Model Driven Development of Process-centric Web Applications

Mario L. Bernardi[1], Marta Cimitile[2] and Fabrizio M. Maggi[3]
[1]*Department of Engineering-RCOST, University of Sannio, Benevento, Italy*
[2]*Faculty of Jurisprudence, Unitelma Sapienza University, Rome, Italy*
[3]*Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands*

Keywords: Model Driven Engineering, Declarative Processes, Domain Specific Languages, Code Generation, Web Applications Development.

Abstract: Despite Model Driven Engineering (MDE) approaches are largely used to develop, update and evolve Web Applications (WAs), the use of these approaches for the development of process-centric WAs is still very limited. This is an important issue in the context of MDE considering that WAs are often used to support users in the execution of business processes. In this paper, we propose the integration of three MDE metamodels used to represent the structure of information, service and presentation layers of a WA with the metamodel of Declare, a declarative language for business process rapresentation. The declarative nature of Declare allows us to combine an efficient roundtrip engineering support with the advantages of an MDE approach. We present and discuss a case study where the proposed approach is used to develop a typical online shopping application with the aim to validate and verify the feasibility and the effectiveness of the approach.

1 INTRODUCTION

In the last years, Model Driven Engineering (MDE) approaches have been used to quickly and easily develop, update and evolve Web Applications (WAs). In this way, it has been possible to meet the increasing necessity for a rapid customization of WAs according to the changeability of the market requests.

Despite the ample diffusion of MDE principles applied to the development of WAs (Ceri et al., 2000; Bernardi et al., 2010), the adoption of these approaches in the context of process-centric WAs (Retschitzegger and Schwinger, 2000) has not largely been investigated and still suffers from a lack of uniformity in terms of languages, technologies and models during the entire development life-cycle.

Moreover, the generation of the business process management components is still not completely automated and needs several manual customizations that are often not trivial and require the developers to iterate the generation steps several times. This lack of adequate round-trip engineering support still thwarts the benefits arising from the MDE adoption.

Nevertheless, the most limiting aspect for a widespread adoption of MDE in industrial contexts is the lack of flexibility in the process definition. Therefore, it is necessary the integration of MDE approaches with adequate metamodels for process representation that ensure flexibility.

In this paper, we propose an approach based on the integration of three MDE metamodels used to represent the structure of information, service and presentation layers of a WA with the metamodel of Declare (Pesic, 2008; Pesic et al., 2007; van der Aalst et al., 2009), a declarative language to represent business processes. This approach enables the automatic generation of a fully working prototype of a process-centric WA. The use of MDE technologies in conjunction with a declarative process modeling language allows the developers to represent the behavior of a WA with a high level of flexibility as a compact set of constraints (Zugal et al., 2011). Furthermore, this approach allows the developers and the designers to work on the process only at the model level without spending any effort customizing and maintaining the generated code.

The paper is structured as follows. In Section 2, we discuss the related work. In Section 3, we describe the proposed metamodels, their integration and the generation approach. Section 4 discusses the application of our approach to a case study while Section 5 provides some conclusive notes and remark.

L. Bernardi M., Cimitile M. and M. Maggi F..
Model Driven Development of Process-centric Web Applications.
DOI: 10.5220/0004069903400346
In *Proceedings of the 7th International Conference on Software Paradigm Trends* (ICSOFT-2012), pages 340-346
ISBN: 978-989-8565-19-8

2 RELATED WORK

The topic of the integration of process modeling techniques with MDE approaches for the development of WAs is not new in literature. In (Merialdo et al., 2003), the authors use a workflow conceptual model supporting the interaction between the hypertext and an underlying workflow management system. In (Koch et al., 2004), an approach is proposed based on the integration of the Object-Oriented Hypermedia method (OO-H) and the UML-based Web Engineering (UWE). In (Torres and Pelechano, 2006), Business Process Management (BPM) (Weske, 2007) and Object Oriented Web Solution (OOWS) (Fons et al., 2003) are combined: model-to-model transformations are used to generate a navigational model from the BPM definition while model-to-text transformations are used to produce an executable process definition in BPEL. This approach has been extended in (Brambilla et al., 2006).

The integration of MDE with process modeling languages proposed in literature has always been based on procedural notations. In our approach, we decided to integrate the MDE methodology with Declare, a declarative process modeling language proposed by Pesic and van der Aalst in (Pesic, 2008; Pesic et al., 2007; van der Aalst et al., 2009). Declare was adopted because it is equipped with a powerful graphical notation together with a formal Linear Temporal Logic (LTL) semantics. Based on its formal semantics, tools for the execution and the verification of Declare models have been developed[1] that are useful in simulation and rapid prototyping to provide formal verification of process properties. Moreover, our preliminary studies shows several advantages, concerning maintenance, quality and development effectiveness, deriving by the adoption of MDE methodology in combination with a declarative language.

3 THE APPROACH

In this paper, we propose an MDE approach for developing WAs based on four integrated metamodels each one associated to a corresponding Domain Specific Language (DSL) describing a particular aspect of the WA: (i) an information layer metamodel, used to represent domain elements and their relationships; (ii) a service layer metamodel used to model and express system operations and business logic; (iii) a presentation layer metamodel introducing the concepts of Views, Sub-views, Canvas, Widgets and a model-based component hierarchy to describe how the user interface is structured and how it behaves; and (iv) a process layer based on the Declare metamodel, integrated with the previous ones, that allows us to represent the business process underlying the WA as a set of constraints.

In the definition of the WA information layer metamodel, our aim is to use a common set of modeling concepts that allow us to define sound and rich information models. In order to define the necessary constructs and modeling primitives, we decided to adhere to the well known object-oriented standard, i.e., to the semantics and notation provided by the UML class diagrams. The information layer metamodel resembles the standard Entity-Relationship model, widely used and accepted for information modeling tasks. An excerpt of this model is shown in Fig. 1-(a). Here, the main concept is the Classifier that models a generic domain object specialized as an Entity (representing the super-class of unique and persistent objects in the WA domain model). A relationship specification is represented as an Association in the model. It specifies minimum and maximum cardinality of relationship roles, caching attributes, mappings to the relational model and other properties useful to support the generation of the source code and of the other artifacts needed for the WA development. The ValueObject metaclass can represent a TransferObject (needed to handle values), a PrimitiveType (mapped to a language type) or a UserDefinedType that refers to types that need special treatment (e.g., binary objects, images, internationalized text). The metamodel also introduces the Operation concept representing behaviors that refer to the elements of the information layer metamodel.

Fig. 1-(b) shows the metamodel used to represent the service layer of the WA. The main metaclass is ServiceDefinition, defined as an externally accessible entry point in the WA specifying a set of behaviors adhering to a well known interface. This interface and its properties, specified through the ServiceOperation metaclass, allow the designer to customize the service behavior in several ways. Each ServiceOperation can specify a behavior directly or delegate it to one or more operations of a DataAccessObject (DAO).

The presentation layer metamodel, shown in Fig. 1-(c), is devoted to the description of the presentation layer. The UIRoot is the entry point of this layer. It is centered on the Activity concept regarded as either a user activity requiring human intervention or a batch activity that can be performed automatically by the process execution engine. The Activity metaclass is the root of a hierarchy of several kinds of predefined activities such as BatchActivity, CRUDActivity (associated by the code generator to the CRUD services

[1] http://www.win.tue.nl/declare

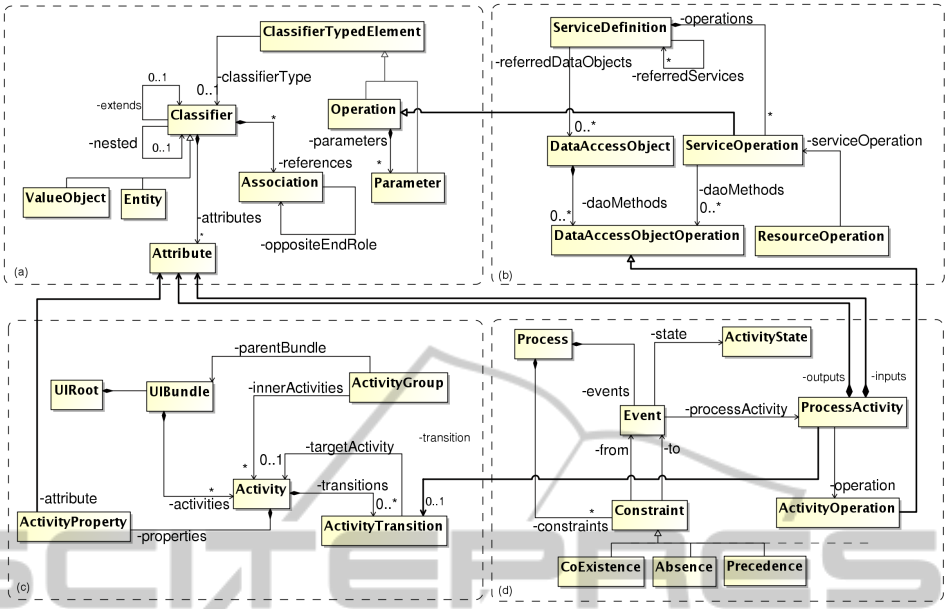


Figure 1: The four linked metamodel.

delegated to DAO classes), ListActivity, TableActivity, GridActivity and ViewActivity. These gather data from the elements of the information model using different kinds of layouts (i.e., lists, tables, views). Another important aspect of the presentation layer metamodel is the navigability: each Activity can specify transitions towards other activities and the post-conditions that must be satisfied on the output data. This is particularly important to ensure the information exchanges among instances of activities as specified by the process definition. As further discussed in the next section, the code generator is responsible for the generation of the ProcessManager code that dynamically injects ActivityTransition instances into ProcessActivity objects by querying the Declare Engine. Finally, when the user chooses to execute an activity, the ProcessManager collects the referenced input instances and process variables by wrapping them into a context object, and expose them to the target activity.

We chose to describe the processes underlying a process-centric WA through Declare, a user-friendly constraint-based language. We do not describe the language in detail (we refer the reader to (Pesic, 2008; Pesic et al., 2007; van der Aalst et al., 2009) for more information about the language) but we will explain the main concepts through the process layer metamodel reported in Fig. 1-(d). In this metamodel, a Process is composed of a set of possible events and constraints on events. The Constraint metaclass is the abstract root of a rich set of constraints (e.g., Precedence, NotCoexistence) each one having a well defined LTL semantics and a graphical representation as specified by the Declare language definition. On the other hand, an Event puts together a ProcessActivity instance with an ActivityState representing a valid state in which that activity can be at a given time. The ProcessActivity is linked to the information layer metamodel mapping attributes to the input/output parameters of the ServiceOperation. It specifies the behavior associated to the ProcessActivity itself. Each ProcessActivity is also linked to the presentation layer being associated to a single UIRoot containing several inner activities. These activities can be created by using the Attribute information thus allowing the user to interact with the process.

3.1 Code Generation

The proposed approach adopts MDE technologies to enable the automatic generation of a fully working workflow-enabled WA starting from its design models. At the heart of our solution there is a generative approach centered on the four linked metamodel illustrated in the previous section. These metamodels are provided as input to a code generator that is able to generate the implementation for a target platform adopting different well-known architectural patterns. In our case, a J2EE spring-based platform was selected as target; it exploits the Command-based Model-View-Controller design to build the UI, an aspect-orientation approach to modularize persistence and logging, and dependency injection to increase flexibility and reuse. The approach is conceived to

fulfill two goals: first, the use of DSL editors supports the designers with precise specifications driving the modeling of all the aspects of the WA (i.e., information, service, presentation and process points of view); second, it allows them to link the models of the WA, written using the DSL editor, to the code automatically generated from the models by the target platform.

The model-to-text transformations are implemented as a hierarchy of templates written in the Xpand language. There are several templates used to generate the UI views and the navigational bars that allow users to move through the application contents. In particular, for each Entity defined in the information model, a page containing a CRUDActivity is generated. This CRUDActivity is built by composing a ListActivity (to implement the retrieve operation) and a contextual menu on the Entity instances that allows the execution of the other CRUD operations. A navigational bar with links to those pages is generated by another template and injected into each view of the application, allowing the user to quickly reach and manage the contents.

Another key page generated is the PendingTasks page, in which all process instances (grouped by process) are shown (using a TableActivity). On each instance a contextual menu is built. This menu mixes a static part (containing process management actions such as start, stop or remove activities) and a dynamic one obtained by querying at runtime the Declare Engine component. This dynamic portion shows only the valid steps that can be executed.

In order to support the approach, a prototypical MDE tool was developed. The core layer of our generation environment is based on the Eclipse Modeling Platform including the Xtext framework, needed to develop the DSL Editors, and the Xpand framework, used to drive the generation tasks. On top of it, the code generation layer contains the Xpand Template manager component. It is responsible of applying templates on the user models in order to generate the needed resources. During this step, several resource-specific generators are used. The IDE integration layer is comprised of four DSL editors (one for each metamodel). They are built using the Xtext framework and integrated into the tool thus allowing the developer to model the application and to start the generation process. The platform also supports the reconciliation of changed artifacts with respect to the models changes.

```
WebApplication OnlineShopping {
 Namespace datamodel {
  Entity Order{
   String name,description;
   List<ItemLine> lines oppositeEnd(ItemLine.
        referencedInOrder); }
  Entity ItemLine {
   Order referencedInOrder oppositeEnd(Order.lines);
   Item item;
   Long quantity;
   Operation BigDecimal linePrice(); }
  Entity Invoice {
   Customer customer;
   Order  orderId;
   BigDecimal total_price,vat;  }
  abstract Entity Item {
   String name,description;
   BigDecimal price;
   List<ItemProperty> properties oppositeEnd(item); }
  Entity Book extends Item{ ... }
  Entity ItemProperty {
   Item item oppositeEnd(properties);
   String property,value; }
  Entity Customer {
   Date birthDate past
   Gender sex ! changeable
   CustomerName name
   List<Address> addresses oppositeEnd(customer); }
   Enum Gender { FEMALE("F"), MALE("M") }
  UserDefinedType Address {...}
      ...
  }
}
```

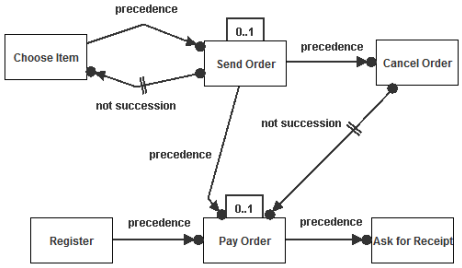
Figure 2: An excerpt of the DSL describing the information model.

4 THE CASE STUDY

We evaluated the feasibility and the effectiveness of our approach through an example application that illustrates its main steps, starting from the definition of the models to the execution of the code generation tasks. The example concerns a typical use case for ordering a product in an online shop.

Fig. 2 shows an excerpt of the DSL instance used to represent the information model of the proposed example. In this model, the main concepts of the domain are represented by entities (Order, ItemLine, Invoice and a small Item hierarchy) linked together by several Association classes. Secondary concepts such as Address, Country and CustomerName are value objects that do not need object identity properties or persistence and hence are represented by means of the PrimitiveType, UserDefinedType and Enum classes. The model also allows us to specify generic collections in order to represent sets of objects of given types. These collections are used to express an Association having roles involving multiple objects (such as the aggregation between Order and Item or the composition between Item and Property).

The service model imports the information model of Fig. 2 and defines the operations needed to implement the business logic of our online shopping example. The model defines a DAO for each Entity in the model and introduces a mapping between Service



(a) The Declare process model.

```
WebApplication OnlineShopping {
  Process processOrder {
    states { READY,STARTED, ... }
    ...
    Event SendOrder {
      inputs { Key Order.id
               String Order.name
               List<Item> Order.items }
      outputs { Key Order.id
                Status Order.status    }
      ActivityOperation execute
          delegates OrderDAO.updateStatus
      Precedence CancelOrder,PayOrder;
      NotSuccession ChooseItem; }
    ... }
```

(b) An excerpt of the DSL describing the Declare model.

Figure 3: Declare model of the online shopping example.

and DAO classes. Each DAO in the model introduces CRUD operations for the entities and each ServiceOperation specifies dependencies on them. The model also includes the definition of three services (CustomerService, OrderService and InvoiceService) used to manage customer information, order management and invoice issuing respectively.

The Declare process model underlying the online shopping WA (depicted in Fig. 3-(a)) is represented as a set of events (e.g., *Choose Item*) and a set of constraints (e.g., *not succession*) on events. In the model, the *precedence* constraint between *Choose Item* and *Send Order* indicates that the buyer cannot send an order without having chosen at least an item. Moreover, the *absence* constraint associated to *Send Order* (labeled with 0..1) means that *Send Order* can be executed at most once in a process instance. The *not succession* between *Send Order* and *Choose Item* specifies that when an order has been sent it is not possible to choose further items anymore. It is possible for the buyer to register an account at any point in time in the process and more than once. It is possible, however, to pay only once, as indicated by the *absence* constraint associated to *Pay Order*. Moreover, it is possible to pay only after the registration and after having sent an order (as indicated by the *precedence* constraints between *Register* and *Pay Order* and between *Send Order* and *Pay Order* respectively). In any point of the process, the buyer can cancel the order but this can be done only after having sent an order (as indicated by the *precedence* between *Send Order* and *Cancel Order*). After having canceled an order it is not possible to pay anymore (as indicated by the *not succession* constraint between *Cancel Order* and *Pay Order*). It is possible for the buyer to ask for a receipt more than once but only after having payed (as indicated by the *precedence* between *Pay Order* and *Ask for Receipt*). In Fig. 3-(b), we show the definition of the SendOrder activity and of the constraints defined on it.

The online shopping application has been made up of 13kLOC of code of which only a small ratio was manually customized by hand (less than 1kLOC). The overall model of the application included 9 entities, 9 DAOs and 4 services. For each Entity, a CRUDActivity was automatically generated with its gap classes allowing for the customization. The process was comprised of 6 events and 9 constraints. After the generation, the entire application was comprised of 54 classes and 12 interfaces; 22 classes were generated as prescribed by the Generation Gap pattern to allow for the customization of data objects, services and activities behaviors.

Fig. 4 is a screenshot of the resulting online shopping WA. In particular, it shows the CRUDActivity of entity Item of the information model. The Items List includes a set of items and for each of them the contextual item menu reports the CRUD operations that are available for it. The upper part of the figure shows the navigation bar that is automatically generated from the information model (with a button for each Entity of the model) and injected into each UI activity to allow navigation: in this case, the application administrator is able to execute CRUD operations on each Entity. The PendingTasks button (in the navigation bar) lists the active process instances for ProcessOrder.

As Fig. 5 shows, for each process (in the example ProcessOrder), a ListActivity is used to list the corresponding active instances. This list reports, for each process instance, the process instance id, the title, the history of the activities already executed (Activity Path), the associated items (quantity and ID) and the current state (the customization class in this case was created to indicate quantity and ID of the selected items). The example shows that in the process order number 258762 three items have been chosen and that this order is valid and running. In the figure, also the contextual menu for this order is shown. The static part of the menu includes the actions for process management (stop, edit and remove). The dynamic part (queried using the Declare Engine) shows

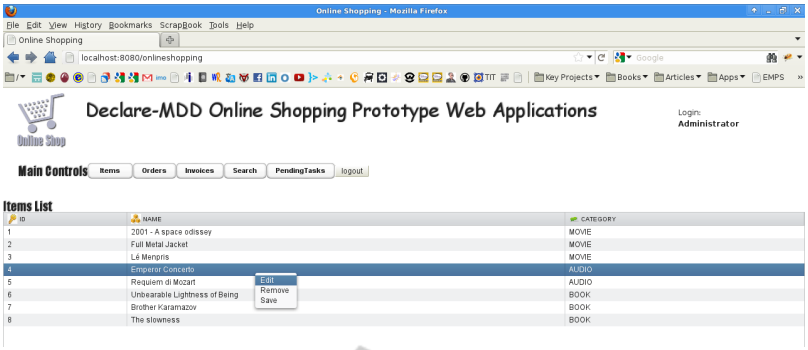


Figure 4: The online shopping example: the auto-generated CRUDActivity of Entity Item.

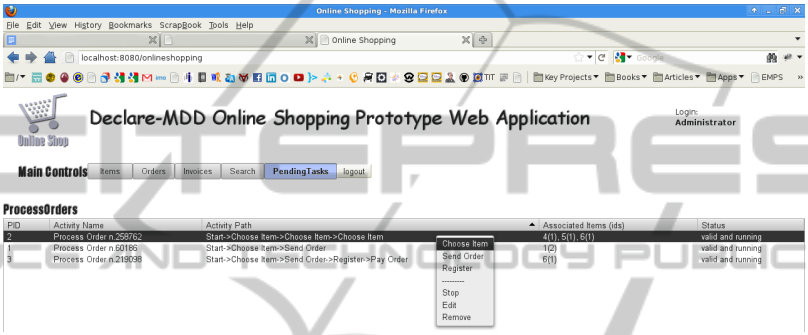


Figure 5: The online shopping example: the possible actions on a running process instance.

that in this process instance it is only possible to execute Choose Item again, Send Order to buy the items or Register according to the reference Declare process model.

5 CONCLUSIONS

This paper proposes an approach based on the integration of four MDE metamodels representing the main components of a WA and the declarative process modeling language Declare. The approach is assessed describing a typical online shopping WA. This application shows the applicability of the approach and underlines some advantages deriving by its adoption. First, we found that our set of metamodels was very effective to model a process-centric WA. Indeed, we observed that in the online shopping application only a small ratio of LOC had to be manually customized. This is a valuable advantage for the developers in the implementation and the maintenance of a WA. The integration of different metamodels supports a clear separation of concerns and enables not only the reuse of the analysis models but also the reuse of the design models (e.g., customer sign in and checkout models) in different contexts.

REFERENCES

Bernardi, M. L., Cimitile, M., Distante, D., and Mazzone, F. (2010). Web applications design evolution with UWA. In *WSE*, pages 3–12.

Brambilla, M., Ceri, S., Fraternali, P., and Manolescu, I. (2006). Process modeling in web applications. *ACM TOSEM*, 15:360–409.

Ceri, S., Fraternali, P., and Bongio, A. (2000). Web modeling language (webml): a modeling language for designing web sites. In *Web Modeling Language (WebML): a modeling language for designing Web sites*, pages 137–157.

Fons, J., Pelechano, V., Albert, M., and Pastor, Ó. (2003). Development of web applications from web enhanced conceptual schemas. In *Conceptual Modeling - ER*, pages 232–245. Springer.

Koch, N., Kraus, A., Cachero, C., and Meliá, S. (2004). Integration of business processes in web application models. *J. Web Eng.*, 3:22–49.

Merialdo, P., Atzeni, P., and Mecca, G. (2003). Design and development of data-intensive web sites: The araneus approach. *ACM Trans. Internet Technol.*, 3:49–92.

Pesic, M. (2008). *Constraint-Based Workflow Management Systems: Shifting Controls to Users*. PhD thesis, Beta Research School for Operations Management and Logistics, Eindhoven University of Technology.

Pesic, M., Schonenberg, H., and van der Aalst, W. M. P. (2007). Declare: Full support for loosely-structured

processes. In *IEEE International EDOC Conference 2007*, pages 287–300.

Retschitzegger, W. and Schwinger, W. (2000). Towards modeling of dataweb applications - a requirement's perspective.

Torres, V. and Pelechano, V. (2006). V.: Building business process driven web applications. In *Springer Berlin / Heidelberg*, pages 322–337.

van der Aalst, W., Pesic, M., and Schonenberg, H. (2009). Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23:99–113.

Weske, M. (2007). *Business Process Management: Concepts, Languages, Architectures*. Springer.

Zugal, S., Pinggera, J., and Weber, B. (2011). The impact of testcases on the maintainability of declarative process models. In *BMMDS/EMMSAD*, pages 163–177.