as Album, Song and Artist (Fig. 1). Content classes are shown in a UML class diagram together with their relationships (associations, composition, inheritance, etc.).

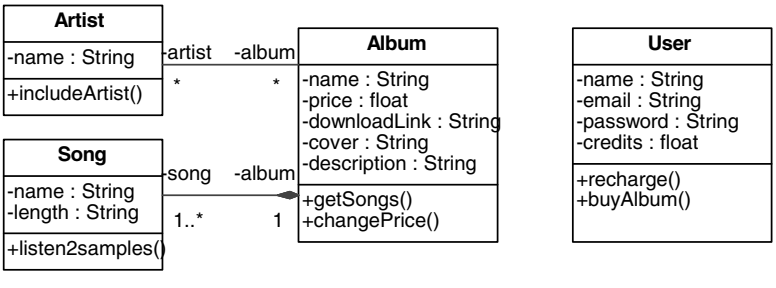

Fig. 1. UWE content mod el for the music portal example

The navigation model is based on the content model and represents the navigation paths of the Web application. A navigation class represents a navigable node in the Web application and is associated to a content class containing the information of the node, e.g. navigation classes Album and Song. Navigation paths representing direct links between two navigation nodes are represented by associations called navigation links (for simplicity the corresponding stereotype is omitted in Fig. 2). Additional navigation nodes are access primitives used to reach multiple navigation nodes (index and guided tour) or a selection of items (query). Menus model alternative navigation paths. Examples of access primitives in the navigation model of the music portal are AlbumQuery and AlbumIndex; example of menu is the MainMenu.

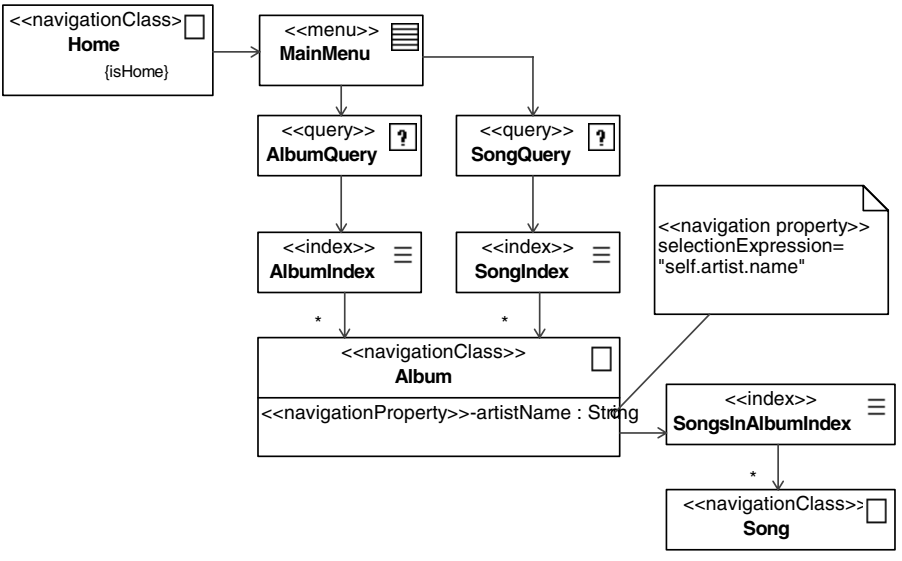

Fig. 2. UWE navigation model for the music portal example

The presentation model is used to sketch the layout of Web pages associated to the navigation nodes. For an example of a presentation model the reader is referred to [6].

Web applications are no longer built for browsing information, they are instead required to offer to the user more and more functionalities, such as search facilities and business process operations. This kind of operations constitutes therefore a new concern that needs to be modeled for certain Web applications. UWE is extended to cover this new concern as follows:

- A process model is added. It includes the process classes which contains the required data for the process, such as Login, BuyAlbum, BuyAlbumConfirmation, and InsufficentCreditMessage (see Fig. 3)[12].
- A navigation model is enriched with process classes indicating entry and exit points of the processes, e.g. process classes Register, Login, BuyAlbum, etc. Process classes and process links are shown in Fig. 4.

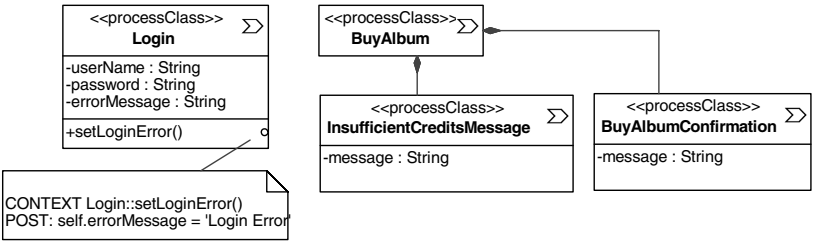

Fig. 3. Excerpt of the UWE process model of the music portal example

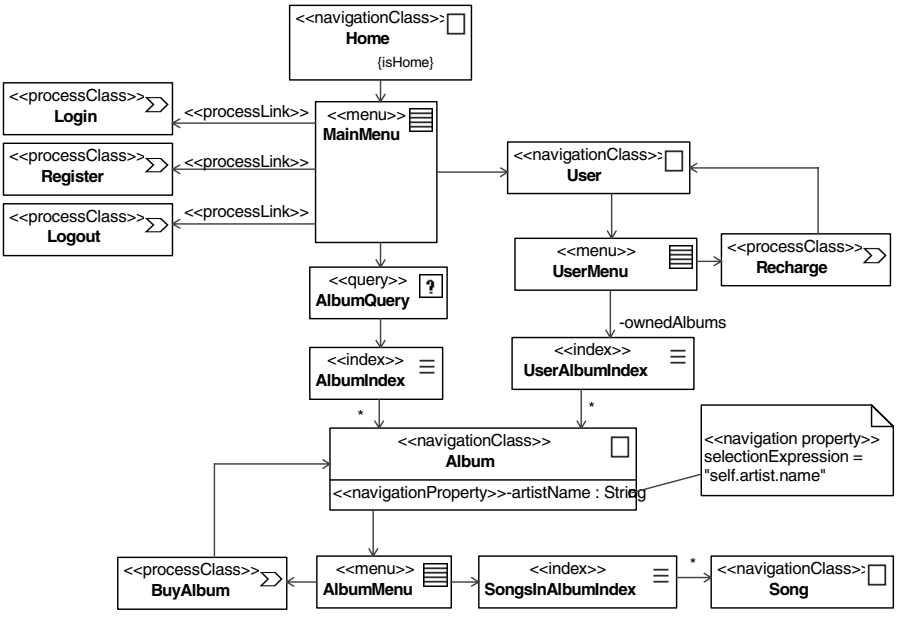

Fig. 4. Extended UWE navigation model of music portal example

- A workflow is specified which models the activities of the business process logic, e.g. describing the Login and Logout processes, the registration or the download of an album (not shown here due to space limitations).

Adding a concern in UWE means that new elements and relationships have to be defined and included in the UWE metamodel. Fig. 5 illustrates how the Navigation package of the UWE metamodel is extended with ProcessClass and ProcessLink.

The UWE approach defines in addition to the UML profile and the UWE metamodel, a model-driven process for the development of Web systems. The process relies on models and model transformations following the MDA principles and using several other standards [13]. A set of design model types is used in UWE to model the different concerns of the Web applications. The transformations for refining the design models comprise mappings from the content to the navigation model, refinements of

the navigation model, and from the navigation into the presentation model. In UWE, an initial navigation model is generated based on classes of a stereotyped content model. This generation step can be rendered as a transformation Content2Navigation. From a single content model different navigation views can be obtained, e.g., for different stakeholders of the Web system.

Starting with this basic navigation model, it can be further refined by a set of vertical transformation rules that can be applied fully automatically. These rules include for example the insertion of indexes and menus. Similarly, presentation elements are generated from navigation elements. For example, for each link in the navigation model an appropriate anchor is required in the presentation model. So far, look and feel aspects have to be added manually. Transformations are defined as OCL constraints (by preconditions and postconditions) in UWE and are implemented either in Java in plug-ins for CASE tools such as MagicDraw and ArgoUML, or in the model transformation language ATL [1].

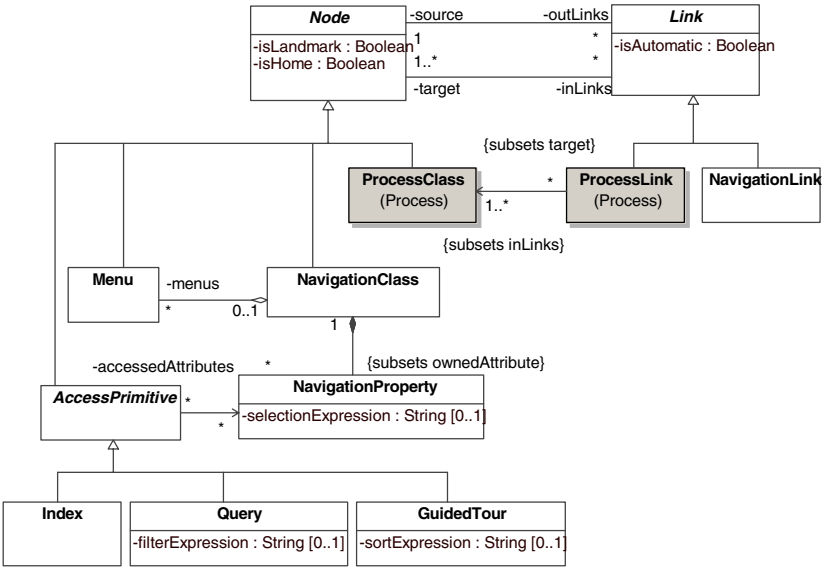

Fig. 5. Excerpt of the UWE metamodel (metaclasses for new concern are colored)

The UWE MDD process comprises also an integration step. The aim is the creation of a single model for validating the correctness of the functional models. This "big picture" model is a UML state machine, representing the content, navigation structure, and the business processes of the Web application as a whole. A model checker like Hugo/RT [6] can check this state machine. In order to transform platform independent to platform specific models additional information of the platform is required. It can be provided as an additional model or it is implicitly contained in the transformations. For mappings from design models to code UWE also uses vertical transformation rules written in ATL that generate Java Beans and JSPs.

The activities required for the extension of the UWE approach with the new concern *business processes* are detailed in the following indicating which is the expertise required for such an extension. This extension process is shown in Fig. 6. The UWE expert has to extend the profile and the metamodel. Model-to-model and model-to-code transformations need to be defined by the transformation expert. Finally, the tool builder has to introduce the corresponding changes in the UWE tool to support modeling and generation of Web applications including business processes. The steps outlined previously would be more complex and costly to implement if the separation of concerns is not previously established as it is in UWE (i.e., where concerns are grouped in a single model). In these cases it may even be necessary to reorganize the models and transformations defined initially for the approach in order to be able to carry out the required extension.

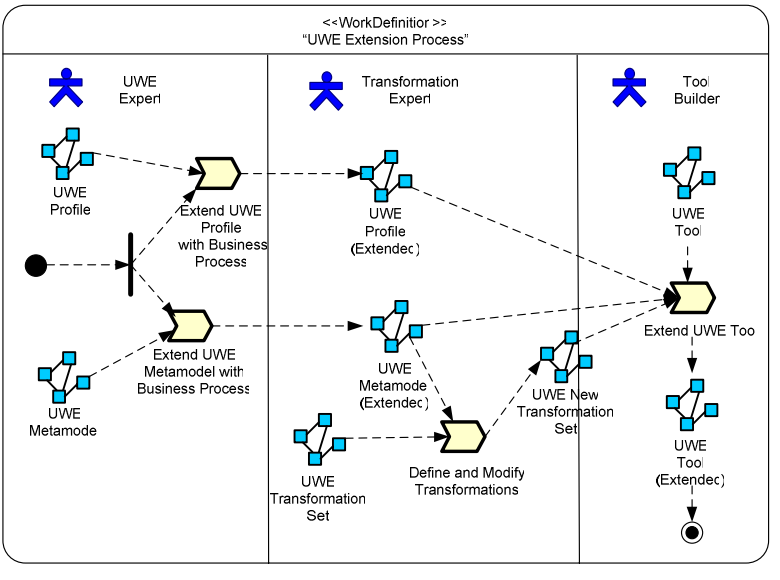

Fig. 6. The UWE extension process for adding the business process concern

3.2 Merging UWE Models with a Model of Another Approach

Instead of extending the UWE approach with new modeling constructs, an alternative is to use modeling features of another approach covering the new concern, i.e. the business process in our example. We choose OOWS [14] to illustrate the merging process. OOWS models business processes in BPMN [3] notation, which is used to generate WS-BPEL code. For OOWS, the business process is a concern depending on their functional and structural models (this is the most complex case of the three types of concerns we looked at in the classification presented in Section 2).

Thus, merging the OOWS business process with UWE requires firstly identifying those UWE models with direct correspondences to OOWS models. Identifying those models is strictly necessary for generating the process model in OOWS and requires the definition of a set of horizontal transformation rules from the UWE content and

requirements models to the OOWS business process model in order to implement the correspondences. Once these transformations are defined, OOWS can generate the code corresponding to the business logic that has been modeled.

However, the objective of merging UWE and OOWS does not end there. It is required in some way to integrate the information of the process model with the other UWE concerns (i.e., to link the business logic with the other views of the UWE system models). To do this, the next step is to determine what relation the new concern has with those already considered by the method. These dependencies must be identified and dealt with at the modeling level. In the case of UWE, it was decided that the process model would provide input for the navigation and presentation models. In order to keep those dependencies, it is necessary to define transformation rules from the OOWS business process model to the UWE navigation and presentation models as well as to establish some links between the code generated using UWE that relates the user interface with the implementation of the business logic generated with OOWS.

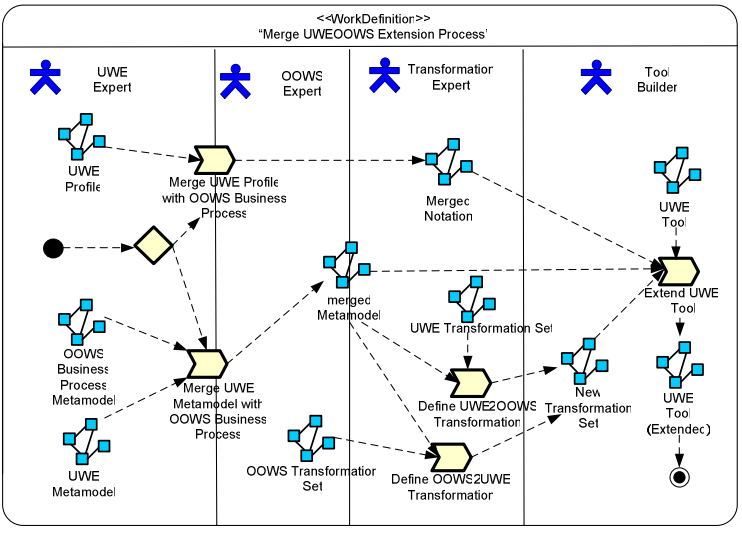

Fig. 7. The Merge UWEOOWS extension process for adding the business process concern

Fig. 7 illustrates these tasks and the experts responsible for them. Essentially, merging the UWE approach with a different one, such as for example OOWS, for modeling processes, means that the UWE expert must combine the UWE metamodel with the subset of the OOWS metamodel that deals with modeling Business Process. The resulting combination requires that a certain graphical notation must be incorporated into the new modeling elements of the UWE metamodel dealing with the processes. If the merge is done at the graphic design level, the transformations previously referred to would still need to be defined so that the UWE modeling elements correspond to the OOWS modeling elements and viceversa: in Fig. 7, the "transformation expert" is in charge of this task. Finally, the tool builder may construct a new code generation tool for UWE and the OOWS process model added,

using the original tool and the newly defined notation and set of transformations as the basis. Obviously, all this process is simplified considerably if instead of addressing a dependent concern we address an independent concern since the first one requires working with at least two tools.

3.3 Adding the Process Concern Using the WEISA Approach

As an alternative to extending UWE and merging UWE and OOWS, this section presents a new model-driven Web approach called WEISA that aims at obtaining interoperability and extensibility through a common metamodel (defined by WEI [18, 11]) based on the consensus of the most important Web methodologies regarding functional concerns; (2) WEISA also proposes a model-driven development process that provides the necessary extensibility able to incorporate a new concern with the lowest possible cost. Moreover, this process introduces an early representation of the software architecture guided by WebSA [9] which permits to reduce the complexity of the Web design with a small set of models and provides a closer match between the system modeled and the final implementation.

Fig. 8 presents the WEISA extension process which permits to add a new concern that comes from any MOF-compliant methodology in two different ways: (1) if WEISA contemplates the mechanisms for modeling this concern, it only requires the definition of horizontal transformations from the models that the initial method does contemplate; (2) If WEISA does not contemplate the requested concern, it is necessary extending their metamodel and studying the extension with a third model proposal that defines this concern. The last step consists on establishing the vertical transformations for introducing the new concern into the different types of WEISA components.

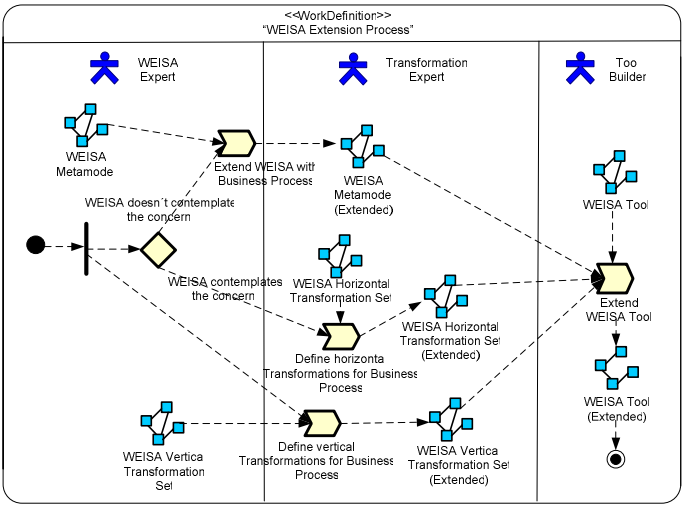

Fig. 8. The WEISA extension process for adding the business process concern

In the case study of this paper, we can apply the first possibility, in which WEISA represents the new process concern with its own process models, and obtains the rest of models (i.e. presentation, navigation and domain models) from UWE through a set of horizontal model transformations called UWE2WEISA. These transformations are defined in ATL. In fact, one of the major advantages of our proposal is its ability to design and implement Web applications reusing existing models from other Web engineering methods.

WEISA represents the process concern using the BusinessLogicStructure Model of WEI shown in Fig. 9. This model is a UML2 [12] stereotyped class diagram that establishes the main classes and operations that implement the business logic from our application. From here, we describe the behavior of each method by means of an stereotyped activity diagram, where the new stereotypes model the operations invoked from the user interfaces and the structure data returned by the business logic and visualized in the interface. Here we describe the structural business aspects, without delving into the behavioral business aspects. However, the interest reader in WEI profiles may refer to [12] for more details.

At the same time, the WEISA designer defines the software architectural model (called Configuration Model, CM) which uses the Web component as architectural unit, and defines around it a set of specific type of components of the Web application family (e.g. Controller, ServerPage, ProcessComponent, etc.). These kinds of components allow structuring the functionality of a Web application according to a given architectural style. Thus, this model provides a representation of the software architecture of the system, orthogonally to its functionality, thereby allowing for its reuse in different Web applications.

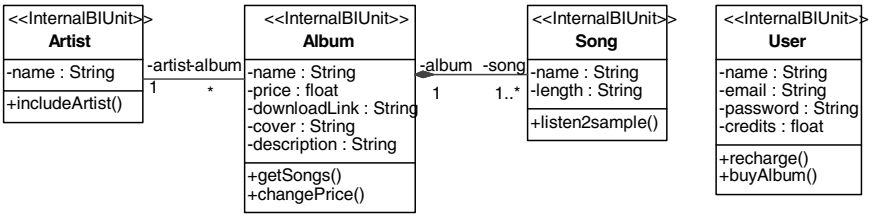

Fig. 9. The Business Logic Structure model of music portal example

Fig. 10 depicts the CM of the case study. The front-end part of the model shows a ServerPage component which receives the user's requests and renders the response in a PC browser. The ServerPage also has a reference to EntityData in order to represent the functionality and is responsible for sending messages to the Controller. At this point, each ProcessComponent PC receives the requests through the BusinessFaçade ServiceInterface from the Controller, and re-sends them to the Entity. Finally, the Entity references to a DataAccessComponent called DAC in order to store and to recover data from a database.

After the models defined by the WEISA designers are completed, these become the entry point of the Merge2Design transformation which converts the functional and

architectural models into a detailed design model represented by the WEISA Integration Model.

This complex and extensible transformation is based on the concatenation of a set of smaller transformations associating each type of component with a concern (e.g. the data model is related to the data access component, the ServerPage to the model presentation, the EntityWeb to the domain model, the ProcessComponent to the process model, etc.). This provides us with the integration model representing the design components that constitute the Web application where we have introduced the functional content from the functional models. Finally, this process establishes a model-to-text transformation called Integration2Platform that allows us to obtain the final implementation. This is a model-to-text transformation that obtains the code from the integration model and the functional models requested by different concerns.

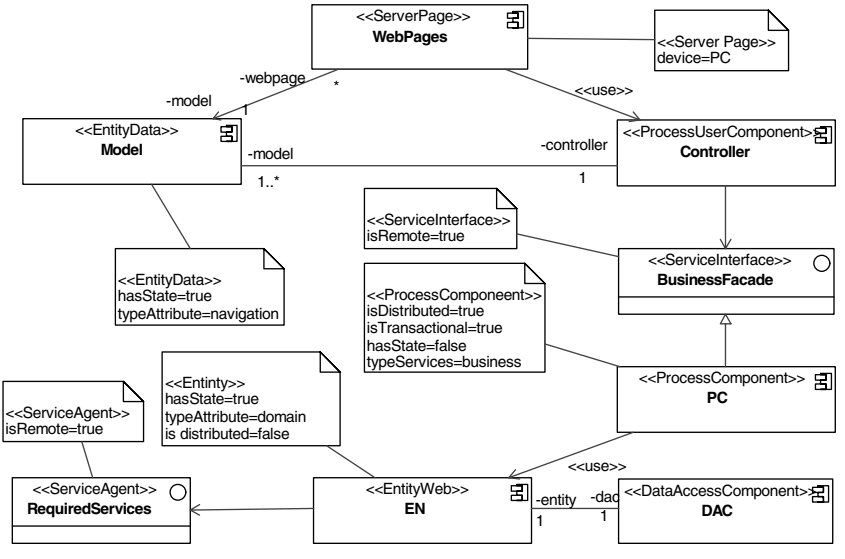

Fig. 10. The Configuration Model of the music portal example

In the case study, we focus on the Merge2Design Tranformation part in charge of introducing the process concern into the design model. More specifically, the ProcessComponent and the EntityWeb are the only ones which obtain the data from the process concern. Fig. 11 shows a fragment of integration model that represents the ProcessComponent and EntityWeb components obtained from the BusinessLogicStructure classes such as Artist, Album and Song.

Finally, this process establishes a model-to-text transformation called Integration2Platform that allows us to obtain the final implementation. This transformation obtains the code from the integration model and the functional models requested by different functional concerns.

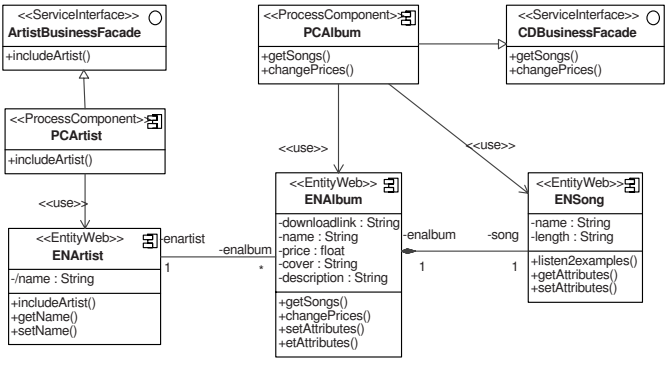

Fig. 11. A fragment of the Integration Model of the music portal example

Table 1. Comparing alternatives for adding a new concern

Criteria	Proprietary Method (UWE)	Merging 2 Methods	WEISA
Abstract syntax	Adding modeling elements to original metamodel.	Putting both metamodels in relation (when it is possible) maybe by means of a third one.	Defining an independent metamodel and putting it in relation with the others.
Notation	Extending UML profile with new artifacts.	Two options: (a) Using, notations of each method for modeling the concerns (b) defining a new notation for modeling element results of the merge.	Extending UML profile with new artifacts.
Transformations	Vertical transformations for the concern added and modifying the others in order to guaranty the consistency between all concerns.	Defining vertical transformations for the concern added; one transformation for each method.	Defining vertical transformations for concern added without modifying previous one. Other methods will benefit of new trans-formation without change.
Tool	Extending graphical interface and transformation engine of CASE tool to include modified vertical transformations.	Importing ad-hoc of all the models in a new environment The total set of transformations rules also have to be included in the merging environment.	Extending graphical interface and adding only one transformation to the transformation engine of the CASE tool.
Interoperability/ Extensibility	Representing different concerns with its own models.	Representing the concerns with both methods.	Representing different concerns with WEISA or with any approach with a MOF metamodel.

4 Analyzing and Comparing the Three Alternatives

Like most design decisions, the choice of any of the previous alternatives may have far-reaching consequences for a Web engineering method. We can indeed find

arguments for and against each way proposed. In this sense, we hope this study will not only make it easier to understand the relative strengths or weakness of each strategy, but will also serve as a basis for analyzing the obstacles to evolve or maintain a method. As a summary, Table 1 identifies the most relevant features of a model-driven Web engineering approach, which can be affected as a consequence of the evolution process when addressing a new concern. Although there are other evaluation criteria, the following may provide a good basis for comparing the three alternatives:

Abstract syntax. The abstract syntax of a modeling language describes the vocabulary of concepts provided by the language, the definition of such concepts and the relationships that exist between them. It also establishes how the concepts may be correctly combined to create models by means of a metamodel definition. Therefore, adding a new concern may involve a review of the current syntax in order to identify if the new concern added requires specific modeling concepts. Furthermore, the method must decide where the new artefacts are going to appear within a system description, (i.e, as a part of an existing concern, in a new model, etc.).

Notation. All methodologies provide a notation that facilitates the presentation and construction of models in their associated languages. There are two main types of concrete syntax or notation: textual and visual. In any case, when the abstract syntax is affected by a change then normally this change will carry out notation changes.

Development process. In model-driven Web development methods, changes to abstract syntax must be also mapped into changes to the development through the definition of transformation rules: (a) on the one hand, it may imply the definition of vertical transformations that convert models from a higher to a lower level of abstraction and (2) on the other hand, it may suppose the definition of horizontal transformations which describe mappings between models of the same level of abstraction.

Tool. New transformation rules must be integrated into the CASE tool that supports the Web engineering method. However, not only the code generation engine requires changes but also the model editor of the same tool. This task may be more complex and time-consuming than we expect, especially when the CASE tool design was not prepared to assume the evolution.

Interoperability/Extensibility. The ability to extend a system and the level of effort required to implement the extension is a measure of its extensibility. The central theme is to provide for current and future changes while minimizing the impact to the existing proposal.

All three alternatives have their own advantages and disadvantages, and therefore it is particularly difficult to offer general guidelines on when a designer should opt for one or for another. At first sight it could be considered, for example, that in those cases in which we deal with an *orthogonal concern* the least expensive decision at the implementation level is the merging or the WEISA approach. On the contrary, in the cases where we deal with the *dependent concern*, it may be more appropriate to directly extend the methodology.

Although the influence of the type of concern is extremely relevant for implementing the method extension, other factors such as the semantic distance between the source and target metamodels may be equally relevant. This semantic distance would determine the complexity of the transformations to be implemented.

5 Related Work

As far as we know, there are not studies in the Web engineering domain that analyze the real impact of extending a design method with an additional concern. However, we have found proposals in other research areas that address relevant and related issues to the proposed research topic of this article.

In the product-line context, released products are built on various versions of core assets and glued together with product specific code. Thus, the domain evolution problem (i.e., the metamodel evolution in our case) arises when existing product-line must be extended and/or refactored to handle unanticipated requirements. Clements and Northrop list in [4] a set of metrics to measure the opportunities for future asset or infrastructure of a certain product-line. These metrics can be adapted and applied on the three alternatives we propose here.

In [5] we find an interesting classification of the changes that may occur in a metamodel. According to the authors, changes can be grouped into three categories: (a) changes that preserve the semantics of the metamodel, (b) changes that introduce new classes and/or properties to the metamodel, (c) changes that remove/destroy elements of the metamodel. Adding a new concern has always effects on a metamodel. In particular, orthogonal and dependent concerns introduce new artefacts and extend the semantic of the original metamodel. On the contrary, replacement concerns may modify it by removing key elements of the metamodel so they require especial attention.

On the problem of synchronizing models with evolving metamodels, [17] introduces and outline an approach to addressing it efficiently. The authors aim to minimize the effort required to perform model migration in face of metamodel changes (some of them are shown in Table 1).

6 Conclusions and Future Work

Model-driven development (MDD) is being adopted due to its advantages of portability and facilities for the integration of models produced by different approaches, which is supported by model transformations. In particular, MDD is applicable in the Web application domain as a very clear separation of concerns is one of the main characteristics of almost all Web engineering methods.

Another advantage of MDD is the flexibility when introducing a new concern that is part of the evolution of Web methodologies. Including a new concern may be more or less difficult depending on the type.

This paper presents a classification of concerns and a discussion on three different alternatives for addressing the evolution of Web engineering methods. The more traditional way is the adaptation of the own method extending it with an additional

modeling concerns. Another alternative is merging the original proposal with another approach covering the specific concern. Finally, we propose a new approach called WEISA based on an interoperable and architecture-centric process that aims to reduce the impact of adding a new concern. A table comparing the three alternatives is presented as well.

We will continue working on the variants detailed, completing the models with dynamic aspects and we will define the complete set of transformations required for the MDD process. In addition, adding a new concern to a Web engineering approach may in general affect three different dimensions: the way of modeling, the way of working (i.e., the associated methodology), and the supporting environment and tools. The approach presented here has focused on the way of modeling first, because the changes to other two dimensions depend on the alternative selected to incorporate the new concern at the modeling level. Once we have identified the alternatives, the next step is to study their potential impact on the other two dimensions.

Acknowledgements. This work has been partially funded by projects Desarrollo de Software para Sistemas Distribuidos Peer-to-Peer (TIN2005-09405-C02-01), MOVIS (P07-TIC-03184), EU project SENSORIA (IST-2005-016004), ESPIA (TIN2007-67078). We would also like to thank the reviewers for their insightful comments and suggestions.

References

1. ATL ATLAS Transformation Language project, `http://www.eclipse.org/m2m/atl/`
2. Bézivin, J.: In Search of a Basic Principle for Model Driven Engineering. UPGRADE V(2), Novática (2004)
3. Business Process Modeling Notation (BPMN) Version 1.0 - OMG Final Adopted Specification (February 6, 2006)
4. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Reading (2001)
5. Gruschko, B., Kolovos, D., Paige, R.F.: Towards Synchronizing Models with Evolving Metamodels. In: Proc. of 11th Workshop on Model-Driven Software Evolution (MODSE 2007) (2007)
6. Koch, N., Knapp, A., Zhang, G., Baumeister, H.: UML-based Web Engineering: An Approach based on Standards. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modelling and Implementing Web Applications. Springer, Heidelberg (2007)
7. Koch, N., Kraus, A., Cachero, C., Meliá, S.: Integration of Business Processes in Web Applications Models. Journal of Web Engineering (JWE) 3(1), 22–49 (2004)
8. Linaje, M., Preciado, J.C., Sánchez-Figueroa, F.: Engineering Rich Internet Application User Interfaces over Legacy Web Models. IEEE Internet Computing 11(6), 53–59 (2007)
9. Meliá, S., Gomez, J.: The WebSA Approach: Applying Model Driven Engineering to Web Applications. Journal of Web Engineering 5(2), 121–149 (2006)
10. Meliá, S., Kraus, A., Koch, N.: MDA Transformations Applied to Web Application Development. In: Lowe, D.G., Gaedke, M. (eds.) ICWE 2005. LNCS, vol. 3579, pp. 465–471. Springer, Heidelberg (2005)

11. Moreno, N., Vallecillo, A.: Towards Interoperable Web Engineering Methods. Journal of the American Society for Information Science and Technology (JASIST) 59(7), 1073–1092 (2008)
12. Moreno, N., Vallecillo, A.: Modeling Interactions between Web Applications and Third Party Systems. In: Proc. of the 5th International Workshop on Web Oriented Software Technologies (IWWOST 2005) (2005)
13. Object Management Group (OMG), http://www.omg.org
14. Torres, V., Giner, P., Pelechano, V.: Web Application Development Focused on BP Specifications I Taller sobre Procesos de Negocio e Ingeniería del Software (PNIS) (2007)
15. Unified Modeling Language (UML). Superstructure, version 2.1.2. Specification, OMG, http://www.omg.org/cgi-bin/doc?formal/07-11-01
16. UML-based Web Engineering (UWE), http://www.pst.informatik.uni-muenchen.de/projekte/uwe/
17. Wachsmuth, G.: Metamodel Adaptation and Model Co-adaptation. In: Ernst, E. (ed.) ECOOP 2007. LNCS, vol. 4609, pp. 600–624. Springer, Heidelberg (2007)
18. Web Engineering Interoperability (WEI), http://www.lcc.uma.es/~nathalie/WEI/