Re-engineering Legacy Web Applications into Rich Internet Applications*

Roberto Rodríguez-Echeverría, José María Conejero, Marino Linaje, Juan Carlos Preciado, and Fernando Sánchez-Figueroa

Quercus Software Engineering Group
Universidad de Extremadura, 10003, Cáceres, Spain
{rre,chemacm,mlinaje,jcpreciado,fernando}@unex.es

Abstract. There is a current trend in the industry to migrate its traditional Web applications to Rich Internet Applications (RIAs). To face this migration, traditional Web methodologies are being extended with new RIA modeling primitives. However, this re-engineering process is being figured out in an ad-hoc manner by introducing directly these new features in the models, crosscutting the old functionality and compromising the readability, reusability and maintainability of the whole system. With the aim of performing this re-engineering process more systematic and less error prone we propose in this paper an approach based on separation of concerns applied to the specific case of WebML.

Keywords: Web Models Transformations, Patterns, Rich Internet Applications.

1 Introduction

More and more, traditional Web applications are being migrated to RIAs and, consequently, more and more, traditional Web methodologies are incorporating RIA modeling features [1][10][13][18]. Among these proposals, WebML deserves our attention due it being one of the most promising approaches because of its significant extensions to accomplish RIA features at different levels: client/side processing and storing [1], event handling [17] or presentation [11].

Despite these efforts, the real fact is that the industry is performing this migration in and ad-hoc manner, leading to two different problems: on the one hand, the original model becomes tangled with concerns for different purposes such as distribution or persistence which compromises the readability, understandability and, consequently, maintainability of the system. On the other hand, the adaptations performed related to RIA features are not reusable since they have to be applied again from the scratch in any new migration process. This is a significant drawback since there are quite a few adaptations that recurrently appear in many RIAs, e.g., synchronization patterns needed to work in a disconnected mode.

Precisely, this paper presents a proposal for the systematic re-engineering of Web applications modeled with WebML into RIAs following an Aspect-Oriented approach [4]

* This work has been supported by MEC under contract: TIN2008-02985.

B. Benatallah et al. (Eds.): ICWE 2010, LNCS 6189, pp. 189–203, 2010.

in the sense that the RIA features are modeled separately. The main contribution of the proposal is twofold: on the one hand, extending the WebML metamodel to include RIA features and, on the other hand, defining a new systematic model driven re-engineering process, based on model compositions driven by weaving models and pattern instantiations, to perform the weaving between the legacy model (WebML metamodel compliant) and the model describing the RIA related features (extended WebML metamodel compliant). As an additional contribution, several RIA synchronization patterns have been also identified and modeled separately, applying one of them to a running example. The RIA features contemplated are those related with data and business logic distribution and their associated issues, e.g. synchronization, event notifications, communications, etc. The separation approach followed is symmetric in the sense that it takes advantage of the entities and units already defined in WebML. A main objective of the proposal is to define a reusable framework automatically applicable to any re-engineering process within the domain.

The rest of the paper is as follows. Section 2 briefly introduces WebML extensions for RIA. In section 3 we present a motivating example to highlight the problems we want to solve with the proposed approach that is presented in Section 4. Section 5 applies the proposal to the motivating example, while section 6 identifies related works and presents the main conclusions.

2 WebML for Rich Internet Applications Capabilities in Brief

WebML is a language for the high-level description of a Web system consisting of data model, hypertext model, presentation model and personalization model. The application data are modeled using Entity-Relationship (E-R) or UML class diagrams. On top of the data model, WebML allows specifying the *business logic* and the content/containers composition by means of the hypertext model, whose key ingredients include *siteviews*, *areas*, *pages*, *content units*, *operation units*, and *links*. New entities have been included to treat new challenges like the XML_{OUT} and XML_{IN} ones, which have been used in [12] to marshall and unmarshall data, respectively.

Recently, WebML has been extended to cover RIA features in an approach called WebML for RIA [1]. Extensions proposed for the data model are characterized by two different dimensions: 1) the architectural tier, where client entities and relationships are marked with a "C" label while server ones are marked with a "S"; and 2) the persistence of the data, which is also indicated.

Extensions proposed for the hypertext model are those ones affecting the structural composition of RIAs typing WebML pages into server pages (marked with a "S") and client pages (marked with a "C"). Links are special cases since they could relate entities of client and server pages. In that sense, a link relating two client units is considered as client tier, whereas a link relating a client and a server unit is considered as inter-layer operation chains. An event model to support pulling/pushing RIA capabilities for WebML is also defined in [17]. This event model extends the original WebML data model and the set of units available in the hypertext model (adding the *sent event* and *receive event* ones). The full set of constrains to be accomplished by the model in order to be computable to generate the code are also

specified in [1]. Both WebML extensions (i.e., [17] and [1]) are used in our work to specify RIA capabilities plus [12] where XML_{IN} and XML_{OUT} units are introduced to cope with marshalling and unmarshalling data into/from a XML file.

3 Motivating Example

Let us consider a simple e-shop for selling tickets for concerts where a pre-booking mechanism is mandatory in order to avoid situations such as two different users booking the same seat for the same show at the same time.

First, the example is solved using WebML. Then, some RIA features are included and modeled with WebML for RIA. The aim is just illustrating how the resulting model becomes tangled, leading to the two problems identified in section 1.

3.1 The Traditional Ticket e-Shop in WebML

Conceptual Data Model. (Fig. 1): *PreBookings* is specified to avoid booking conflicts (where the *endTime* attribute represents the deadline of a pre-booking). *CartItem* entity stores the different seats selected by a user in a volatile way (note the italic style in the name). Unlike *CartItem*, *PreBooking* is not volatile to ensure that the seats are not booked more than once in the database. The *Order* and *OrderItems* entities manage the information required to make persistent an order.

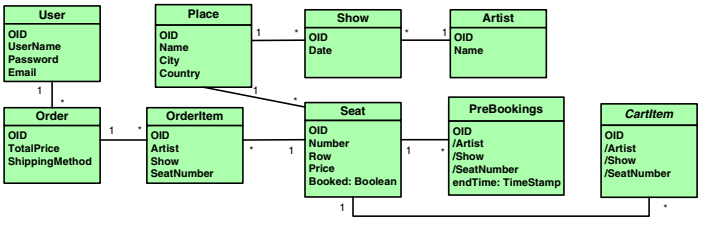

Fig. 1. Data model for the tickets reservation

Conceptual Hypertext Model. (Fig. 2)[1]: When a user accesses to the details of a show from the list of shows (*Events for artist* page), the pre-bookings and availability of seats for this show are displayed (*Booking* page) and they can be added to the cart. By using traditional Web techniques, the free seats are displayed when the user accesses the *booking* page, so this information is not later updated even when other users are performing bookings in that moment.

The operation chains depicted in the excerpt of the hypertext model are responsible for two different actions: i) the addition of a seat to the cart, including the management of the pre-booking (marked ***A*** with a dashed rectangle) and ii) the generation of an order from the user's cart and the removal of the cart items and the pre-bookings of that user (marked ***B***).

[1] In the example some units have been omitted for simplicity and space reasons.

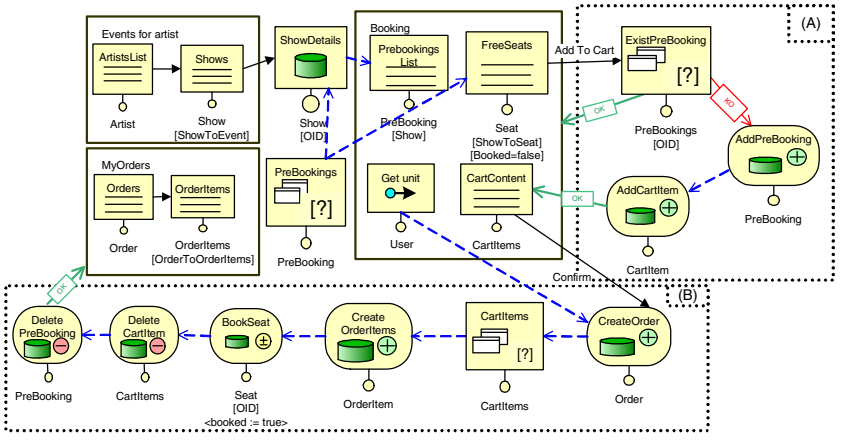

Fig. 2. Hypertext model for the tickets reservation

This solution is good enough to solve our initial problem. However, note some issues arising in this solution: on one hand, the whole process is performed at server side while part of the business logic could be at client-side reducing workloads in the server. On the other hand, the problem of a double pre-booking of a seat could be avoided if the data of the booking page is updated when a different user pre-books a seat. This is only possible with additional synchronous roundtrips in traditional web applications. This synchronous communication imposes processing, rendering and refreshing the whole page (even those parts that have not changed, e.g. cart items).

To solve these issues, RIA features are necessary. Processing and storing capabilities at client side and asynchronous communications come to the scene.

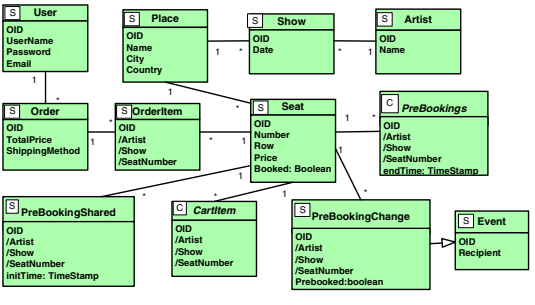

Fig. 3. Data model for the RIA version of the tickets system

3.2 The Ticket e-Shop Revisited Using RIA Capabilities

Conceptual Data Model. (Fig. 3): This data model adds the persistence level and location of the entities. *CartItem* volatile entity is marked as client because each client has a cart and its content is managed at client side. *PreBookings* entity is duplicated following [1] (*PreBooking* volatile at client side and *PreBookingShared* persistent at

server side) and two new entities are defined to support event notification according to [17]. For the latter, *Event* stores the information related to each event that is related (through a generalization) with *PreBookingChange* that stores changes in the *PreBooking* or *PreBookingShared* entities.

Conceptual Hypertext Model. (Fig. 4 and Fig. 5): In Fig. 4 the new RIA operations chain (*A* from Fig. 2) is depicted. In this model, the pre-bookings are permanently updated since an event is sent whenever a user makes a pre-booking. This event is received by the rest of clients to update their *PreBooking* entity (Fig. 4 right). Using this approach, users may work with their local pre-bookings (e.g., sorting by expiration time) avoiding continuous invocations to the server.

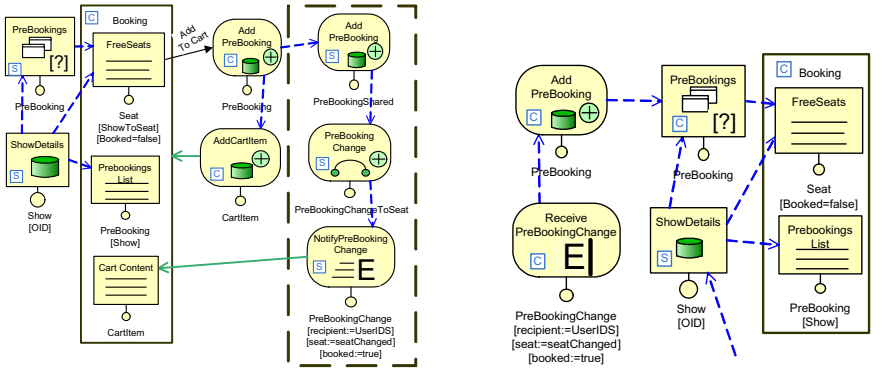

Fig. 4. Adding items to the cart and notifying the rest of users (left-side) and Reception of the PreBookingChange event by clients (right-side)

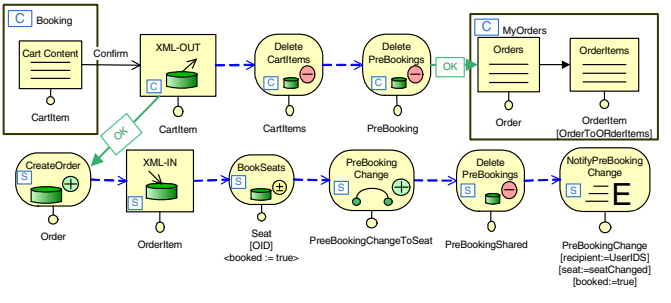

Fig. 5. Client confirms the buying and the order is processed

Regarding to the operation chain *B* (from Fig. 2), now part of the pages (Figure 5) are at client side (i.e., *Booking* and *MyOrders* pages) and also part of the operations related with these content management (i.e., *Delete CartItems* and *Delete PreBookings*). XML_{OUT} is used to collect the items in the cart (pre-booked by the user) at client side and to send this data to the server creating the corresponding order. Server side part of the operation chain also removes the seats from the *PreBookingShared* entity since these seats become booked. This action implies a new event raised by the

server to notify the rest of the clients that these seats have been definitively booked. At the client side, the event is received through the *ReceivePreBookingChange* event unit (not shown due to space limitations)

The process used for migrating the system may introduce potential problems that could compromise its applicability:

- Firstly, observe that the original model has been **tangled** with concerns related to distribution, synchronization, event notification or persistence. This situation is harmful for **readability** and **understandability** complicating, thus, the maintainability of the system.
- Secondly, the adaptations performed are **not reusable** since they should be applied again in any new migration process, and these adaptations will be very similar independently of the application context.

4 The Approach

Following the Aspect-Oriented Modeling (AOM) principles [4] three different meta-models are defined: CMM, RMM and WMM. CMM is the meta-model of the model being migrated (core functionality in AOM terminology so it is called Core metamodel). RMM is an extended CMM with RIA features. In our context these RIA features can be considered as aspects in AOM terminology. Finally, WMM is the weaving meta-model. Fig. 6 shows the basic architecture of the approach. The legacy Web application is represented as the model M that conforms to CMM. RIA features are collected in the model M' that conforms to RMM. The composition (weaving) of both models, using the weaving model WM, produces the model M'' that conforms to RMM. M'' represents the resulting model of the migration process from the legacy Web application to the final RIA. In other words, M'' would be a similar solution to that provided in section 3.2, but here obtained following a systematic approach that avoid the two problems identified in previous section.

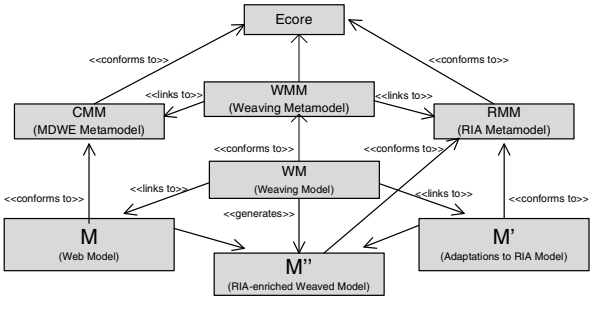

Fig. 6. Weaving of models to obtain the migrated application

4.1 RMM: WebML Metamodel with RIA Features

Among the different WebML metamodel definitions, the work in [16] has been selected due to its completeness and suitability for our approach. This metamodel is enriched with the RIA features introduced in Section 2 as follows (see **Fig. 7**):

- A new attribute *tier* is defined as a member of the metaclass *WebML::ModelElement*. This attribute may hold two different values: *client* and *server*. It indicates the architectural tier of existence for a data entity, or the tier of processing for a hypertext unit.
- New types of units are defined for marshalling data by means of the XML_{IN} and XML_{OUT} metaclasses. They are introduced in the package *ContentManagement* inheriting from *EntityManagementUnit* metaclass.
- And, finally, the units for modeling distributed events, *Send Event* and *Receive Event*, are defined as new kinds of operation units: *Notification Management Unit*.

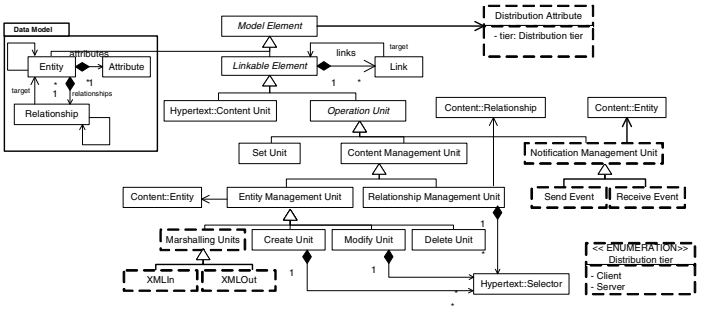

Fig. 7. WebML Extended Metamodel to cope with RIA concepts

4.2 WMM: The Weaving Metamodel

The WMM (Fig. 8) is defined as an extension of the generic model weaver AMW metamodel [5] [6] for the Model Driven Web Engineering [14] (WebML) domain. The composition process consists on the definition of weaving models and their processing by means of generic transformation rules defined in ATL [9]. For the sake of simplicity, we only consider here the extensions needed to process WebML data entities and operation units.

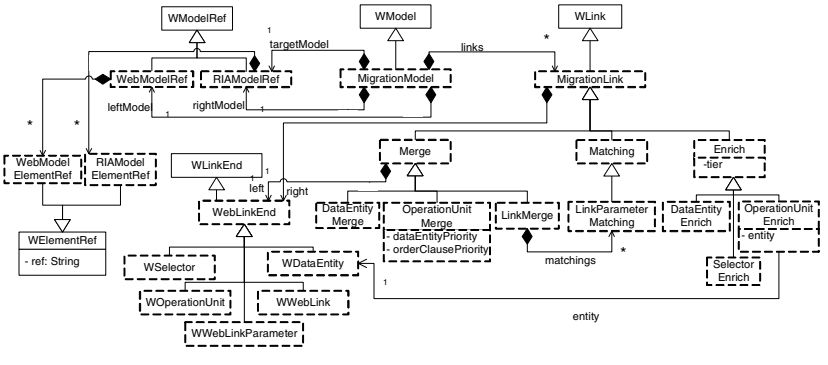

Fig. 8. WMM excerpt (extension of the AMW core metamodel)

As shown in Fig. 8, basically, WMM extends AMW metamodel as follows:

- A new type of weaving model, *MigrationModel*, is defined. It contains references to the woven models: the legacy Web Model (*leftModel*), the RIA model (*rightModel*) and the final model (*targetModel*). It is also composed by a collection (*links*) of *MigrationLink*.
- The metaclass *WLink* has been extended by the definition of three types of migration links: *Merge*, *Enrich* and *Match*. They define how the elements from the source models are woven into the target model.
- The *Merge* metaclass defines a merging connection between a model element from the Web Model and a model element from the RIA model of the same type. Concretely, it may define a link between two data entities, two operation units or two Web links. In case of merging operation units, additional information may be specified to indicate the selection priority of their different components. This information is maintained by the attributes *dataEntityPriority* and *orderClausePriority*. These attributes may hold the values *left* and *right*, indicating which end is selected to compose the target unit. Thus, the *Merge* instances are used to generate a target model element with the selected information of the two source elements.
- The *Enrich* metaclass allows specifying the tier (client or server) of the target model element. In its basic form, this link has only one end that refers to a model element from any of the source models. This metaclass has been extended with three new types, indicating the type of model elements that can be enriched, i.e. data entities, operation units and selectors. As a special case, we highlight the *OperationUnitEnrich* which additionally allows the instantiation of its components, e.g. its related data entity according to the *entity* reference. The Enrich instances of a WM work as annotations of Web elements useful for their transformation to RIA elements.
- The *Matching* metaclass defines a mapping between a model element from the Web Model and a model element from the RIA model. In concrete, the *LinkParameterMatching* defines the parameter matching within a merged link, i.e. it specifies the mapping between the source parameters from the left link and the target parameters from the right link.
- The metaclass *WebLinkEnd* establishes the joinpoint model of the source models. In other words, it defines the set of elements that may be linked by the weaving model (selectors, operation units, data entities, links and link parameters).

4.3 The Composition Process

The current composition process is performed by means of the ATL model transformation language. We have defined a set of ATL rules that generate the final RIA model taking as input the weaving model that links the legacy Web model and the RIA model. These ATL rules are only dependant on the WMM and the CMM. So, once defined for a concrete tuple (wmm, cmm), they can be automatically applied to any migration process conformed to that metamodel tuple.

In order to keep simple the definition and maintainability of this set of ATL rules, we specify the weaving process focusing on the first-class entities of WebML, i.e., entities, units, pages and areas, which drives the composition. In this sense, following

the layer decomposition of WebML, we have grouped the ATL rules in two different subsets: one for data entities, and another one for hypertext elements. Here we explain only the rules defined to merge data entities and operation units.

For WebML **data model composition**, two basic rules are defined:

Rule 1. Migrating not merging data entities. They appear in the weaving model as *DataEntityEnrich* instances. They are the data entities from any of the source models that remain barely unchanged on the target model. The application of this rule only produces two kinds of modifications over a data entity: (1) updating the value of its tier attribute; and (2) resolving the target entities of its relationships pointing to a merging data entity (dangling relationships).

Rule 2. Migrating merging data entities. They appear in WM as *DataEntityMerge* instances. The application of this rule produces a new data entity as the result of merging the linked entities of the source models. This new entity takes its name from the source entity of the legacy Web model. It contains the union of the attribute sets contained by the merging entities. It is also composed by all the relationships whose origin was one of the source entities. In this case, it is also necessary to solve the dangling relationships generated, referencing the merged unit in the target model. Finally, the redundancy is removed from the collection of relationships.

For WebML **hypertext model composition** (only operation units), three basic rules are defined:

Rule 1. Migrating not merging operation units. They appear in the weaving model as *OperationUnitEnrich* instances. They are the operation units from any of the source models that remain barely unchanged on the target model. The application of this rule only produces three kinds of modifications over a operation unit: (1) updating the value of its tier attribute; (2) resolving the related data entity according its *entity* reference; and (3) removing its merging outgoing links (processed later by the rule 3).

Rule 2. Migrating merging operation units. They appear in the weaving model as *OperationUnitMerge* instances. In this case, the application of this rule produces a new operation unit as the result of merging the linked units of the source models. This new unit takes its name from the source unit of the legacy Web model. Next, the description of how the different components of a unit are merged is shown:

1. If the units are related to a data entity, the weaving model specifies which one will be used in the final model by means of the *dataEntityPriority* attribute.
2. If they both present selectors, all their conditions are concatenated in the final selector clause, indicating the tier in which they will be processed. In the weaving model, *SelectorEnrich* instances specify the processing tier of each selector.
3. If they both present order clauses, the sorting attributes will be concatenated in the final order clause, according to the sequence specified by the *orderClausePriority* attribute
4. Regarding the link composition, in this case, we only focus on the outgoing links because they are defined as components of the merging units. In this sense, considering the operation unit merging, we only selected the set of not merging outgoing links because their target unit and parameter matching are not modified.

Rule 3. Migrating merging links. They appear in the weaving model as *LinkMerge* instances. In this case, the target unit and the parameter matching of the target link must be solved. The target unit selected is always the one pointed from the right link. And the parameter matching is established according to its collection of *LinkParameterMatch* instances (matching).

Fig. 9 shows a simplified version of one of the ATL rules implemented, in particular, the rule to enrich operation units (rule 1 of hypertext model composition).

```
helper def: entity2rule: Map(CMM!Entity, WMM!MigrationLink) =
  CMM!MigrationLink.allInstances()->iterate(mlk;
    res: Map(CMM!Entity, WMM!MigrationLink) = Map{}|
    res->including(mlk.left,mlk));

rule OperationUnitEnriching {
 from
   oue : WMM!OperationUnitEnrich
 to
   ou : RMM!OperationUnit (
     name <- oue.left.name,
     entity <- thisModule.entity2rule(oue.entity),
     tier <- oue.tier,
     links <- oue.left.links->union(oue.right.links).asSet()
  )
}

rule LinkMigration {
 from
   lnk : CMM!Link
 to
   mlnk: RMM!Link (
     type <- lnk.type,
     target <- thisModule.target2rule(lnk.target)
     )
}
```

Fig. 9. ATL rule to deal with OperationUnitEnrich entities

Although we have only presented adding and merging operations, our approach supports also deletion of elements from the legacy Web application. The simplest form of specifying the deletion of a concrete element is not referencing it from the weaving model.

5 The Motivating Example Revisited

This section illustrates how the approach presented in Section 4 may be applied in a real application (our ticket e-shop example). In particular, this section describes how to model separately a concrete crosscutting concern: synchronization. The behavior of this concern has been defined by means of synchronization patterns so that they are encapsulated into separated models as a highly reusable RIA concern. Those patterns may be stored in

model repositories to facilitate their localization and instantiation in future migration processes, reducing costs substantially. Finally, the section shows a particular weaving model (based on the meta-model shown in Fig. 8) used to generate the final RIA system, composing, thus, the original web model with the synchronization patterns.

5.1 Aspect Identification: Synchronization Scenarios

One of the main characteristics of RIA applications is the distribution of data and business between server and client minimizing the communications between them. However, it usually involves synchronization mechanisms to ensure the data consistency at both sides of communication. This implies that many actions related to the synchronization are scattered throughout the system and tangled [20] with the core functionality and business logic of the system (as mentioned in Section 3.3).

In this setting, based on the analysis of many RIA applications (and, in particular, our running example), we have classified the synchronization behavior involved in any RIA application in terms of two different dimensions: source of synchronization (client or server) and matter of synchronization (operation or data). This classification is presented in Fig. 10.

Observe that several kinds of synchronization scenarios may be classified into the same category. As an example, when a client is initialized or reconnected (when offline mode is available), the server must send the data required to run the application to the client. As a different example, when some data have changed at the server tier, the client copy of these data must be synchronized according to these changes. This synchronization is performed in RIA by event notification (e.g., observe the *NotifyPreBookingChange* unit used in the cart confirmation process in our running example, Fig. 5).

The event notification may be also used to synchronize an operation carried out at server tier. In this case, the event stores the data regarding to the operation produced (see also the *NotifyPreBookingChange* unit in Fig. 4 or the task assignation to a user event presented in [17]).

Synchronizations from client to server are also typical in RIA environments (especially in collaborative environments). For instance, when a client reconnects (after working in offline mode) the client must send to the server the data collected in order to be synchronized (e.g., see the cart synchronization using the *XMLOut* unit in Fig. 5). Also in collaborative applications, the replication at server tier of an action performed at client side is very common. This action allows the server (and the rest of clients connected) to keep synchronized (e.g., observe the *addPrebooking* replication when a client adds a seat to the cart in Fig. 4).

5.2 Models for Synchronization Patterns According to RMM

According to the kinds of synchronization identified in previous section, different patterns may be defined to model the behavior of the synchronization concern. In particular, a synchronization pattern has been defined for each type of synchronization. This section presents just one of these patterns: a pattern to model the synchronization carried out when the client must send to the server a set of data (*User Triggered Bulk Data Replication* of Fig. 10)

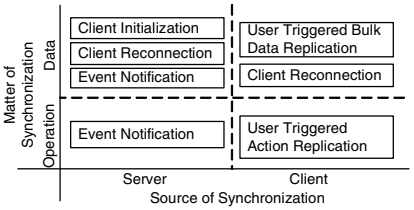

Fig. 10. Scenarios of synchronization based on source and matter dimensions

User Triggered Bulk Data Replication. This pattern happens when a client requires sending a set of data to the server so that these data must be synchronized at client and server tiers. The pattern is defined in terms of the XML_{OUT} and XML_{IN} entities to marshall and unmarshall a set of data (see Fig. 11). As it is suggested by the guidelines defined in [1], this synchronization usually requires the duplication of the data entity which stores the data to be synchronized at client and server tiers (sometimes with different names, e.g. *Order* and *CartItem* entities in our running example).

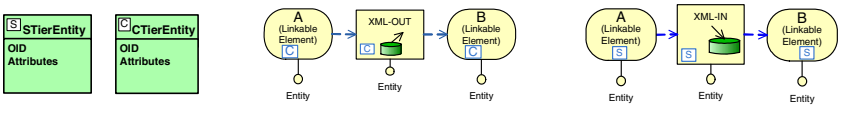

Fig. 11. User Triggered Bulk Data Replication pattern: (a) data entity duplication; (b) marshalling data, (c) unmarshalling data

5.3 Weaving Model

A weaving model, conformed to WMM, is defined to indicate the merging elements that constitute the join points linking the different models. Moreover, this weaving model contains annotations to incorporate additional information for the composition process, such as the distribution tier of different elements of the legacy model. Fig. 12. shows an excerpt of the weaving model for our tickets e-shop.

The weaving model presents the necessary elements to merge the second part (Fig. 11c) of the *User Triggered Bulk Data Replication* pattern with the operation chain *B* from our motivating example. The elements related to the rest of the pattern (Fig. 11a and Fig. 11b) have been omitted due to space limitations. The *OUMerge* instance *V* links the operation unit *Create Order* from the legacy Web model and the linkable element *A* previous to the XML_{IN} unit. In this case, the target unit will be a create unit related to the *Order* data entity (from the left model), as specified by the priority attributes of this instance, and the outgoing link to the XML_{IN} (from the right model), as indicated by a specific *LinkMerge* instance (not shown in the figure). The *OUEnrich* instance *X* indicates that the unit XML_{IN} has to be added to the target model only modifying the data entity related (*entity* reference). The *OUMerge* instance *Z* links the operation unit *Book Seat* and the linkable element *B*. So the target unit will be a modify unit related to the *Seat* data entity and the outgoing link to the *PreBooking Change* unit (both from the left model). Finally, the *LinkMerge* units *W* and *Y* relate the links that must be merged due to the merging of two operation units.

Moreover, two operation units of the chain *B* from the legacy Web Model are deleted, i.e. they do not appear in the final model. These units (selection unit *CartItems* and creation unit *Create OrderItems*) are no longer necessary for the RIA chain *B* redefinition. In this sense, they are not referenced from the weaving model. Observe that the output of the weaving performed (using the ATL rules explained in Section 4.3) is the model M''. The lower part of Fig. 12. shows an excerpt of the application resulting of the composition process, in particular the part obtained by the composition explained by this figure. Note that the models resulting of the whole composition process have been previously presented in the figures of Section 3.2. Thus, the lower part of Fig. 12. shows a snippet of Fig. 5.

The weaving model is the only part that depends on the concrete application. The rules and the ATL transformations generated would be the same for any WebML application. Therefore, the migration of different applications may be performed just defining the corresponding pattern instantiations and weaving models.

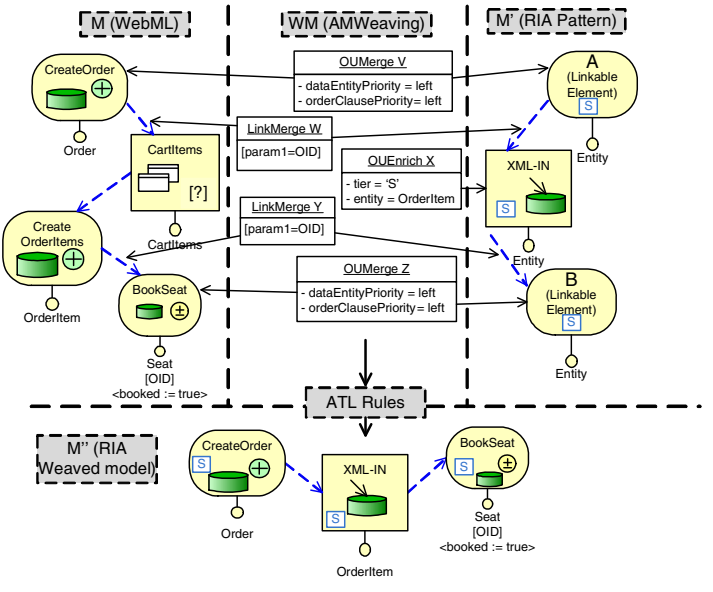

Fig. 12. Snippet of the weaving model

6 Conclusions and Related Works

This paper has presented a systematic model driven re-engineering process, based on a weaving metamodel and composition rules, to perform the migration of traditional Web applications into RIAs. With the proposal here presented this re-engineering process can be barely reduced to the definition of weaving models driving the model composition. Meanwhile the weaving models must be defined specifically for every migration case; the transformation rules are generic and can be applied automatically to any re-engineering process, improving highly the time and effort to carry out such migrations. As an ongoing work, we are studying the application of automatic model matching techniques in order to alleviate the effort of defining the weaving model.

The approach has been presented with the help of a motivating example where synchronization patterns appear as a crosscutting concern that has been separately defined. The patterns ensure the reusability of the migration process to different systems just by defining new weaving models. The applicability of the patterns is not limited to adding RIA features to legacy systems. They may be also applied to new developments where RIA capabilities are added from the beginning. Indeed, similar patterns may be also identified to add different concerns to the system (e.g., to add volatile concerns [7] or context aware information [15].

Crosscutting concerns in Web models have been previously treated in the literature. The approach in [19] allows to clearly decoupling requirements that belong to different concerns. In [15] aspects are introduced, using an asymmetric aspect separation approach which increases the complexity of the obtained models and the learning process. In both approaches RIA issues are not considered.

In [21] a RIA metamodel for OOWS is presented to deal with the new technological challenges arisen with Web 2.0. In [18] an aspect-oriented solution to include RIA issues in OOHDM is presented. However, both works only deal with presentation issues, not dealing with the RIA features here treated.

Although many works have previously introduced the concept of model weaving in WebML, none of them deals with RIA capabilities. In [3] KM3 and Abstract State Machines are introduced defining a Composition Weaving Model to check the consistency among the models of the WebML stack. In [8] a proposal to mix domain specific modeling languages (i.e., WebML and XACML for access control) using a general purpose modeling language (i.e., AMW) is introduced. For the weaving approaches listed above as well as for our work, the WebML metamodel plays a pivotal role. While there is not an official WebML metamodel, many researchers have covered this issue. For example, in [2] a WebML metamodel is presented to transform WebML models to MDA.

Although here applied to WebML, the proposal is generic enough to be applied to other Web modeling languages. The scope of our project is broader than here shown and includes not only other Web modeling languages but also the migration of legacy applications that have not been implemented using a modeling notation. In this case, an important process of reverse engineering and business process reengineering is needed.

References

1. Bozzon, A., Comai, S., Fraternali, P., Carughi, G.T.: Conceptual modeling and code generation for rich internet applications. In: 6th international Conference on Web Engineering ICWE ’06. LNCS, vol. 263, Springer, Heidelberg (2006)
2. Brambilla, M., Fraternali, M., Tisi, M.: A metamodel transformation framework for the migration of WebML models to MDA. In: MDWE, CEUR Workshop Proceedings, vol. 389, pp. 91–105. CEUR-WS.org (2008)
3. Cicchetti, A., Di Ruscio, D.: Decoupling web application concerns through weaving operations. Sci. Comput. Program. 70(1), 62–86 (2008)
4. Clarke, S., Baniassad, E.: Aspect-Oriented Analysis and Design: The Theme Approach. Addison-Wesley Professional, Reading (2005)
5. Del Fabro, M., Bézivin, J., Jouault, F., Breton, E., Gueltas, G.: AMW: A generic model weaver. In: Procs. of IDM ’05, pp. 105–114 (2005)

6. Del Fabro, M., Valduriez, P.: Towards the efficient development of model transformations using model weaving and matching transformations. Software and Systems Modeling 8(3), 305–324 (2009)
7. Ginzburg, J., Distante, D., Rossi, G., Urbieta, M.: Oblivious Integration of Volatile Functionality in Web Application Interfaces. Journal of Web Engineering 8(1), 25–47 (2009)
8. Hovsepyan, A., Van Baelen, S., Berbers, Y., Joosen, W.: Specifying and Composing Concerns Expressed in Domain-Specific Modeling Languages. In: 47th International Conference, TOOLS EUROPE '09, pp. 116–135 (2009)
9. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
10. Koch, N., Pigerl, M., Zhang, G., Morozova, T.: Patterns for the Model-Based Development of RIAs. In: Gaedke, M., Grossnilkalus, M., Diaz, O. (eds.) ICWE 2009. 9th international Conference on Web Engineering 2009. LNCS, vol. 5648, pp. 283–291. Springer, Heidelberg (2009)
11. Linaje, M., Preciado, J.C., Sanchez-Figueroa, F.: Engineering Rich Internet Application User Interfaces over Legacy Web Models. IEEE Internet Computing 11(6), 53–59 (2008)
12. Manolescu, I., Brambilla, M., Ceri, S., Comai, S., Fraternali, P.: Model-driven design and deployment of service-enabled web applications. ACM Trans. Internet Technol. 5(3), 439–479 (2005)
13. Meliá, S., Gómez, J., Pérez, S., Díaz, O.: A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. In: Eighth international Conference on Web Engineering ICWE '08, pp. 13–23. IEEE Computer Society, Los Alamitos (2008)
14. Moreno, N., Romero, J.R., Vallecillo, A.: An Overview of Model-Driven Web Engineering and the MDA. In: Web Engineering: Modeling and Implementing Web Applications, pp. 353–382. Springer, Heidelberg (2007)
15. Schauerhuber, A., Wimmer, M., Schwinger, W., Kapsammer, E., Retschitzegger, W.: Aspect-Oriented Modeling of Ubiquitous Web Applications: The aspectWebML Approach. In: 14th Annual IEEE international Conference and Workshops on the Engineering of Computer-Based Systems, pp. 569–576. IEEE Computer Society, Los Alamitos (2007)
16. Schauerhuber, A., Wimmer, M., Kapsammer, E.: Bridging existing Web modeling languages to model-driven engineering: a metamodel for WebML. In: 2nd international workshop on model driven Web engineering MDWE '06 (2006)
17. Toffetti-Carughi, G., Comai, S., Bozzon, A., Fraternali, P.: Modeling distributed events in data-intensive Rich Internet Applications. In: Benatallah, B., Casati, F., Georgakopoulos, D., Bartolini, C., Sadiq, W., Godart, C. (eds.) WISE 2007. LNCS, vol. 4831, pp. 593–602. Springer, Heidelberg (2007)
18. Urbieta, M., Rossi, G., Ginzburg, J., Schwabe, D.: Designing the Interface of Rich Internet Applications. In: LA-WEB '07. Latin American Web Congress, pp. 144–153 (2007)
19. Valderas, P., Pelechano, V., Rossi, G., Gordillo, S.: From crosscutting concerns to web systems models. In: Benatallah, B., Casati, F., Georgakopoulos, D., Bartolini, C., Sadiq, W., Godart, C. (eds.) WISE 2007. LNCS, vol. 4831, pp. 573–582. Springer, Heidelberg (2007)
20. Van den Berg, K., Conejero, J., Hernandez, J.: Analysis of Crosscutting in Early Software Development Phases based on Traceability. In: Rashid, A., Aksit, M. (eds.) Transactions on AOSD III. LNCS, vol. 4620, pp. 73–104. Springer, Heidelberg (2007)
21. Valverde, F., Pastor, O.: Facing the Technological Challenges of Web 2.0 - a RIAModel-Driven Engineering Approach. In: Vossen, G., Long, D.D.E., Yu, J.X. (eds.) WISE 2009. LNCS, vol. 5802, pp. 131–144. Springer, Heidelberg (2009)