Decoupling web application concerns through weaving operations

Antonio Cicchetti, Davide Di Ruscio

Dipartimento di Informatica, Università degli Studi dell’Aquila, I-67100 L’Aquila, Italy

Received 19 January 2007; received in revised form 4 July 2007; accepted 3 October 2007
Available online 17 October 2007

Abstract

Today’s Web applications require instruments and techniques able to face their complexity which has noticeably increased at the expense of productivity and quality factors. A number of design methodologies have been proposed in the process of trying to provide developers with languages and tools to abstract and capture Web applications under orthogonal views, like data, navigation and presentation. While the different modeling language constructs can be unified in a common metamodel, consistency among the distinct concerns is guaranteed by less formal relations. Usually, they are based on name conventions and/or ad hoc tool support that could affect reuse and maintenance ratings of specifications.

In order to define rigorous and explicit correspondences between the artifacts produced during a system development, this paper proposes the exploitation of dedicated weaving models. The approach aims at providing structural mappings that do not interfere with the definition of the views on either side achieving a clear separation between them and their connections. Furthermore, following the “everything is a model” principle, this work can enable the use of general-purpose theories and tools. For example, model transformations can be applied to evaluate the given specifications or to derive alternative descriptions like Webile or WebML.

Keywords: Abstract state machines; Model transformations; Model weaving; Web applications; Metamodel specification

1. Introduction

Today’s Web applications require instruments and techniques able to face their complexity which has noticeably increased at the expense of productivity and quality factors. To cope with the technical difficulties of these systems many design methodologies have been proposed like Hera [1], OO-H [2], OOHDPM [3], UWE [4], W2000 [5], WebML [6] and Webile [7]. All of them adopt a number of notations, even if as expected many concepts are similar and could constitute a common metamodel for the Web domain [8]. In particular, these methodologies propose several views comprising at least a conceptual model, a navigation model and a presentation model although with different terminologies. While the constructs specifying such aspects can be precisely unified, consistency among them is guaranteed by less formal relations. Usually, models are kept related through name conventions exploiting shared namespaces that occur on each of them or by means of tools that use internal mechanisms hidden from the developer. The consequent lack of abstraction in the separation between the concerns and their connections could...
hamper some quality factors, like reuse of models which results in intertwined and not autonomously maintainable
ones. Furthermore, having models that explicitly express relations amongst the source view specifications is a
necessary prerequisite to the use of general-purpose theories and for enabling tool chains [9]. This paper proposes
weaving models [10] to specify formal relations between the different views produced during the development of
Web applications. The weaving models do not interfere with the definition of the views on either side, achieving a
clear separation between them and their connections. Furthermore, designers can gain a deeper understanding about
the explicit dependencies between the parts, and they are able to recognize the consequences of local changes to
the whole system. Finally, the weaving descriptions enable the automatic processing and manipulation of the related
models by executing operations based on the link semantics.

In the proposed approach, the source views are woven together according to weaving models whose semantics
is given by means of automated model transformations that are mathematically specified through Abstract State
Machines (ASMs) [11]. The proposed constructs for the view specifications are inspired by [8] which represents
a step towards a common reference metamodel for Web modeling languages. The views and the weaving models
conform to metamodels that are precisely defined in KM3 [12] which is a text-based language conceived for describing
metamodels. A prototypical implementation of the approach is available at [13] to support the development of all
the source artifacts by using graphical editors which have been realized through the Eclipse Graphical Modeling
Framework (GMF) [14]. Furthermore, XASMs-based transformations [15] are provided to define the semantics of
given weaving operators and to generate target woven models.

The presentation of the approach is based on a running example aimed to develop a simple Web application. Once
the source views are defined, they are related and kept consistent with respect to given weaving descriptions. In turn,
such descriptions enable the generation of models specified with two target modeling languages that are Webile and
WebML. The source concerns are produced by using simple metamodels in order to give a flavor of the approach
which mainly focuses on the weaving operations; discussions about the limited expressiveness of such metamodels is
beyond the scope of this paper.

The structure of the paper is as follows: the next section describes the problem statement, and sketches our proposed
solution. Section 3 illustrates the metamodels used to describe the source views that will be related through the
weaving models discussed in Section 4. In Section 5 a model transformation approach based on ASMs is summarized.
Then it will be used in Section 6 to obtain the automated transformations to weave together the source concerns. After
a discussion reported in Section 7 and related the work presented in this paper with other approaches, Section 9 draws
the conclusions and presents some perspective works.

2. Dealing with web application concerns

Most of the current methodologies for Web application development propose a number of views comprising at least
a conceptual, a navigation and a presentation model. The first one consists of the data specification the application
being modeled is based on. Usually well-known object-oriented modeling principles or Entity/Relationship (ER)
diagrams [16] are used for this purpose. The navigation model describes those objects the user can reach: by means
of concepts like Node, Link and their specializations, the designer specifies the paths and eventually the access
primitives which are usual in hypermedia applications. Finally, the presentation model specifies how navigation nodes
have to be graphically arranged in the presentation space by means of concepts like Location and its specializations.

The constructs provided by the available methodologies aiming at specifying the concerns of a Web application
can be precisely unified into a common metamodel like in [8]. On the contrary, the consistency among the views is
guaranteed by less formal relations. In fact, the formalisms specify under which conditions the views can be
integrated by name conventions and/or ad hoc tool support. For instance, Fig. 1 presents a small fragment of an
OO-H specification where the upper and the lower sides correspond, respectively, to (portions of) the conceptual and
navigation models of a conference review system given in [17]. The models are kept connected by means of a common
namespace which occurs on both sides. In particular, the Track and Conference entities in the conceptual model
are referred by means of compound class names whose form is nodeName:entityName such as Track:Track and
Conference:Conference nodes in the navigation model (by coincidence the names of both the nodes and the
entities are the same).

A similar problem affects WebML as shown in Fig. 2 where a fragment of the conference review system described
in [18] is modeled. The consistencies between the data and hypertext views are guaranteed by the WebRatio [19] tool
support according to the references embedded in the models. For example, in the navigation model presented on the right-hand side of the figure the data unit Conference, in the dashed part of the page Create subjects and tracks, refer to the data entity Conference of the data model on the left-hand side of the figure.

The consequent lack of abstraction in separating between the concerns and their hard-coded connections could reduce some quality factors of models making them intertwined, not autonomously maintainable and not fully reusable. For example, hypertext specifications with embedded references to data structures could not be suitable to design different systems sharing a part of the same navigation structure or page compositions. This paper stresses the application of the separation of concerns principle in Web engineering. As for any kind of software system, this principle states that a given problem involves different kinds of concerns which should be identified and separated to cope with complexity, and to achieve the required engineering quality factors such as robustness, adaptability, maintainability, and reusability [20]. To improve the separation of concerns in Web application development, this paper proposes the approach depicted in Fig. 3. The application concerns (data, navigation and page composition) are described by means of separated models. Each of them conforms to a proper metamodel that represents a domain-specific language (DSL) [21] which is capable of specifying a particular view of a system. Being more precise, DSLs provide domain experts with prefabricated abstractions that are defined at the same level as their problem domain. Consequently, domain experts can quickly learn, apply and extend DSLs without a steep learning curve. For example, a metamodel for the data specification might include the concepts of data entity and data relationship that are well-known to the data designers. Moreover, DSLs do not include details that are unrelated to a specific problem domain.
For example, a DSL for the data specification does not contain constructs related to navigation- or page-composition perspectives. This permits domain experts to be concentrated only on their views without caring about entities that are not related to the problem they are solving.

According to the approach in Fig. 3, once the views are separately given, they are related by additional models, called weaving models [10] to explicitly describe the connections between the elements belonging to the different concern specifications. Originally introduced for metadata integration and evolution in databases [22], weaving models represent a useful technique also in software modeling. They can be used for setting fine-grained relationships between models or metamodels and executing operations on them based on the link semantics. Adhering to the “everything is a model” principle [10], model weaving offers a number of advantages. All the information, relationships and correspondences between the concerns, could be described by specialized weaving models avoiding to have large metamodels for capturing all the aspects of a system. Furthermore, metamodels focusing on their own domain can be individually maintained, and at the same time interconnected into a “lattice of metamodels” [10]. In other words, each metamodel could represent a domain-specific language dealing with a particular view of a system, while weaving links permit describing the aspects both separately and in combination.

Weaving models conform to precise metamodels defined and specialized for the given domain. In this paper, a weaving metamodel is proposed for specifying how to relate elements belonging to the Web application concerns. The proposed weaving metamodel provides one with modeling constructs devoted to the specification of links between data-, navigation- and page-composition views. The weaving links have a semantics which can be precisely specified in terms of model transformations [23]. The transformations depend on the weaving metamodels and are written once for all with respect to the semantics of the provided weaving link types. From a development process point of view, Web application designers provide separate concern specifications which are then linked explicitly by means of specific weaving models. Designers have knowledge of the used weaving links from a high level of abstraction only and do not know the complexity of the corresponding model transformations. In fact, link semantics is specified through transformations in advance, i.e. during the development of the involved DSLs.

In the proposed approach, Abstract State Machines (ASMs) are used to formally specify the transformation of the source concern specifications to generate woven descriptions with respect to the semantics of the weaving links.
ASMs have been proposed as semantic basis for model transformation by various papers and frameworks [23–26]. In this work, ASMs have been used since they capture in a mathematically rigorous form the fundamental operational intuitions of computing [27]. The provided notation has a simple syntax that permits one to write specifications that can be seen as “pseudocode over abstract data”. On the one hand they are mathematically rigorous and represent a formal basis to analyze and verify transformations; on the other hand, they combine declarative and procedural features to harness the intrinsic complexity of this task. The idea aims at formally specifying the behaviour of transformations in order to produce a formal and implementation independent reference which permits one to convey the design decisions recorded by the designer to the transformation implementors. In this way the transformation developers can check their implementations (written in a specific language like ATL [28], QVT [29], etc.) against an accurate and executable high-level model of the transformation itself.

Inspired by the Web development methodologies mentioned above and by the metamodel presented in [8], the approach proposes metamodels for expressing the data-, navigation- and page-composition perspectives without considering the presentation and behavioural ones (see Section 7). These metamodels could be extended by taking into account a number of available contributions [8,30] even if this work mainly focuses on the weaving operations and their applications for the Web domain.

In order to have a precise and formal definition of the metamodels, in this paper the KM3 [12] language and its tool support are used. KM3 is based on the same core concepts used in OMG/MOF [31] and EMF/Ecore [32] that are classes, attributes and references. Compared to MOF and Ecore, KM3 is focused on metamodeling concepts only. For instance, Java code generation facilities are offered by Ecore are not supported by KM3. The use of KM3 is mainly justified by its simplicity and flexibility to write metamodels and to produce domain-specific languages. A number of experimental KM3 metamodels have been specified both from academia and industry and are currently collected into a library that can be found at [33]. Furthermore, the available tool support is able to generate Ecore and MOF metamodels corresponding to the given KM3 specifications. This facility has been very helpful for developing the prototypical implementation of the approach discussed in this paper. In fact, the GMF-based graphical editors of the source concerns and weaving descriptions are developed on top of the corresponding metamodels that have to be necessarily expressed in Ecore. In this sense the KM3 to Ecore facilities of the KM3 tool have been exploited.

In the sequel, each phase of the approach (see Fig. 3) is explored starting from the next section where the metamodels devoted to the definition of the Web application perspectives are illustrated.

3. Concern specifications

This section illustrates the metamodels that will be used for describing the data (Section 3.1), navigation (Section 3.2) and page-composition (Section 3.3) views of Web applications according to the left-most side of Fig. 3. The discussion is based on a running example consisting of a simple academic Web site that will be considered in the presentation of the overall approach. The sample application is intended to provide information about departments, affiliated professors and papers which have been published. From the index of departments, the user may access the description of a selected one, e.g. the list of all professors affiliated to that given department, who in turn can be further selected to access the details in their homepage, including the publication list.

3.1. Data modeling

The specification of data on which the system under study is based will be developed by exploiting ER modeling principles giving place to the metamodel in Fig. 4 and the KM3 code in Listing 1. In particular, Entities
represent common features that can have typed Attributes and can be associated with each other by means of Relationships. For each of the entities involved in a relationship, a corresponding Role description is given.

```
class DataModel {
    reference entities[0-*] container : Entity;
    reference relationships[0-*] container : Relationship;
}
class Entity {
    attribute name : String;
    reference attributes[0-*] container : Attribute;
}
class Attribute {
    attribute name : String;
    attribute contentType : String;
}
class Relationship {
    attribute name : String;
    reference source container : Role;
    reference target container : Role;
}
class Role {
    attribute minOccurs : String;
    attribute maxCard : String;
    reference entity : Entity;
}
```

Listing 1. KM3 specification of the data metamodel.

The KM3 specification of the data metamodel is canonically obtained by taking into account the following rules: each metaclass of the metamodel is defined by the keyword `class`; the keyword `attribute` is used for defining meta-attributes of the metaclass being specified. The relationships between metaclasses are declared by using the keyword `reference`. If a given relationship is a composition (like the one between the metaclasses `Entity` and `Relationship` in Fig. 4) the attribution `container` is added to the reference definition. In the rest of the paper, for presentation purposes the metamodels will be graphically represented only, the interested reader can consider the corresponding KM3 specifications available for download at [13].

According to the sample application requirements, the proposed metamodel can be used for specifying the data model shown in Fig. 5. In particular, the conceptual structure consists of departments (modeled with the data entity `Department`) which have several professors (`Professor`) and each of them has a number of publications (`Publication`). The direction of the relationships specifies the kind of subordination amongst the entities whose purpose will be clarified in the rest of the paper.

3.2. Navigation modeling

The navigation view describes the paths a user can follow in terms of reachable nodes connected through links. This view gives only the navigation map of the application without defining, for instance, the data that will be published or the link properties, i.e. whether a link should propagate relevant information to retrieve data in the target node.
Borrowing concepts from [8], a navigation metamodel is proposed in Fig. 6 consisting of directed links (NavigationLink) and nodes (NavigationNode). This is used to define the navigation model of the running example (see Fig. 7) which is made up of four nodes (Departments, Professors, ProfessorHomePage, Publication) connected by links with respect to the application requirements.

3.3. Composition modeling

The structure of pages is captured by a composition model abstracting from data and navigation details. These information will be available once this model will be related to the navigation and data descriptions (see Section 4). Initially, for each page the designer defines the name and the available contents only and, in order to distinguish whether the data that will be published activate some link or not, the types index or data are used, respectively (see Fig. 8).

The composition model of the running example is provided in Fig. 9: it specifies the page ProfessorHome as consisting of two contents, ProfInfo and Pubs, respectively. Such contents will be fed later on by the proper data according to the weaving models which will be introduced in the rest of the paper.

4. Weaving specification

Once the different concerns of a Web application are specified, they have to be related and kept consistent with respect to the application requirements. For instance, Fig. 7 represents a navigation topology without taking into
account the data that will be filled in the pages. Furthermore, the structure of each page is specified regardless of its location in the navigation structure.

As said above, the weaving operation is proposed to support such a decoupling among models. The concept of weaving is not new and the definition of model weaving that will be considered in this work is that provided by Didonet Del Fabro et al. in [34]. They leverage the need of a generic way to establish model element correspondences by proposing a solution aimed at reaching a trade-off between genericity, expressiveness and efficiency of mappings which are considered as models that conform to a weaving metamodel. The general operational context of the model weaving is depicted in Fig. 10. It consists of the production of a weaving model \( WM \) representing the mapping between the metamodels \( LeftMM \) and \( RightMM \). Like other models, this should conform to a specific weaving metamodel \( WMM \). Weaving operations may be applied to models instead of metamodels. This is the case of what is proposed in this work where weaving models are produced in order to relate different concerns in an explicit and precise way.

By going into more details, a metamodel inspired by [34] is produced and depicted in Fig. 11. With respect to this metamodel, a weaving model (\( WModel \)) consists of elements (\( WElement \)) related through weaving links (\( WLink \)). According to the different kinds of elements involved in weaving operations, Data\( WElement \), Composition\( WElement \) and Navigation\( WElement \) specialize the \( WElement \) concept. Moreover, \( WLink \) is specialized into CompositionNavigation\( WLink \) and DataComposition\( WLink \) because of the different kinds of links that can be defined between composition and navigation models, or data and composition models, respectively.

Linked elements belonging to the composition and data models specify the correspondences between each page content defined in the composition model and the data entities from which the information has to be retrieved. In this case the weaving links have the attribute restricted to denote whether the data collection has to be filtered with respect to information local to the page the content has to be delivered. For example, in the weaving model in Fig. 12 the content \( Profs \) is connected with the \( Professor \) entity, and the association has the attribute restricted set to \( true \). This denotes that the information forwarded by the incoming links of the \( ProfessorsList \) page will be used for filtering the data that will be retrieved and then published to the user. This information forms the context of the page whose semantics is defined by weaving together the composition and the navigation models by means of CompositionNavigation\( WLink \) elements. For instance, in Fig. 13 the page \( ProfessorHomePage \) of the navigation model is linked to the page \( ProfessorHome \) of the composition model.

The weaving operation can be supported by heuristics raising its automation level. These heuristics are closely related to ontology and to schema matching approaches that aim at discovering semantic relationships between...
elements of different ontologies or schemas [35]. In our case, these approaches might help the creation of (fragment of) simple weaving models even though this is not always possible.

Weaving models induce complex computations to derive information that can be obtained only by performing queries and analysis over the related models (see Section 7). In the remainder of the paper a discussion on how to deal with non-trivial situations by means of ASMs-based model transformations is presented. Subsequently, Webile and WebML models are generated from the given source concerns and weaving specifications.

5. ASM-based model transformation

Despite the increasing relevance of model transformations to software development and integration, at the moment none of the existing approaches have been universally accepted in the same way EBNF is commonplace for syntax specification, for instance. Nevertheless, according to our experience, ASMs represent a formal and flexible platform for specifying model transformations: on the one hand they are mathematically rigorous and represent a formal basis to analyze and verify that transformations with respect to some criteria [36]; on the other hand, they combine declarative and procedural features to harness the intrinsic complexity of such task.
ASMs have been proposed as semantic basis for model transformation by various papers and frameworks [23,24]. Some of them combine ASMs with other approaches like graph transformations [25,26]. The differences between such approaches and the one proposed in this section are discussed in Section 8. However, in this work an approach completely based on ASMs is proposed in order to have a unique formalism for specifying the semantics of proposed weaving models. This transformation approach has already been proposed and validated by the authors on different applicative domains, such as data-intensive Web applications, middleware-based systems and for software architecture modeling [37]. In the following, the approach is illustrated by introducing in Section 5.1 some preliminaries about ASMs and then explaining how to encode models and metamodels that will be manipulated and transformed through transformation rules.

5.1. Abstract state machines

ASMs, introduced by Yuri Gurevich in 1988 [38], offer a framework for high-level system design and analysis. ASMs methodologies are practical in modeling and analyzing different sizes of systems belonging to widespread areas like hardware and software architectures, programming languages and network protocols [27]. Furthermore, ASMs offer methods for specifying algorithms and systems at a highly abstract level by providing Turing-complete machines.

ASMs form a variant of first-order logic with equality, where the fundamental concept is that functions are defined over a set \( \mathcal{U} \) and can be changed point-wise by means of transition rules. The set \( \mathcal{U} \), referred to as the superuniverse in ASM terminology, always contains the distinct elements \( \text{true} \), \( \text{false} \), and \( \text{undef} \). Apart from these, \( \mathcal{U} \) can contain numbers, strings, and possibly anything, depending on the application domain.

Systems can be modeled as sequences of state transitions which are captured through ASMs rules executed if corresponding predicates are verified. Being slightly more formal, we define the state \( \lambda \) of a system as a mapping from a signature \( \Sigma \) (which is a collection of function symbols) to actual functions. We write \( f_\lambda \) for denoting the function which interprets the symbol \( f \) in the state \( \lambda \). Subsets of \( \mathcal{U} \), called universes, are modeled by unary functions
from \( \mathcal{U} \) to \{true, false\}. Such a function returns true for all elements belonging to the universe, and false otherwise. A function \( f \) from a universe \( U \) to a universe \( V \) is a unary operation on the superuniverse such that for all \( a \in U \), \( f(a) \in V \) or \( f(a) = \text{undef} \). The universe Boolean consists of true and false. A basic ASM transition rule is of the form

\[
f(t_1, \ldots, t_n) := t_0
\]

where \( f(t_1, \ldots, t_n) \) and \( t_0 \) are closed terms (i.e. terms containing no free variables) in the signature \( \Sigma \). The semantics of such a rule is: evaluate all the terms in the given state, and update the function corresponding to \( f \) at the value of the tuple resulting from evaluating \( (t_1, \ldots, t_n) \) to the value obtained by evaluating \( t_0 \). Rules are composed in a parallel fashion, so the corresponding updates are all executed at once. Apart from the basic transition rule shown above, there also exist conditional rules where the firing depends on the evaluated boolean condition term, do-for-all rules which allow the firing of the same rule for all the elements of a universe, and lastly extend rules which are used for introducing new elements into a universe. Transition rules are recursively built up from these rules.

The ASM-based transformation rules are executable and several compilers and tools are available both from academia and industry implementing most of the ASM constructs. In the reminder of the paper, ASM rules are given in the XASM dialect compiler [15] which has already been used by the authors for implementing other proofs of concepts. However, any execution environment could be used alternatively since the approach exploits the basic constructs of ASMs provided by various ASM implementations like ASM Workbench [39], AsmGofer [40], Asm2C++ compiler [41], and .NET AsmL [42].

5.2. Model and metamodel encoding

ASM-based model transformations start from an algebra encoding the source model and return an encoding of the target one (see Fig. 14). This final encoding contains all the needed information to translate the resulting algebra into the corresponding model through a pretty printing operation.

The signature of the algebra encoding a model is canonically induced by the corresponding metamodel whose elements define sorts and functions as for instance in Fig. 15: the reported sample metamodel borrows concepts from the Web Application Extension (WAE) [43] conceived by J.Conallen who proposed a UML profile for modeling Web applications. According to this extension Web pages are modeled by giving both server-side and client-side aspects by means of ServerPage and ClientPage elements, respectively. A server page can be associated with other server-side objects. A ClientPage represents a HTML page which is usually associated with other client or server pages. In the last case the build element is used to state that a server page builds a client one. A hyperlink between pages is modeled by link elements. If the hyperlink includes parameters, they are modeled as link attributes of the association by means

\[
\Sigma = (S, OP)
\]

\[
S := \{\text{Page}, \text{ServerPage}, \text{ClientPage}, \text{Link}, \text{Build}, \text{DataType}\}
\]

\[
OP :=
\]

\[
\text{name} : \text{Page} \rightarrow \text{String}
\]

\[
\text{name} : \text{DataType} \rightarrow \text{String}
\]

\[
\text{source} : \text{Build} \rightarrow \text{ServerPage}
\]

\[
\text{target} : \text{Build} \rightarrow \text{ClientPage}
\]

\[
\text{target} : \text{Link} \rightarrow \text{ServerPage}
\]

\[
\text{params} : (\text{Link}, \text{Parameter}) \rightarrow \text{Bool}
\]

\[
\text{belong} : (\text{Link}, \text{Parameter}) \rightarrow \text{Bool}
\]

\[
\text{type} : (\text{Parameter}) \rightarrow \text{DataType}
\]
of Parameter instances. This metamodel induces the signature \( \Sigma \) (on the right-hand side of Fig. 15) composed of sorts \( (S) \) and functions \( (OP) \). In particular, for each metaclass of the metamodel a correspondent set in \( S \) is available. Functions are induced by meta-attributes, meta-associations, and roles. For instance, the attribute name of Page gives place to the function name: Page \( \rightarrow \) String. Moreover, in order to specify the source and the target of a Build association, the functions source: Build \( \rightarrow \) ServerPage and target: Build \( \rightarrow \) ClientPage are provided, respectively. Multiple meta-associations are encoded by means of relations.\(^1\) For instance, a given link can have a number of parameters as stated by the role params of the meta-association between the Link and Parameters metaclasses. In this case the relation params in \( OP \) is provided and it will be true if a parameter belongs to a given link, false otherwise. Meta-associations which are compositions induce the definition of the relation belong. For instance, in the case of the composition between Link and Parameter, the relation belong: (Link, Parameter) \( \rightarrow \) Bool is defined and it is true for each couple \((l,p)\) such that params\((l,p)\)=true.

The approach permits the encoding of specializations in the metamodels by means of subsorting. For instance, the inheritance between ServerPage and Page is encoded by means of the subsorting relation ServerPage \( \prec \) Page.

The sets and the functions induced by a metamodel are used for encoding models that conform to the given metamodel as in Fig. 16 where the encoding of a sample WAE model is depicted. In particular, on the lower side of the figure the sets and the functions defined in Fig. 15 are updated according to the instance diagram on the upper side of the figure. The canonical encoding of metamodels and models can be performed in an automatic way as discussed in [44].

5.3. Model transformation rules

During the specification of model transformations, the designer defines how to generate target models from source ones. The generation is based on relationships between the involved metamodels and it can be based on simple correspondences or it could require complex computations on the models. An ASM-based transformation program consists of a collection of multiple independent rules of the form

\[
\langle \text{Query} \rangle \implies \langle \text{Transformation} \rangle
\]

with Query declaratively defined as first-order logic predicates over finite universes containing model element representatives and Transformation procedurally expressed as parallel updates of the encoding algebra. Moreover, the transformation branch may contain further transformation rules of the same form. This form could induce inefficiencies from an execution point of view especially if the predicates for querying a given model are complex. However, we believe that first-order logic predicates are suitable to specify queries on models in a precise and concise way. Moreover, as said above, the approach aims at providing a formal ground supporting implementation independent references of model transformations. The transformation developers can check their implementations (written in a more efficient way and in a specific language like ATL, QVT, VIATRA2, etc.) against an accurate and executable high-level model of the transformation itself.

Model transformation rules are iteratively fired until they do not cause any further update depending on whether their queries have a non-empty result or not. Thus, the matching algorithm is implicitly defined by the queries which establish also their relative precedences. Being more precise, according to the approach depicted in Fig. 14, an ASM-based model transformation specification consists of one or more rules having the following form:

\[
\begin{align*}
\text{do forall } \text{IN_PATTERN} \\
\text{--Transformation} \\
\text{OUT_PATTERN} \\
\text{endo}
\end{align*}
\]

where a query on the source model encoding is performed to find all the matches of the input pattern (IN_PATTERN). A pattern is a specification of source type coming from the source metamodel and it can be decorated with

\(^1\) Relations are special cases of functions whose values can be true or false.
conditions that drive the searching of matches in the source models. As discussed above, in the proposed approach a query is expressed by means of first-order logic predicates and for each of the matched pattern, the encoding of the target model is modified by changing the population of universes and the point-wise function definitions, as procedurally specified by the OUT_PATTERN. This one could embed the specifications of further transformation rules which will be executed until the query of the outermost one fails or no more changes on the algebra occur.

In order to give a flavor of the ASM-based transformation approach, in the following we outline a transformation which targets models conforming to the WAE metamodel previously described, starting from navigation models that conform to the metamodel in Fig. 6. A rule which composes the overall transformation is reported in the following (NavigationNode2Pages) and it specifies how navigation nodes give place to target server and client pages related through link and build associations. For each navigation node a server page and a client one are generated (see lines 6–9 in the following listing) and the build association relating them are obtained by extending the universe Build as specified in the lines 10–12 of the rule. The overall transformation applied to the model in Fig. 7 generates the WAE model in Fig. 16.

Fig. 16. Algebraic encoding fragment of a sample WAE model.
During the transformation phase, the relationships between models that are created by the transformation executions can be stored to preserve mapping information in a permanent way. There are many different usages for such traces like code synchronization, incremental transformations, etc. Leveraging the “everything is a model” principle, trace information can be stored by means of models. Since there is no standard trace metamodel, trace links can be based on a metamodel like the one depicted in Fig. 17 inspired by [45,46]. A TraceModel is a composition of several TraceLink elements that is traceability links that record the transformation rule and the source elements that were involved in creating target ones. The lines 13–25 of the transformation rule above takes into account such a metamodel for storing the trace link between the source NavigationNode and the corresponding generated elements (that are ServerPage, ClientPage and Build instances).

The described approach can be used to specify transformations which consist of complex rules based on intricate navigation and computations on the source models like for example the calculation of transitive closures [37]. Moreover, in order to increase the reuse of the transformations, the approach is capable of specifying generic transformations inspired by the work of Varró and Pataricza in [47] where data types, including model element types, are parameters of transformations.

For more information about the described ASMs-based transformation approach, the interested reader can refer to [37]. In fact, the focus of this paper is on the use of the weaving operation in the Web domain and the details about the transformation specification technique are beyond the scope of this work.

![Fig. 17. Sample trace metamodel.](image-url)
6. Target model generations

As already mentioned, the Web application concerns that are described by different models can be connected by means of explicit weaving models. Then ASMs-based transformations, defined in advance once for all, can be used to weave together the different concerns in order to generate target specifications comprising all the aspects of the system.

The rest of the paper describes with more detail the transformation phase for generating Webile and WebML models of the sample application whose concerns have been separately described above and explicitly connected through the given weaving models. The transformation rules start from an algebra whose signature includes the universes and functions induced by the involved metamodels that are the Data, Navigation, Composition and Weaving. The application of ASM rules generates a target algebra whose signature is induced by the target metamodels, which are Webile profile and WebML in our example. In the prototypical implementation of the proposed approach, the signatures are specified in XASM and they are automatically obtained from the KM3 specification of the metamodels. For example, with respect to the canonical encoding described in Section 5.2, the Data metamodel specified in the Listing 1 gives place to the following XASM specification

```plaintext
1 universe DATA_DataModel
2 function entities(a : DATA_DataModel, b : DATA_Entity) -> Bool
3 function relationships(a : DATA_DataModel, b : DATA_Relationship) -> Bool
4
5 universe DATA_Entity
6 function name(a : DATA_Entity) -> String
7 function attributes(a : DATA_Entity, b : DATA_Attribute) -> Bool
8
9 universe DATA_Attribute
10 function name(a : DATA_Attribute) -> String
11 function contentType(a : DATA_Attribute) -> String
12
13 universe DATA_Relationship
14 function name(a : DATA_Relationship) -> String
15 function source(a : DATA_Relationship) -> DATA_Role
16 function target(a : DATA_Relationship) -> DATA_Role
17
18 universe DATA_Role
19 function minCard(a : DATA_Role) -> String
20 function maxCard(a : DATA_Role) -> String
21 function entity(a : DATA_Role) -> DATA_Entity
```

For each class in the KM3 specification a corresponding sort is given by means of the keyword universe. The name of the sort is obtained from the name of the KM3 class prefixed with the name of the metamodel being encoded. The attributes and references in the KM3 specification induce corresponding functions.

6.1. Generating weble specifications

Before defining the transformations, a brief introduction to few Weble concepts is given through the model in Fig. 18; it is the outcome of the weaving operation that will be presented in the rest of the section. Such a model presents commonalities with the concern models defined in Section 3 since it merges them opportunely. Data are modeled in an Entity/Relationship fashion using the ≪DataEntity≫ and ≪DataRelation≫ stereotypes. The application functionalities lie on a conceptual structure consisting of departments (Department) which have several professors (Professor) and each of them has a number of publications (Publication). Pages are denoted by means of ≪StructuredContent≫ classes whose content is specified by means of ≪DataSource≫ stereotyped associations which allow one to define which data have to be retrieved from the conceptual structure and how.

In the figure, ProfessorsList contains the index of the professors who belong to the selected department in the page Department; the page ProfessorHome contains information about the selected professor and all her/his publications. This is described by annotating the corresponding data source associations. In fact, the tag Bound of a DataSource stereotype states whether the data retrieval has to consider the context of the involved structured content, in other words it declares that the data have to be filtered. Moreover, different data source associations targeting the same structured content and denoted by the same tagged value Label define a join operation. On the contrary, in ProfessorHome two different query operations are defined, because the labels on the associations
with Professor and Publication are different. Hyperlinks are modeled by means of the \texttt{\langle\texttt{CLink}\rangle} and \texttt{\langle\texttt{NCLink}\rangle} stereotyped associations which denote contextual and non-contextual links, respectively. The main difference between them is based on the fact that the former propagates parameters from the source to the target structured content, as in the case shown in the figure where the unique identifier of a selected professor is propagated to her/his home page. A more detailed discussion about Webile can be found in [7,48] while a fragment of its graphical specification is given in Fig. 19.

The transformation process is logically decomposed into four main phases, each devoted to the generation of specific fragments of the target model. In particular:

- The first phase generates Webile data entities and relations with respect to the source Data model, giving place to the data structure description of the application being developed;
- The second phase is devoted to the generation of target structured contents (that is pages) according to the nodes defined in the source Navigation model;
- Then the transformation produces the Webile data source elements establishing relationships between previously generated target data entities and structured contents. In this phase the source Data-Composition weaving specification plays a key role as explained in the following;
- Finally, navigation links between target pages are generated. During this phase all the five source models are taken into account in order to distinguish target Webile contextual and non-contextual links.

By going into more detail, in each of the previous specified steps, the first phase generates the algebraic representatives of the Webile data structure description the application is based on. This phase is performed by means of the following ASMs specification where for each \texttt{Data Entity} in the source Data model, a corresponding Webile data entity is generated (see lines 1–2 in the following ASM specification fragment). In this phase the trace information is heavily exploited. For readability reasons the auxiliary function \texttt{transformed} is used as a shortcut for the trace link management briefly described in Section 5.3. For example in the following rule the function

\[
\text{transformed} : \texttt{DATA Entity} \rightarrow \texttt{WEBILE DataEntity}
\]

is used for maintaining trace information that will be useful in the overall transformation rule. However, in the complete transformation specification available for download at [13], the trace links are managed according to the trace metamodel depicted in Fig. 17.
The derivation of StructuredContent stereotyped classes is performed dependently on the source Navigation model. For each navigation node a corresponding structured content element is generated (lines 4–7 below) and the name of the new element is the same as that of the page (see Fig. 13) which is woven with the considered navigation node (line 2).

DataSource elements are generated by the following code fragment with respect to the Data-Composition weaving specification. In particular, each weaving link between the Data and the Composition models gives place to a DataSource stereotyped association (line 8) in the target model. The transformed ones of the woven data element will be the data counterpart of the created DataSource association (lines 9–11). The determination of the StructuredContent element involved in this association is performed by considering the content which is woven in the source Data-Composition model. This content is used to find out the corresponding navigation element by traversing the Composition-Navigation weaving model (lines 2–6). Then the trace information stored in the function transformed (line 10) is used to discover the StructuredContent that has to be involved in the DataSource stereotyped association being generated.

The derivation of the CLink and NCLink stereotyped associations is more complex as a navigation through the five source models is necessary to establish whether a Link specified in the Navigation model has to propagate data. This information is evaluated by means of queries over the involved elements. In particular, the navigation links given in the source Navigation model in Fig. 7 states the navigation map of the application. As previously said, a non-contextual link is a simple connection between pages and does not affect the context of the target one, i.e. it does not propagate any information to the destination page. Consequently, a NCLink stereotyped association is created by the
following rules in two cases: whenever the target of a navigation link is not connected to data entities according to the weaving models (line 2–10), and when the contents of the corresponding pages are not related (line 31–34) through a data relationship path. Otherwise, for each couple of contents that belong to linked navigation nodes and that are woven with data entities related by a relationship path, a CLink stereotyped association is created as specified in the lines 13–29.

```plaintext
do forall l in NAVIGATION_Link
  if (not exists c in COMPOSITION_Content, p in COMPOSITION_Page,
    d in DATA_Entity, w1 in WEAVING_CompositionNavigationWLink,
    w2 in WEAVING_DataCompositionWLink : ownerPage(c)=p and
    isWoven(p,target(l),w1) and isWoven(d,c,w2))
    then
      extend WEBILE_NCLink with x
      source(x)=transformed(source(l))
      target(x)=transformed(target(l))
    endextend
  else
    do forall c1 in COMPOSITION_Content
      if (exists p in COMPOSITION_Page,
        w1 in WEAVING_CompositionNavigationWLink : ownerPage(c1)=p and
        isWoven(p,target(l)))
        then
          do forall c2 in COMPOSITION_Content
            choose w1 in WEAVING_CompositionNavigationWLink,
            p in COMPOSITION_Page, w2 in WEAVING_DataCompositionWLink,
            d in DATA_Entity : ownerPage(c2)=p ) and
            isWoven(p,target(l),w1) and isWoven(d,c2,w2)
              if (related(c1,c2) and (restricted(w2)) )
                then
                  if (type(c1)="index")
                    then
                      extend WEBILE_CLink with cl
                      source(cl)=transformed(source(l))
                      target(cl)=transformed(target(l))
                      ...
                    endextend
                  endif
                else
                  extend WEBILE_NCLink with ncl
                  source(ncl)=transformed(source(l))
                  target(ncl)=transformed(target(l))
                endextend
              endif
        endif
      endif
    enddo
```

Different auxiliary submachines are used in the above transformation rules, as isWoven(p, n, w) that returns true if the page p is woven with the navigation node n by means of the weaving link w described in the Composition-Navigation weaving model. Another submachine, called related(c1, c2), returns true if there exists a relationship path amongst the data entities to whom the contents c1 and c2 are woven in the Data-Composition weaving model. These submachines do not perform any change in the algebras and are used to collect information by navigating the models as for instance to compute the transitive closure of a relation. The interested reader can observe and execute the complete implementation of the described rules available for download at [13].

6.2. Generating WebML specifications

WebML is a modeling language that allows the conceptual description of Web applications under two conceptual dimensions: a data model specifies the schema of resources according to ER principles; a hypertext model describes how resources are assembled into information and pages, and how such units and pages interconnect to constitute a hypertext [6]. The WebML specification of the running example can be seen in Fig. 20. In particular, Fig. 20(a) specifies the data organization in terms of the relevant entities and relationships. Concerning the hypertext description, the language provides the designer with a number of different content units that can be composed into pages. Content units can be related by means of links that express Web site navigation as well as information transfers from one unit to another. In the hypertext model in Fig. 20(b) four pages are specified: the DepartmentList page contains the Departments index unit which will publish all the instances of the Department data entity. This kind of unit enables the selection of one of the shown instances and the outgoing link will bring the identifier of the selected instance to the target content unit. The index unit Profs in the ProfessorList page will publish instances of
the data entity Professor selected by the incoming identifier and filtered with respect to the relation between the Department and Professor data entities. Links can be also expressed between units belonging to the same page like in ProfessorHome where the data unit ProfInfo is linked with the index unit Pubs. In this case, once the ProfessorHome page is reached from ProfessorList, the information about the selected professor is shown and the index of publications is automatically updated with the data coming from the data entity Publication according to the Professor-Publication relation.

The rest of the section describes the ASMs transformation rules able to generate the models shown in Fig. 20 (and conforming to the metamodel in Fig. 21) according to the weaving specification given in Section 4. There is no official metamodel of WebML even if a number of research groups have been working on it [30,49]. The one in the figure is a subset of the available metamodels and contains only the concepts that will be considered in the rest of the section.

The transformation process is decomposed into three phases as explained below:

- a first phase generates the WebML data model the specified application is based on (like the one in Fig. 20(a));
- the second phase produces the Web pages that will be connected by means of the following step;
the links connecting the units belonging to the same page and those amongst distinct pages are created giving place to an hypertext like the one in Fig. 20(b).

Concerning the first phase of the transformation, for each Data_Entity in the source data model, a corresponding WebML data entity (lines 1–4 of the following ASMs fragment) is obtained. Furthermore, for each DATA_Relationship two corresponding WebML relations have to be generated, one for each direction (lines 8–16).

```
1   do forall de in DATA_Entity
2     extend WebML_Entity with wmlde
3         name(wmlde):=name(de)
4         ...
5   enddo;
6   endextend
7
8   do forall dr in DATA_Relationship
9     extend WebML_Relationship with wmlr1
10        name(wmlr1):=name(entity(source(dr)))+"_2_"+name(entity(target(dr)));
11        ...
12     endextend
13
14     extend WebML_Relationship with wmlr2
15        name(wmlr2):=name(entity(target(dr)))+"_2_"+name(entity(source(dr)));
16        ...
17     endextend
18   enddo;
```

The hypertext generation needs to visit all the source models as specified in the following ASMs rules. In particular, for each navigation node a corresponding target page is created (lines 1–2). If the type of the content expressed in the source Composition model is data, a DataUnit is defined (lines 7–12) otherwise an IndexUnit (line 15–24) will be put in the page being generated. The information that will be published in each content unit has to be specified by referring to data or relationships belonging to the conceptual structure. For example, according to the model in Fig. 20 the data published in the ProfInfo page are retrieved from the Professor data entity, whereas the Pubs index unit will contain data coming from the Publication data unit selected by means of the Professor_Publication relationship. This generation is performed by exploiting the submachine calculateSelectorRelationship(cu) (lines 7–21) devoted to calculate the data relationship which has to be used to select the data of the content unit cu.

```
1   do forall nn in NAVIGATION_Node
2       choose cp in COMPOSITION_Page, cnl in WEAVING_CompositionNavigationWLink:
3          isWoven(cp, nn, cnl)
4         extend WebML_Page with wmlp
5         ...
6   enddo;
```

```
7   do forall cc in COMPOSITION_Content
8       if (ownerPage(cc)=cp) then
9           if (type(cc)="data") then
10                 extend WebML_DataUnit with du
11                    name(du):=name(cc)
12                    ...
13           endif
14       endif
15
16     if (type(cc)="index") then
17         extend WebML_IndexUnit with iu
18            name(iu):=name(cc)
19         extend WebML_SelectorCondition with wsc
20            selector(iu):=wsc
21            relationship(wsc):=calculateSelectorRelationship(cc)
22            ...
23     endif
24   endchoose
25   enddo;
```

Finally, the links connecting the units belonging to the same page and those amongst distinct pages are created as follows
More precisely, a contextual link between an index and a data unit (belonging to different pages) is obtained if they are related (line 11) and belong to pages that are connected according to the source Composition-Navigation weaving and Navigation models respectively (lines 2–9). Otherwise a non-contextual link between the involved pages is generated (lines 17–20).

7. Discussion

Starting from the description of distinguished Web application concerns, weaving models can be used to generate several Web application specifications. The different models are easily kept separated, enabling focused changes to small portions of the specification, whereas when exploiting name conventions even narrowed modifications could require a wide inspection of the description in order to restore previous links. Furthermore, explicit relationships between concerns by means of models gives the possibility to take advantages from current model-driven methods and technologies. For instance, it is possible to (re)use weaving models or the concern specifications in a separate manner.

The proposed approach induces a more abstract layer in which it is possible to give a general platform independent model (PIM) of the system, which in turn will be mapped to one of the possible target PIMs. This process can be seen as a general trend related to the model-driven engineering approaches; people recognized that the use of a large monolithic model tends to be uncomfortable because of the intrinsic difficulties of the modeling task [10]. In fact, on the one hand all deep particulars need to be specified, which is less easy using a general-purpose language than a domain-specific one; on the other hand, the transformation definitions can be simplified, each being devoted to build narrowed portions of the target [50].

As already said, now there are several attempts to define a standard set of models able to completely describe Web application concerns; this work could support this evolutionary process mainly by proposing a technique to keep interconnected the various representations. This paper discussed the application of the weaving operation between static description of systems. However, a common metamodel for Web applications need to include also views on behaviour and deployment in order to describe complex Web-based applications. For example, in the metamodel presented in [8], behavioural elements are provided in terms of the Task and Adaptation subpackages that contain modeling elements for the workflow and personalization aspects, respectively. The weaving operation proposed in the paper can be used to specify relationships between static and behavioural models. In fact, as explained in Sections 3 and 4, the basic concepts behind the weaving operation are the specification of a “syntax” (by means of specific weaving metamodels) and a corresponding “semantics” through model transformations. If the ends of the weaving links are structural and behavioural concepts, the designer of the weaving specifies also the effects of such links. For example, different UML state machines can be used to specify the behaviour of operations used in the navigation model (e.g. a login operation or a checkout one). In this case, weaving operators can be specified to relate
navigation model elements with the behavioural descriptions. The corresponding weaving semantics may consist of the generation of a target UML state machine that integrates the separate concerns including the business logic which might be then formally validated for consistency and behavioural properties. This is just an example inspired by the work proposed in [51] that can be implemented with the approach proposed in this paper.

The main effort required for using the proposed approach is related to the development of suitable transformations able to support each different modeling language as output. However, this task has to be performed only once during the development of the weaving operators and, as just reasoned, it is simplified thanks to the specificity of the input models. Another criticism could be the proliferation of models, that is data, composition, navigation, data-composition and composition-navigation weaving descriptions, which could affect the usability in case of large scale projects. By the way, it has to be noted that usually a data model – even if not conforming to the same metamodel of the one used in this work – has to be designed (at least for validation and maintenance reasons). Moreover, the navigation model can be derived from requirement documents as storyboards, for example.

Last but not the least, weaving model specification can be supported by heuristics. In particular, schema and ontology approaches that aim at discovering semantic relationships between elements of different schemas or ontologies [35] could be used for deriving simple weaving models. However, for complex situations, the opportunities for heuristic applications are reduced especially when the relationships between model elements require intricate queries and computations over the models. This is the case discussed in this work that proposes the use of ASMs to formally specify the transformations capable of generating target models starting from source ones related by means of specific weaving specifications.

To summarize, despite the number of used models in the proposed approach we believe that the automation opportunity is wide, improving the usability of the technique in non-trivial situations too.

8. Related work

Concerning the proposals related to Web applications, the work in [52] proposes a development process based on architectural-centric transformations from design to implementation. In particular, models are integrated by means of transformations which adopt an implicit weaving in order to amalgamate together architectural and functional models. The main difference with respect to the work presented in this paper lies on the aspects they intend to weave together and the lack of an explicit specification of the correspondences among the models. The work also outlines automated transformations to generate platform-specific models for the J2EE, .NET or CORBA platforms similar to the approach we defined in [48].

Starting from the set of shared concerns in the development of web applications, we have given a set of weaving links to keep them interconnected in an explicit way; further, we have anchored to them a transformational semantics. With respect to the work in [23,24], the semantic units are the various typologies of weaving links, while the resulting semantics is implicitly given by means of the generated woven model, which conforms to the chosen DSL (e.g. Webile or WebML). In this way, once the mapping toward a particular language has been specified for all the semantic units, the effort left to the designer will be the description of the links between concern models.

ASMs have been already used as semantic basis for the specification of model transformations by other works like VIATRA2 [26]. It is an Eclipse-based general-purpose model transformation engineering framework intended to support the entire life-cycle for the specification, design, execution, validation and maintenance of transformations within and between various modeling languages and domains [36]. Its rule specification language is a unidirectional transformation language based on the combination of graph transformations [53] and ASMs into a single paradigm. Comparing the VIATRA2 with the transformation specification approach presented in Section 5, the former uses graph transformation rules for the specification of elementary model manipulation which are assembled into complex ones by abstract state machine rules. For the purpose of this work, which focuses mainly on the use of the weaving operation in the Web domain, the authors propose an approach completely based on ASMs in order to have a unique formalism for specifying the semantics of proposed weaving models. The idea is to provide the transformation developers with an implementation independent reference that can be implemented more efficiently in specific languages like ATL, QVT, VIATRA2, etc. against an accurate and executable high-level model of the transformation itself.

The work in [54] proposes a metamodel independent representation for understanding the semantics of concepts and features in domain modeling. Relationships between concept models, eventually pertaining to different domains, are represented by means of a mathematical formalism called Sigma. In particular, feature models describe the
semantics of the links which keep interconnected the concepts, building a model web. Compared against our approach it is possible to find some similarities; in fact, the semantics of our weaving links is based on a mathematical formalism and the separation of concerns to avoid monolithic models is one of the aims of this work. However, the weaving links presented in this proposal have not only a representation purpose, but are also used to derive output models from which the application implementation could be generated. The separation of concerns and the related issues are fundamental in the Aspect-Oriented environment; similar to our approach, orthogonal (or crosscutting) concepts are composed by means of weaving engines, which exploit join points and pointcuts to merge down the portions of the system and obtain a single running application [55]. In general, the aspect-oriented weaving technology is based on the assumption that the woven models and the resulting one conform to the same metamodel; consequently the merging specifications do not need complex operators. On the contrary, model weaving makes no assumptions about sources and targets, as shown by the examples in this paper. Hence, in the simplest cases it uses links to relate elements of different models while in complex ones it is possible to provide the connections with explicit and formal semantics.

This work can be related to a number of techniques dealing with model composition [34,56–59] even if it is far from presenting yet another technique for such a purpose. The work is more general aimed at proposing an approach for keeping models connected, taking into account the “everything is a model” principle. Furthermore, transformations could animate the model correspondences with respect to the semantics of the used links. Eventually such correspondences could give place to the composition of the involved models by exploiting one of the available algorithms [60]. The concept of weaving appears in numerous approaches for model management with the objective of handling fine-grained relationships between elements of distinct metamodels, by establishing links among them. Typical applications of model weaving are database metadata integration and evolution as in [22] which proposes Rondo, a generic metamodel management algebra which uses algebraic operators to manage mappings and models. The main difference between such an approach and ours lies in the way algebras are manipulated: Rondo defines a number of algebraic operators, such as difference and merge, while the techniques presented here have a more general-purpose flavor since ASMs constitute a formalism for defining algebraic (non-homomorphic) transformations, i.e. macro operations over the algebras.

9. Conclusions and future work

To deal with the complexity of current data-intensive Web applications, specific languages and techniques are often required. Over the last years a number of methodologies have been introduced for this purpose even if they share part of the proposed notations. Such constructs could give place to a common metamodel [8] which comprises at least elements devoted to the specification of the conceptual, navigation and presentation perspectives. The way by which the different views are kept connected is less formal as mainly based on implicit name conventions or on dedicated tool supports. The lack of abstraction in the separation between the concerns and their connections compromises several quality factors including reuse and maintenance of the source descriptions. Moreover, the possibility of using explicit relationships between models for tool chaining [9] can be also precluded.

This paper discussed a formal approach to define how the distinct concerns of a Web application can be connected better by means of weaving models. However, the approach is general enough to be applied not only for the Web domain. The main idea consists of the specification of weaving operators to establish relationships among the models that describe the various perspectives of the application being developed. The execution of the operators is based on a semantics defined in terms of model transformations formally specified by means of ASMs. We believe that the benefits of the proposed approach could be observed better by taking the advantage of a proper tool support. In this work, models are described by means of GMF-based graphical editors and the transformation rules are defined by means of executable specifications compiled with the XASM compiler [15]. Current efforts are devoted to the development of an integrated general environment supporting ASM-based transformations, weaving operations and composition of transformation programs. Furthermore, an interesting direction is the investigation on how the proposed approach can scale up to establish relationships between coarse-grained metamodel fragments. This might permit one to inherit the advantages discussed in Section 2 also in a global model management setting. In particular, the weaving operation could be generalized to support the concept of megamodel intended as a model of which at least some elements represent and/or refer to models or metamodels [61].
Acknowledgements

We would like to thank the anonymous reviewers for the useful and careful comments that permitted the authors to improve the previous version of the paper. Moreover we thank Alfonso Pierantonio for the long and interesting discussions on this work.

References