Model Transformations in the Development of Data–Intensive Web Applications

Davide Di Ruscio and Alfonso Pierantonio

Dipartimento di Informatica,
Università degli Studi di L'Aquila,
67100 L'Aquila, Italy
{diruscio, alfonso}@di.univaq.it

Abstract. Over the last few years, Web-based systems became commonplace. Despite the complexity and the economic significance of such applications, current practice does not always apply robust and well-understood principles. Model driven architecture (MDA) separates the application logic from the underlying platform technology and represents them with precise semantic models. Web application development therefore has potentially the most to gain from adopting such techniques that can offer a greater return on development time and quality factors than traditional approaches. In particular, the paper presents model-driven transformations between platform-independent (conceptual descriptions of Web applications) and platform-specific (Model-View-Controller conformant) models. The design of such transformations is documented (and possibly animated) through mathematically rigorous specifications given by means of Abstract State Machines.

1 Introduction

Over the last few years, Web-based systems became commonplace and underwent frequent modifications due to technological and commercial urges. Web sites rapidly evolved from simple collections of static pages to data-intensive applications which rely on dynamic contents usually stored in databases enabling a much wider range of interaction.

Despite the complexity and the economic significance of such applications, current practice does not always apply robust and well-understood principles. Model driven architecture [15] (MDA) separates the application logic from the underlying platform technology and represents them with precise semantic models. Web application development therefore has potentially the most to gain from adopting such techniques that can offer a greater return on development time and quality factors than traditional approaches.

In this paper, we describe a systematic approach to model-driven development of data-intensive Web applications meant as hybrid between hypermedia and information systems [13]. Starting from a suitable UML profile, called

O. Pastor and J. Falcão e Cunha (Eds.): CAiSE 2005, LNCS 3520, pp. 475–490, 2005.

Webile [24], conceptual descriptions of these systems are given as *platform-independent models* (PIMs), i.e. abstract descriptions that do not refer to the technologies they assume to exist. The process of transforming a PIM to obtain concrete implementations on the target architecture described by *platform specific models* (PSMs) is the ultimate consequence of shifting the focus of software development from coding to modeling. Different PSMs can be generated from a Webile model in order to describe different aspects of J2EE Web applications designed according to the Model-View-Controller [14] architectural pattern.

Model transformation presents intrinsic difficulties. It requires *"specialized support in several aspects in order to realize the full potential, for both the end-user and transformation developer"* [25]. The ability to simulate arbitrary algorithms on their natural levels of abstraction, without implementing them, makes Abstract State Machines [7] (ASMs) appropriate for high-level system design and analysis [5] and a candidate for specifying model transformation as well. Generating models in a formal setting can facilitate information traceability, reuse and evolution of software systems, but also represents a basis to reason about the intuitions encoded into unambiguous transformation descriptions. Furthermore, the ASM execution environment [2] represents an open framework with built-in support to syntax–tree manipulation (see [3]) useful for both tool integration and automatic transformation.

The paper is organized as follows. The next section illustrates an extended version of the Webile profile, which is used for the description of PIMs. Section 3 presents the founding elements for modeling J2EE Web applications designed according to the MVC architectural pattern. After introducing some basics about the ASMs, next section presents the ASM rules for transforming Webile models. Sect. 6 relates the work presented in this paper with other approaches. Finally, the last section draws some conclusions.

2 Webile

Webile [24] is a UML profile for describing in a uniform and conceptual way the proper aspects of data-intensive Web applications without referring to platform-specific assets. Leveraging the recurrency of certain application patterns which typically compose Web applications permits to raise the level of abstraction adopting a *model-centric* development whose main artifacts are models. These models are supposed to span the entire life cycle of a software system and ease the software production and maintenance tasks.

Descriptions encompass several concerns by capturing data, pages and navigation into extended class diagrams. In particular, data are given similarly to E/R models exploiting stereotyped classes and associations to model entities and relations, respectively. The profile prescribes the ≪DataEntity≫, ≪DataRelation≫, ≪DataStrongRelation≫ and ≪DataAttribute≫ stereotypes for modeling data. For instance, in Fig. 1 the elements contained in the dotted area, represent a

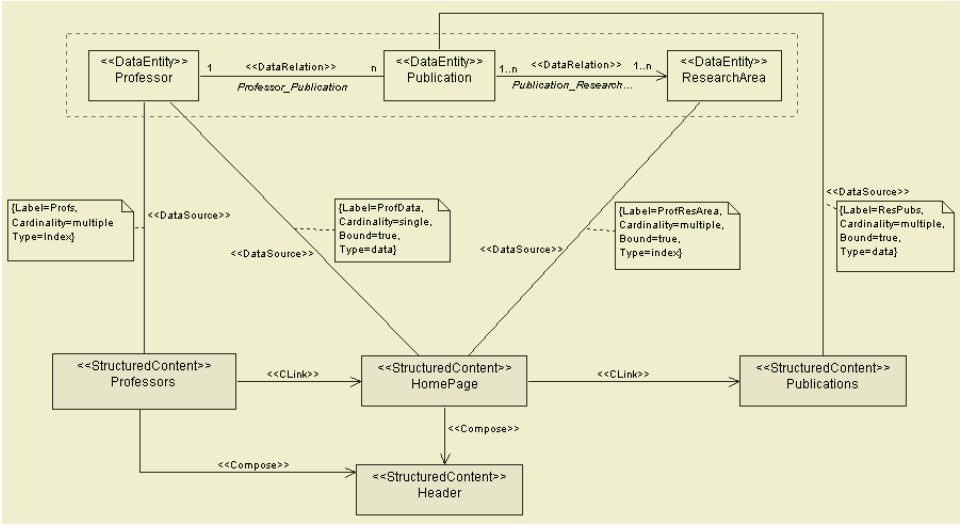

Fig. 1. A fragment of an academic site

simplified[1] conceptual data model of an academic site fragment, where professors (Professor) can have different publications (Publication), each belonging to one or more research areas (ResearchArea).

Pages and their fragments are denoted by means of ≪StructuredContent≫ stereotyped classes that are eventually associated with data entities providing contents by means of ≪DataSource≫ stereotyped associations. These associations are qualified with a collection of tagged values, amongst them Cardinality describes the cardinality of the items to be included in the content, i.e. whether the content consists of a single item or a list of them. In the figure, the Professors structured content contains the list of all professors in the database, which are retrieved through the associated entity Professor, in contrast with HomePage which contains information about one professor, respectively, because of the different specified cardinalities. Relevant aspects of the data source association affect the way the data are retrieved to form structured contents. In fact, different data source associations converging on the same structured content and denoted by the same tagged value Label define the same query operation (see Sect. 5). On the contrary, in HomePage two different query operations are defined, because the labels on the associations with Professor and ResearchArea are different.

Hyperlinks are modeled by means of the ≪CLink≫ and ≪NCLink≫ stereotyped associations which denote contextual and non-contextual links, respectively. The main difference among them lies in the fact that the formers propagate parameters from the source structured content to the target one. These

[1] For presentational purposes, we omitted attributes and other information which are not relevant at this stage of the discussion.

parameters are used when data source associations have the tagged value Bound set to *true* to filter the data retrieved from the corresponding entities. For instance, in Fig. 1 the contextual link going out from Professors allows the user to select a single professor in order to access her/his personal profile in HomePage, which is collected by means of the ≪DataSource≫ stereotyped associations with the entities Professor and ResearchArea. Analogously, the contextual link outgoing from HomePage provides with the access to Publications of the selected research area. Non contextual links are much simpler since they connect structured contents which are not semantically correlated.

Support to modularity is also important to achieve pragmatic qualities. In fact, the structured contents do not describe only pages but also portion of them, as with Header which is shared by the Professors and HomePage structured contents via the correspondent ≪compose≫ stereotyped associations. Further stereotypes have been defined to cover additional aspects. In particular, the association ≪DataEntry≫ between a page and some data entities is used to declare data entry forms. Moreover, structured contents can be subject to authentication/authorization in order to secure certain contents accessible only to specific users and/or groups specified by the ≪User≫ and/or ≪Group≫ stereotyped classes. Due to space limitations, we did not report data entry forms and protected pages in the examples. Finally, a relevant case study has been produced by modeling a large institutional site in order to validate the expressiveness of the profile (see [12]).

The Webile profile was originally devised to generate code directly from models in an *one-step* fashion without any human intervention. The approach has shown immediately problems not limited to poor consistency and traceability between models and code, as the formers start to diverge from the latter as soon as changes are operated on the generated system. Thus, the approach has been considerably extended introducing proper model transformations able to map Webile models into model chains which, at different level of abstractions, are descriptions of the chosen implementation.

3 Describing PSMs

MVC is an architectural pattern which aims at minimizing the degree of coupling between elements to relate the user interface to underlying data models in an effective way. Increasingly, the MVC pattern is used in program development with object-oriented languages and in organizing the design of J2EE Web applications proposing a three-way factoring paradigm based on the following

- the model holds all data relevant to domain entity or process, and performs behavioral processing on that data;
- the view displays data contained in the model and maintains consistency in the presentation when the model changes; and
- the controller is the glue between view and model reacting to significant events in the view, which may result in manipulation of the model.

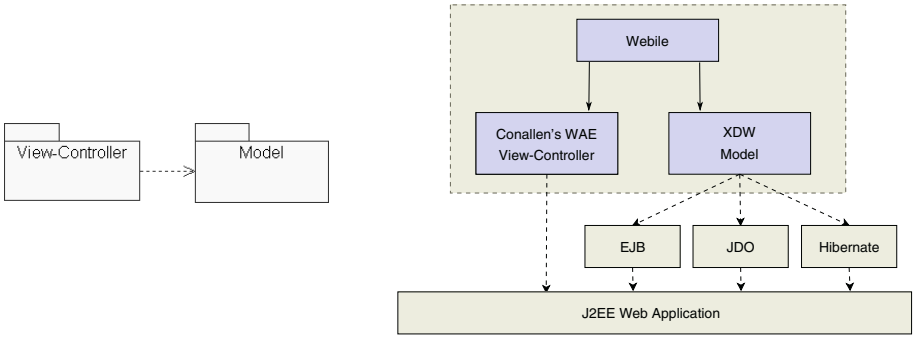

Fig. 2. Different Views of the MVC pattern

The description of PSMs referring to the J2EE platform may distinguish the model from the view and the controller. This separation of concerns is motivated by the abundance of persistence frameworks, such as EJB [11] and JDO [18] to mention a few, which suggests further refinements of the model into more specific PSMs retaining the view-controller design (see Fig. 2). According to the figure, a Webile specification is mapped into platform-specific descriptions of the view-controller and the model, respectively. This mapping is automatic and mathematically defined by executable ASM transition rules as described in Sect. 5. In the proposed approach, the View-Controller package (see Fig. 2) is given by means of Conallen's Web Applications Extension [10] (WAE) whereas the Model package is given by means of the data part of Webile opportunely extended to some abstraction for realizing given business tier patterns [1].

3.1 View-Controller: Conallen's WAE

The Web Application Extension (WAE) is an extension of UML for modeling Web applications proposed by J.Conallen. Web pages are modeled by giving both server-side and client-side aspects by means of ≪Server Page≫ and ≪Client Page≫ stereotyped classes, respectively. A server page can be associated with other server-side objects, i.e. database, middle-tier components and so on, although we are not going to model data aspects here. The ≪Client Page≫ stereotype represents a HTML page which is usually associated with other client or server pages. In the last case the ≪build≫ stereotyped association is used to state that a server page builds a client one. An hyperlink between pages is modeled by a ≪link≫ stereotyped association. If the hyperlink includes parameters, they are modelled as link attributes of the association. A directional relationship between one server page and another server or client page is modeled by the ≪forward≫ stereotyped association. This association represents the delegation of processing client's requests for a resource to another server-side page and it is a pivotal aspect proper of the view-controller metaphor.

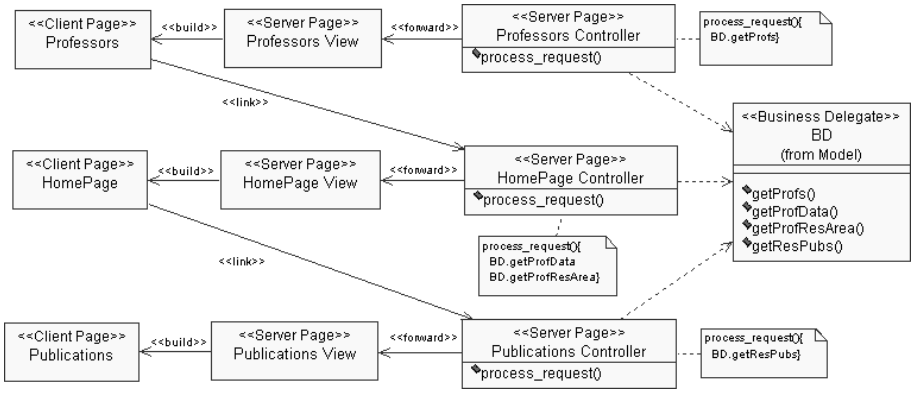

Fig. 3. Conallen's View-Controller description

In fact, referring to Fig. 3 and according to the adopted pattern, client requests are processed by the controller server pages which perform the data retrieval by invoking the proper operations on the business delegate object (as explained in the next section). Each controller declares exactly the operation which must be invoked according to the data source associations in the conceptual model, e.g. the server page class HomePage Controller depends on the methods getProfData() and getProfResArea() to retrieve the data. Once the data are available to the controller, the request is forwarded to the corresponding view server page. In particular, the figure illustrates how to implement the application logic of the system described in Fig. 1 by means of several views and controllers; each structured content is mapped to a pattern consisting of linked client page, view and controller server pages. Alternatively, the front controller pattern [1], i.e. a unique controller which serves as a centralized access point for requests and link, could have been adopted. It is a solution which is widely used by software developers, which encodes information about the navigation in the url requests, thus is less convenient to illustrate how the navigation in Webile is propagated during model transformation.

Finally, the idea of adopting Conallen's approach for specifying PSMs in not novel, since it mainly represents the implementation and is therefore suitable for PSMs rather than PIMs [21, 22].

3.2 Model: eXtended Data Webile

This section presents how to describe the Model component of the MVC pattern by means of an extension of the data part of Webile, called eXtended Data Webile (XDW). A better maintenance and flexibility in accessing business services requires specific abstraction layers as the ones realized by means of the business delegate and the transfer object design patterns [1]. In particular, the business delegate hides implementation details of the business service and encapsulates access and lookup mechanisms; whereas the transfer object serves to optimize data

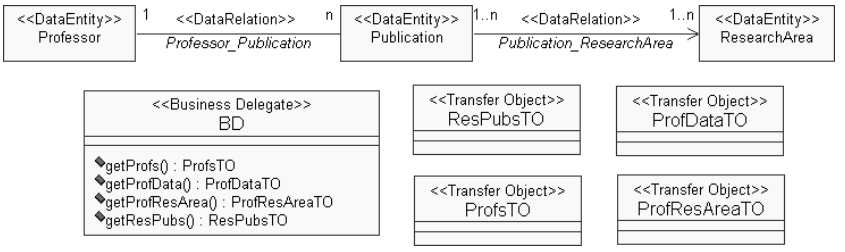

Fig. 4. XDW Model description

transfer across tiers. Instead of sending or receiving individual data elements, a transfer object contains all the data elements in a single structure required by the request or response. To summarize, a controller can access business services by performing requests to a business delegate which implements the services and returns the result as a transfer object. For instance, Fig. 4 depicts a diagram which describes by means of XDW the Model components of the application which has been modeled in Fig. 1. It comprehends only the data aspects of the original model and additionally introduces the business delegate and a transfer object for each different query operation defined within the business delegate. To understand how such elements are defined, let us consider the data source association in Fig. 1 labeled ProfData between HomePage and Professor, this association defines the query operation in the business delegate called getProfData() which returns a transfer object of type ProfDataTO. In order to keep a certain degree of abstraction, the query operations in the business delegate are specified by means of relational algebra expressions which are computed by ASMs rules presented and commented in Sect. 5.

4 Abstract State Machines

Due to space limitation, we only briefly introduce ASMs here insisting on few introductory aspects. For more information, the reader is referred to [6, 7]. ASMs bridge the gap between specification and computation by providing more versatile Turing-complete machines. The ability to simulate arbitrary algorithms on their natural levels of abstraction, without implementing them, makes ASMs appropriate for high-level system design and analysis (see [5]). Additionally, ASMs are executable and several compilers and tools are available both from academy and industry.

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. The set $\mathcal{U}$ referred to as the *superuniverse* in ASM terminology, always contains the distinct elements *true*, *false*, and *undef*. Apart from these, $\mathcal{U}$ can contain numbers, strings, and possibly anything, depending on the application domain.

Being slightly more formal, we define the *state* λ of a system as a mapping from a signature Σ (which is a collection of function symbols) to actual functions.

We write f_λ for denoting the function which interprets the symbol f in the state λ. 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 $a \in U$, $f(a) \in V$ and $f(a) = undef$ otherwise. 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 Σ. The semantics of such a rule is this: evaluate all the terms in the given state, and update the function corresponding to f at the value of the tuple resulting out of 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. Of course not all functions can be updated, for instance the basic arithmetic operations are typically not redefinable.

5 Model Transformations

The main motivation behind MDA is to shift the focus of software development from coding to solution modeling by assuming models as the primary artifacts of the development. Key concepts of MDA are model transformations intended as programs which mutates one model into another similarly to a compiler.

In the sequel, unidirectional stateless transformations are given to map Webile models into Conallen and XDW ones. Unidirectional transformations map the source metamodel into the target metamodel but not the converse. Although this may appear a limitation, in practical cases this is essentially unavoidable since a bidirectional transformation implies the adoption of declarative rule-based formalisms which pose severe questions about the termination of transformations. A persistent (in contrast with stateless) model transformation enables change propagation, in the sense that performing the transformation when the source model has changed does not always result in a newly creation model. In fact, persistence implies version policies towards the target model which in combination with information tracking allows not to rewrite completely the target model for different incarnations of the transformation. An interesting and detailed discussion on model transformation languages can be found in [25].

The transformations are defined as ASM rules which starting from an algebra encoding the source model, return an algebra encoding the target model. The signature of an algebra encoding a model is induced by the UML metamodel whose

elements define the sorts of the signature, for instance the class and association elements give place to the *Class* and *Association* sorts, i.e. the algebra has two universes containing distinguished representatives for all the classes and associations in the model. Stereotypes extending the model elements define subsets in the universes induced by the extended elements itself. This is nicely modeled since ASMs allow subsorting, for instance in the Webile profile the ≪DataEntity≫ and ≪DataSource≫ stereotypes induces the following subsorting relations

$$DataEntity < Class \quad \text{and} \quad DataSource < Association$$

Additionally, the metamodels induce also functions which provide with support to model navigation, e.g. the associations have source and target functions

$$source,\ target : Association \rightarrow Class$$

which return the source and the target class of the association. Methods are represented by the sort *Method* and the class they belong to is computed by the function

$$belong : Method \rightarrow Class$$

further functions defined over methods are *name* and *body* which return the name and the body of a method, respectively. Also tagged values are encoded by means of functions, for example the tagged value Cardinality of the ≪DataSource≫ stereotyped association defines

$$cardinality : DataSource \rightarrow \{single, multiple\}$$

Moreover, further functions and sorts are given by the basic data types and by those functions which are used in transition rules to accumulate information during the transformation. As an example, the algebraic encoding of the model in Fig. 1 is illustrated in Fig. 5.

In the next sections, the ASM rules for generating the PSMs for the Model and for the View and the Controller are presented, respectively, according to the Fig. 2.

5.1 Model Transformation: View-Controller

The transformation introduced here consists of a number of ASM rules, in particular for each structured content the rule *StructuredContent* extends the algebra encoding the source model with three new classes, two server pages modeling the view and the controller and a client page which is generated by the view server page. Furthermore, the rule introduces the following functions

$$controller, serverView : StructuredContent \rightarrow ServerPage$$
$$clientView : StructuredContent \rightarrow ClientPage$$

used to track the structured contents from which the client and server pages have been generated. The rule is

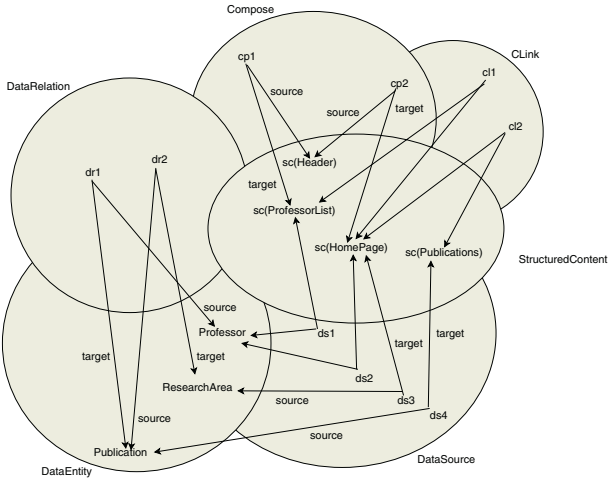

Fig. 5. A model encoded in an algebra

```
asm StructuredContent is
  do forall x in StructuredContent
    extend ServerPage with s1,s2 and ClientPage with c and Build with b
      and Forward with r and Use with u
      source(b) := s1, target(b) := c
      source(r) := s2, target(r) := s1
      source(u) := s2, target(u) := bd                                    (1)
      controller(x) := s2, serverView(x) := s1, clientView(x) := c
      extend Operation with op
        name(op) := "process_request"
        body(op) := Invocations(x)                                        (2)
        belong(op) := s2
      endextend
    endextend
  enddo
endasm
```

Line (1) in the above rule contains a reference to *bd*, the representative of the business delegate component which is incrementally assigned the query operations; line (2) contains the invocations to the *Invocations* sub-machine which computes and returns the list of method names which the controllers have to invoke in their body.

The *CLink* rule for each ≪CLink≫ stereotyped association in Webile extends the universe *Link* with a new element whose source and target are the linked *ClientPage* and *ServerPage*, respectively

```
asm CLink is
  do forall x in CLink
    extend Link with l
      source(l):=clientView(source(x)), target(l):=controller(target(x))
```

```
    endextend
  enddo
endasm
```

The rules described up to now are not very complex, they could even be considered declarative, since they make use only of the update rule (simpler than in attribute grammars, for instance, which requires some resolution). Algebraically, they can be given as a set of positive conditional equations which induce a (free) functorial transformation on the source algebras. Finally, the rules for handling the composition of structured contents and non-contextual links are missing, since their complexity is comparable to that of the rules above.

5.2 Model Transformation: Model

The most interesting rules are not just attributions as the ones above. It is crucial, to be able to collect information while navigating the model, as when computing the transitive closure of a relation for instance. The following rule *DataSource* has to generate the specification of the query operations in the business delegate as relational algebra expressions starting from the data sources in the Webile model. Depending on the tagged value Label of the ≪DataSource≫ associations, the way the contents are retrieved is defined giving place to different expressions. All the ≪DataSource≫ stereotyped associations related to a specific ≪StructuredContent≫ can be grouped according to their *Label* tagged value and associated to a *Label*-indexed query operation. The rule has to navigate the source model to understand which data entities are involved in the relational algebra expressions. The *DataSource* rule is defined as follows

```
asm DataSource is
  DefineAllContents
  do forall x in StructuredContent and l in Label : cont(x,l)!=undef
    extend Operation with op
      belong(op) := bd
      name(op) := "get"+name(l)
      choose t in TransfObject : name(t)=name(l)+"TO"
        type(op) :=t
      endchoose
      body(op) := Expr(x,l)
    endextend
  enddo
endasm
```

where *DefineAllContents* sub-machine creates lists of data sources according to the Label tagged value partitioning explained above. The rule is given below and makes use of *addListElement* which adds elements to a list

```
asm DefineAllContents is
  do forall x in StructuredContent
    do forall y in DataSource : target(y)=x
      do forall l in Label : label(y)=l
        addListElement(cont(x,l),y)
      enddo
```

```
    enddo
  enddo
endasm
```

The sub-machine *Expr* of *DataSource* generates the relational algebra expression whose evaluation supplies the content l for the structured content x.

```
asm Expr(x, l) is
  extend Body with y
    join(y) := unify(findPath(cont(x,l)))
    selectionKey(y) := findKey(cont(x,l))
    return y
  endextend
endasm
```

To better understand this rule, let us consider Fig. **??** where an abstract representation of a Webile model is presented. The structured content *SC* is fed by three data sources ds_1, ds_2 and ds_3 with the same *cont1* label. In order to obtain a relational algebra expression

$$\sigma_F(T_1 \bowtie_{c_1} T_2 \bowtie_{c_2} T_3 ... \bowtie_{c_{n-1}} T_n)$$

two macro steps have to be executed:

- the definition of joins between the right relations and,
- the definition of the selection formula F.

The former is obtained by means of the *unify* rule, the latter by means of the *findKey* one. Note that the definition of the expression is not trivial and, due to space limitation, we present the solution by outlining the description for the *findPath*, *unify* and *findKey* rules. Two data entities involved in the definition of a content by means of two ≪DataSource≫ associations, may give place to ambiguous scenarios. In fact, E_1 and E_3 in Fig. 6, are related by means of two different paths. This causes problem for the definition of the joins involving them. Webile deals with this

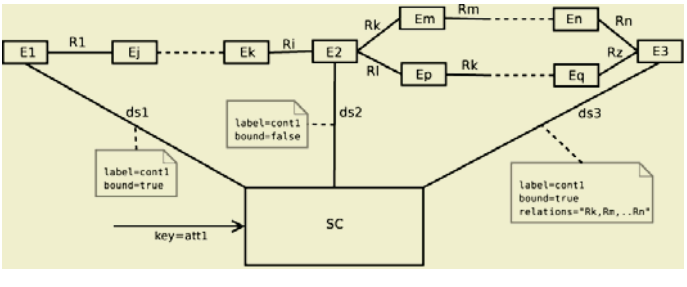

Fig. 6. An abstract representation of a Webile model

problem by means of the tagged value *Relations* of the ≪DataSource≫ stereotype. This is used by the *findPath* rule which, for each pair of entities involved in the content definition, finds the right path of relations connecting them.

For this rule, the set $Path$ is defined as $DataRelation^* \times Bool$ whose elements are terms *path(R,C)*, where the first parameter R is the list of relations defining the path in the source model, the second parameter C is a boolean denoting whether the conditions of the joins involving the entities in the relation chain are empty or equals to conjunctions of equations involving the corresponding keys. For instance, in Fig. 6, *findPath* returns the list containing the following elements: $\{path(R_k, R_m, ..., R_n, true), path(R_1, ..., R_i, false)\}$.

The *unify* rule evaluates the paths and defines the joins between the *logical* relations in the paths recursively, whereas *findKey* defines the selection formula for the final expression. Accordingly, if E_1 contained the attribute att_1 in Fig. 6, the formula F would be the equation $E_1.att1 = att1$. Otherwise, if E_2 contained the attribute att_1, then the key propagated by the contextual link has no effect and the selection formula is empty. The relational algebra expression defined with this process represents the body of a server-side operation part of the server page obtained by means of the transformation of the structured content *SC*.

The last rule handles the creation of the transfer objects, i.e. each query in the business delegate returns a different transfer object type which needs to be defined, as follows

```
asm CreateTransfObj is
  do forall l in Label
    extend TransfObject with t
      name(t) := name(l)+"TO"
      do forall d in DataSource : label(d)=l
        do forall a in DataAttribute : belong(a)=source(d)
          extend Attribute with a1
            name(a1) := name(a), type(a1) := type(a)
          endextend
        enddo
      enddo
    endextend
  enddo
endasm
```

6 Related Work

Model transformations are increasingly gaining attention in different areas of software design, development and integration. Such significance is also witnessed by the OMG's Queries/Views/Transformations RFP [16] issued to define a standard way of performing model transformations. The submitted proposals range from imperative unidirectional stateless transformations [23] to declarative persistent bidirectional ones [9]. The former does not allow any form of persistence, while the latter may easily lead to solutions which can take potentially unbounded time to execute [25]. Other approaches are considered hybrid, such

as ATL [4], since they wrap imperative bodies inside declarative shells to specify unidirectional transformations.

Focusing on the languages for describing Web applications, many topics may be related to the work proposed here, among them many are suggesting extensions to model web applications using UML. Apart from Conallen's work, it is worth mentioning the approaches proposed by Koch and Hennicker [17]. In particular, they structure the design of web applications into three different aspects: content, navigational structure and presentation (the data dimension is not considered). Since these aspects are related together, they propose to model each one using a different UML model and to relate them together using mapping rules. This separation (present also in other approaches [8, 20]) is currently missing in Webile. Differently from us, Koch and Hennicker use UML class and sequence diagrams to describe and model behavior.

Regarding model-based code generation for the Web, Kraus and Koch [19] show how the UML design models produced in [17] can be automatically mapped into XML documents. Araneus [20] and WebML [8] represent the more interesting model-based approaches for the Web, especially WebML which is supported by the WebRatio commercial tool.

7 Conclusions and Future Work

This paper describes a model-driven approach for the development of data-intensive Web applications. Starting from conceptual models that do not refer to any technological asset, formal model transformations are used to obtain several PSMs for different aspects of an MVC conformant J2EE application. Compared with techniques which allow *one-step* model-to-code generation, flexible and practical model transformations enhance traceability and consistency between models and code, since they tend to diverge as soon as changes are manually operated on the generated applications.

Model transformations are intrinsically difficult. According to our experience they should combine desirable features as formality and declarativeness with good pragmatic qualities. The transformations introduced above are applicable in a declarative way since models are queried by means of first-order predicates and subsequently manipulated in an operational fashion.

Model composition is also important, since generating models from the same source model requires to merge them on certain correspondences. Exploiting algebraic and categorical constructions (in the sense of category theory), which have been investigated since decades, we plan to define a theory of transformation composition to perform complex model weaving and transformation maintenance.

Acknowledgments

We thank Amleto Di Salle and Fabio Mancinelli for the lively and enlightening discussions. Also, we are grateful for the insightful comments we received from the anonymous reviewers.

References

1. D. Alur, J. Crupi, and D. Malks. *Core J2EE Patterns.* Sun Microsystems Press (Prentice Hall), 2nd edition, 2003.
2. M. Anlauff. XASM – An Extensible, Component-Based Abstract State Machines Language. In *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 69–90. Springer, 2000.
3. M. Anlauff, S. Chakraborty, P. Kutter, A. Pierantonio, and L. Thiele. Generating an action notation environment from Montages descriptions. *Int. J. Software Tools for Technology Transfer*, 3(4):431–455, 2001.
4. J. Bézivin, G. Dupé, F. Jouault, G. Pitette, and J.E. Rougui. First Experiments with the ATL model transformation language: Transforming XSLT into XQuery. In *2nd OOPSLA W. Generative Techniques in the context of MDA*, 2003.
5. E. Börger. Why Use Evolving Algebras for Hardware and Software Engineering? In *Procs. SOFSEM '95, 22nd Seminar on Current Trends in Theory and Practice of Informatics*, volume 1012 of *LNCS*, pages 236–271. Springer, 1995.
6. E. Börger. The Origins and the Development of the ASM Method for High Level System Design and Analysis. *J. Universal Computer Science*, 8(1):2–74, 2002.
7. E. Börger and R. Stärk. *Abstract State Machines - A Method for High-Level System Design and Analysis.* Springer, 2003.
8. S. Ceri, P. Fraternali, and A. Bongio. Web Modeling Language (WebML): a Modeling Language for Designing Web sites. *Computer Networks*, 33(1–6):137–157, 2000.
9. Compuware and Sun. XMOF queries, views and transformations on models using MOF, OCL and patterns, 2003. OMG Document ad/2003-08-07.
10. J. Conallen. Modeling Web Application Architectures with UML. *Comm. ACM*, 42(10):63–71, 1999.
11. Enterprise JavaBeans. http://java.sun.com/products/ejb/.
12. E.Romina. Modellazione concettuale di un portale mediante UML. Master's thesis, Università degli Studi dell'Aquila, 2003/04. In italian.
13. P. Fraternali. Tools and Approaches for Developing data-intensive Web Applications: A Survey. *ACM Computing Surveys*, 31(3):227–263, 1999.
14. S.T. Pope G.E. Krasner. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *J. Object-Oriented Programming*, 1(3):26–49, 1988.
15. Object Management Group. OMG/Model Driven Architecture - A Technical Perspective, 2001. OMG Document: ormsc/01-07-01.
16. Object Management Group. MOF 2.0 Query/Views/Transformations RFP, 2002. OMG document ad/02-04-10.
17. R. Hennicker and N. Koch. Systematic Design of Web Applications with UML. In *Unified Modeling Language: Systems Analysis, Design and Development Issues*, chapter 1, pages 1–20. Idea Publishing Group, 2001.
18. Java Data Objects. http://java.sun.com/products/jdo/.
19. A. Kraus and N. Koch. Generation of Web Applications from UML Models using an XML Publishing Framework. In *Procs. 6th World Conference on Integrated Design and Process Technology (IDPT)*, volume 1, 2002.
20. P. Merialdo, P. Atzeni, and G. Mecca. Design and Development of data-intensive Web sites: The Araneus approach. *ACM Trans. on Internet Technology*, 3(1):49–92, 2003.
21. P.-A. Muller, P. Studer, and J. Bezivin. Platform independent web application modeling. In *UML 2003*, volume 2863 of *LNCS*, pages 220–233. Springer, 2003.

22. A. Vallecillo N. Moreno. Using MDA for Designing and Implementing Web-based Applications. In *Int. Conf. on Web Engineering*, Munich, Germany, 2004. Tutorial.
23. OpenQVT. Response to the MOF 2.0 Queries / Views / Transformations RFP, 2003. OMG Document ad/2003-08-05.
24. D. Di Ruscio, H. Muccini, and A. Pierantonio. A Data Modeling Approach to Web Application Synthesis. *Int. J. Web Engineering and Technology*, 1(3):320–337, 2004.
25. L. Tratt. Model transformations and tool integration. *J. Software and Systems Modeling*, 2004. To appear.