A Model-Driven Approach for the Fast Prototyping of Web Applications

Mario Luca Bernardi
Department of Engineering
University of Sannio, Italy
mlbernar@unisannio.it

Giuseppe Antonio Di Lucca
Department of Engineering
University of Sannio, Italy
dilucca@unisannio.it

Damiano Distante
Faculty of Economics
Unitelma Sapienza University, Italy
damiano.distante@unitelma.it

***Abstract*—This paper presents an approach for the model-driven fast prototyping of Web applications. The approach exploits well known Model-Driven Engineering frameworks and technologies, such as Eclipse EMF, GMF, and Xpand, to enable the design of a Web application and the automatic generation of the code artifacts implementing a ready to deploy prototype of it. The approach allows to effortlessly and quickly carry out a modeling-generation-validation process in order to validate and refining the design of a Web application before actually implementing it. The paper describes the approach and the process followed to define it, the supporting tools and the technologies used to develop them, and the results from a case study of designing and generating the prototype of a Web application for on-line note taking and sharing. The process and the technologies used to develop the proposed approach can be reused to develop a fast prototyping approach for a different design model and a different target technology platform.**

Keywords: Web Applications, Model Driven Web Engineering, Fast Prototyping, MOF, Metamodel, EMF, GMF, Xpand, MVC, JavaServer Faces

I. Introduction

Web applications (WAs) are usually required to be developed and delivered in a very short time and after that to be updated and evolved even faster. This very short development lifecycle often forces developers to focus on implementation and devote low effort and short time to the design phase, which, in the end, negatively affects the quality of the resulting WA. In this scenario, the availability of methods and tools easing the design of a WA and enabling the automatic transformation of the defined design model into a runnable prototype of the application, may result very useful. By using such tools, developers can validate and incrementally refine the requirements and the design model of the application against a working prototype of it. Moreover, they can regenerate the prototype as many times as needed, before starting the actual implementation of the application.

Such a model-driven fast prototyping approach adopted in the process of development of a WA can significantly reduce the risk of rework during or after the implementation phase, and improve the overall quality of the developed WA. Furthermore, depending on its characteristics in terms of code reusability, the generated WA prototype can also be used as a starting point or a reference for the actual implementation of the WA.

According to Model-Driven Engineering (MDE) principles [1][2][15], software development is focused on the modeling of the application structure, behavior and requirements using formal modeling languages, and on the usage of transformation engines and generators to produce artifacts at a lower level of abstraction and a higher level of detail, and source code at the final stage. The aim is to increase productivity (by reusing standardized models), simplify the design process (by using recurring design patterns), and promoting communication between individuals and teams working on the system.

In MDE, every artifact, including source code, becomes a model element, and the overall development process can be seen as a chain of transformations from one model to the next one enabling the automated implementation of a system starting from its requirements. Models are formalized in a way to be generative by means of meta-models, and each transformation takes models as input and produces some other models.

A high degree of automation in model driven software development makes more efficient the entire software lifecycle, and makes higher the overall quality of the resulting software products. In order to adopt an MDE approach two main elements are required: (*i*) the definition of a meta-model of the design model to use in developing the WA along with a modeling language standard (such as the Object Management Group's (OMG) Meta Object Facility (MOF) [12]) allowing to represent the design elements in a standardized and formal way, and (*ii*) a transformation technology (such as Atlas Transformations Language - ATL, or Query View Transformation - QVT [12]) to generate output models (e.g., source code).

In this paper we present a model-driven fast prototyping approach for WAs which exploits methods and technologies proper of MDE domain implemented upon the Eclipse Modeling Project [18]. The approach is based on the definition of a meta-model that enables designing a WA adopting a Model-View-Controller (MVC) architectural design pattern

[13], adopts the JavaServer Faces technology [17] as target technology for the generated application prototypes, and provides tool support both for the design phase and the code generation phase.

The proposed approach represents the intermediate result of a broader research activity that aims to fully automatize the development process of a WA starting from its conceptual/domain level design models. We present the approach, the way we developed it (which can be replicated to develop other model-driven fast prototyping approaches), the tool supporting it, and describe an example of its application.

The paper is organized as follows. Section II describes the proposed approach, the process and the technologies used to develop it, and the tool support. Section III reports and discusses the results from a case study of designing a WA and generating the source code of a prototype implementing it. Finally, Section IV discusses some related work, and Section V provides some conclusive remarks and announces future work.

II. The Approach

The fast prototyping approach described in this paper adopts MDE technologies to enable the automatic generation of a fully working prototype of a WA starting from its design model.

Within the MDE context, there exists a plethora of modeling technologies and transformation languages for different uses and domains: (*i*) XML/XSLT-based languages, (*ii*) ad-hoc languages (such as, TXL [5]), and (*iii*) languages based on OMG standards (such as, Query/View/Transformation - QVT). QVT principles, also known as Model to Model (M2M) transformation techniques, have also been provided by several other languages of which the ATLAS Transformation Language (ATL) is currently the most widely adopted. Of particular interest for our goals is the MOF Model to Text (MOFM2T) specification [18] which defines a template-based language for transforming models conforming to (i.e., instance of) a MOF metamodel into text. The `Xpand` framework provided under the Eclipse Modeling Project represents an implementation of the MOFM2T concepts.

All such frameworks can be effectively used together to define a generative approach that, starting from the design models of an application, synthesizes both the source code and the metadata required to implement it. Metadata may include, in particular, XML descriptors, SQL scripts to create, populate and eventually update the application database, metadata for an IDE, and documentation at different levels of abstraction.

In this section we describe: (*i*) the process followed to develop the approach, (*ii*) the adopted frameworks and technologies and their role in the model-driven code generation, and (*iii*) the developed tool support.

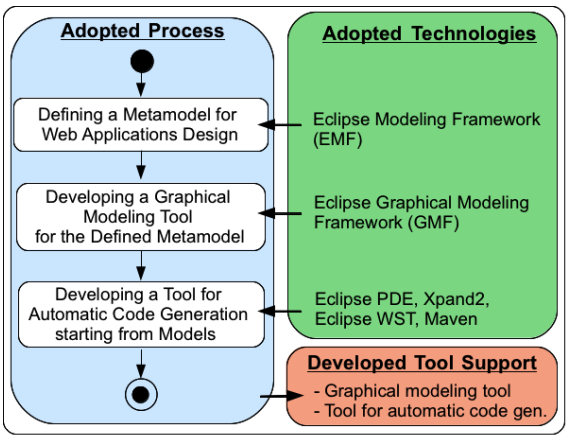

Figure 1. Process and technologies adopted to define our model-driven fast prototyping approach, and resulting tool support.

A. Steps Followed to Develop the Approach

The process followed to define our fast prototyping approach is comprised of the three main steps depicted in the left side of Figure 1 and listed below:

- *Defining a Metamodel for Web Applications Design.* To this aim, we first chose a reference architecture along with a target technology platform for WAs development, and then we defined a metamodel able to represent the design of a WA for the chosen architecture and technology platform. The reference architecture was chosen to be enough abstract to be implemented on different real platforms and enough detailed to express the main design choices and trade-offs. The list of considered architectural design patterns includes: Model-View-Controller (MVC), Publish/Subscribe, Factories, Proxy, just to name a few.
- *Developing a Graphical Modeling Tool for the Defined Metamodel.* A graphical editor was developed to enable the developer to create, view, and edit models which are instances of the defined metamodel. The resulting tool is a modern graphical editor which supports cut&paste, in-place editing, connectors among model elements to represent relationships and so on. A screenshot of the graphical editor is showed in Figure 6.
- *Develop a Tool for Automatic Code Generation from Models.* M2T transformations rules were defined to automatically transform models defined using the developed graphical editor into code, in order to generate the application prototype conforming to the chosen reference target architecture and platform. Such transformations rules were then integrated into a graphical tool which enables the developer to drive the application prototype generation process.

B. The Adopted MDE Technologies and Frameworks

The right side of Figure 1 summarizes the MDE frameworks and technologies used to develop our approach and its supporting tools.

To define our metamodel for WAs design we used the Eclipse Modeling Framework (EMF), while to develop the graphical editor for creating instances of it, we used the Eclipse Graphical Modeling Framework (GMF). Both the modeling tool and the code generation tool supporting our approach have been developed as Eclipse plugins using the Eclipse Plug-in Development Environment (PDE). In this way we were able to support the entire fast prototyping approach (from modeling to code generation) within a single development environment.

EMF provided all the functionalities needed to formally define the chosen design metamodel and to perform an adequate validation of correctness for the generated models. Particularly important was the possibility to attach embedded validation to the metamodel elements (as OCL or Java constraints) and to generate a Java framework to instantiate and handle model instances by means of a Java API. This ensures that every model created with the editor is checked for validity and can be transformed automatically into the running prototype with a reasonable set of default settings and behavior without incurring into runtime errors due to modeling mistakes. Customization at the generation level is possible by making the generation model explicit.

In order to transform the design models of the WA into code and thus generate the application prototype, we chose the Xpand template language, mainly because of its very good support of EMF metamodels. Moreover, its other features (aspect orientation, polymorphism, extensibility, type system abstraction, and validation) came very in handy in generating the implementation of crosscutting features like persistence, logging and especially to customize the generated code.

C. The Defined Design Metamodel and the Tools Supporting the Approach

At the heart of our approach is the definition of a metamodel suitable for representing a generic WA adopting the MVC pattern at an architectural level. This metamodel is used to generate the implementation for some target platforms that adopt MVC as a core architectural pattern (e.g., the JSF implementation platform in the case of this paper). This model fulfills two goals: first, it provides developers with precise specifications guiding the modeling of the WA during high-level design tasks by means of the graphical editor; second it enables to link the model of the WA obtained using the graphical editor to the code generated implementing it for the target platform.

Figure 2 shows a simplified version of the metamodel. The model is based on three packages:

- the *MModel* package, representing the domain or business layer of the WA, that is centered on the ModelClass element made up of Methods and Attributes;
- the *VModel* package, containing meta-classes defined to represent Views and concepts related to them (like VisualComponents and NavigationControlValue objects used to organize navigation among views themselves);
- the *CModel* package, with meta-classes used to represent the control part of the MVC pattern.

The *MModel* package contains the structural elements of the domain model of the application, while the other two (*VModel* and *CModel*) represent the presentation layer. The latter describes the structure and content of pages in terms of visual/behavioral elements and the navigation between Views through the Controllers objects. Associations between Visual and Behavioral elements of the View with Link and Action elements model the desired navigation of the WA.

Figure 3 summarizes the process to generate the prototype of a WA, from a developer point of view. In particular, the figure highlights the technologies adopted for the generated prototype and recalls tool support provided at each step of the process. The prototyping process comprises two steps: first the developer creates a model of the WA, according to the defined metamodel, by using the developed Graphical MVC Model Editor tool; then the Code Generation step is executed which automatically produces the source code of the WA prototype, by using the Eclipse EMF MVC Code Generator tool which, in turns, executes the defined M2T Xpand transformation rules.

Figure 4 depicts the overall architecture of the supporting tools, with their base models and the adopted implementation frameworks.

As showed in the figure, the MVC model, its java framework to create and edit model instances, the wizards and tree-based editors are all implemented in the MVC Design Model component that is based upon the Eclipse EMF and Eclipse IDE layers. This component provides the hooks into the IDE to handle MVC models at Java API level. Upon such component the MVC Graphical Editor, also based on Graphical Modeling Framework (GMF), provides the graphical DSL notation to edit MVC models graphically and a set of wizard integrated into IDE to start the creation of MVC diagrams and to drive the code generation step from them.

Our tool-chain is managed by the generator code plugin, showed on the top of Figure 4. The generator takes the existing models produced by the editor and launches the right Xpand templates for the selected target platform. At

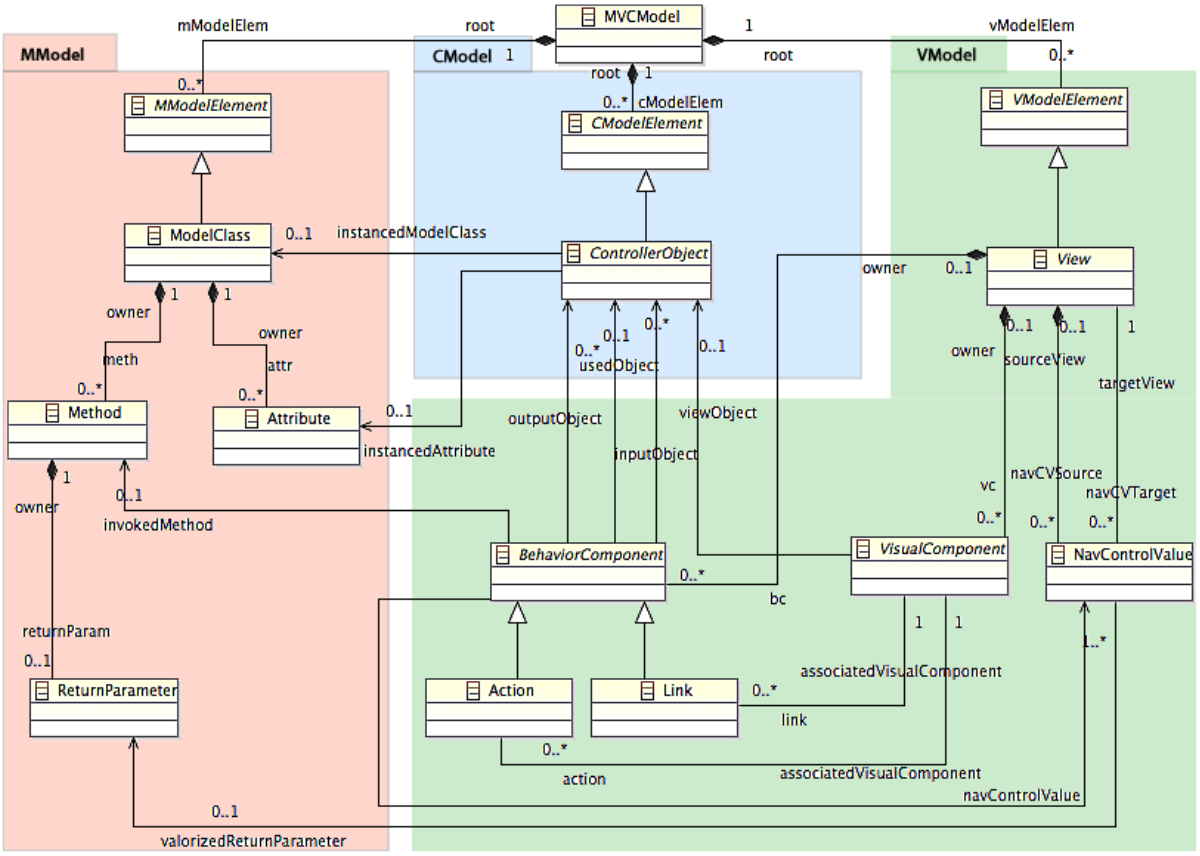

Figure 2. A simplified version of the MVC design metamodel.

the time of writing this paper we have developed a complete generator for a JSF-based application.

Starting from a model instance, textual output can be generated by defining and expanding polymorphic macros based on the model elements and their attributes and properties. The elements of the model can be referenced and accessed by the Xpand transformation expressions specified in the templates. The templates can import other templates and model instances and, in addition, also functional Java extensions can be implemented.

In our approach a Java component (i.e., the Xpand MVC Java Ext block in Figure 4 imported by all templates) implements several handlers and element mappers needed to drive the code generation process and produce all the resources building up the application prototype. Such resources include: HTML/JS code, Java code, SQL scripts, metadata descriptors and project metadata to configure the whole set of resources as an Eclipse WST Faceted project with adequate values for the target platform.

III. From a Model to a Running Application Prototype - The Post-it Case Study

In order to validate our approach, we applied it to automatically generate the prototype of a simple WA for on-line note taking and sharing. The application enables its users to: write notes and classify them by topic; view and reply to other users' notes; select some notes as favourite notes. We will refer to this application as to the "Post-it" application.

We applied our approach by carrying out the two-step process described in Figure 3. First we used the Graphical Model Editor to design the desired WA (according to the MVC pattern) in terms of model classes, presentation views, relations between views and model classes through controller objects, and navigation paths. Then, we provided the resulting design model as input to the code generation tool wizard which produced the source code and all the other files needed to implement the application prototype.

A. The Post-it MVC Design Model

Figure 6 shows an excerpt of the design model defined for the Post-it application. In the upper

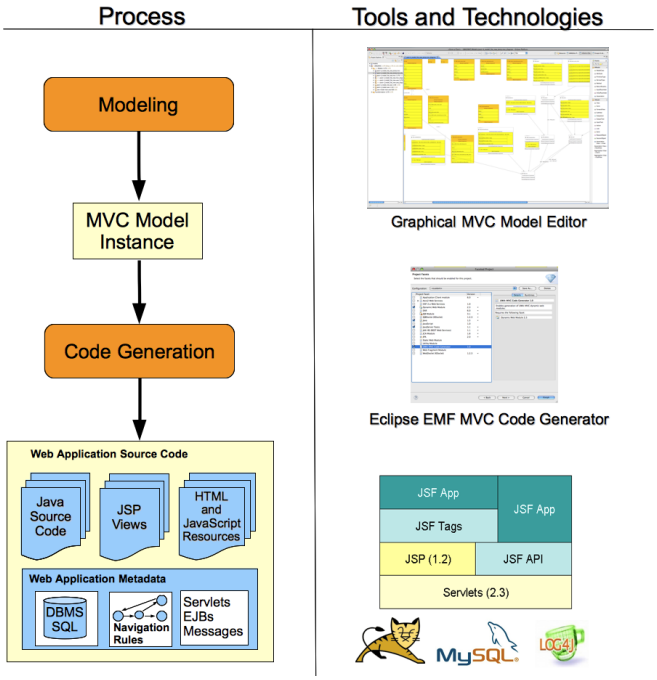

Figure 3. The fast prototyping approach from the developer point of view.

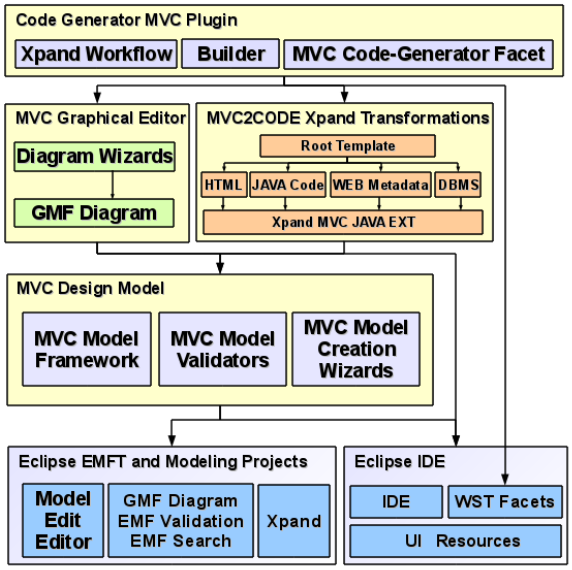

Figure 4. The fast prototyping tools architecture.

part of the diagram we can see five UML classes characterized with the `<<ModelClass>>` stereotype and representing classes belonging to the Model component of the MVC architecture. In particular these classes are: `IME_Author`, `IMC_Author_Collections`, `IME_PostIt`, `IMA_PostIt_Associations`, `IMC_PostIt_Collections` and

```
<?xml version="1.0" encoding="UTF-8"?>
<mvc:MVC xmi:version="2.0" ...xmlns:mvc="http://mvc.ecore">
  ...
  <mModelElem xsi:type="mvc:ModelClass" name="IME_PostIt"
  associationTarget=//@mModelElem.14 ...
  associationSource=//@mModelElem.16 ... />
     <attr name="subject" type="//@mModelElem.1"/>
     <attr name="creationDate" type="//@mModelElem.2"/>
     <attr name="topic" type="//@mModelElem.1"/>
     <attr name="summary" type="//@mModelElem.1"/>
         <attr name="content" type="//@mModelElem.3"/>
         <attr name="idPostIt" type="//@mModelElem.0"/>
         <attr name="numOfVisits" type="//@mModelElem.0"/>
          ...
  </mModelElem>
  ...
  <vModelElem xsi:type="mvc:SubView" name="NMIEPU_PostIt"
  view="//@vModelElem.1">
         <vc xsi:type="mvc:OutputText"
         value="IME_PostIt.idPostIt_:_Integer"/>
         <vc xsi:type="mvc:OutputText"
         value="IME_PostIt.subject_:_String"/>
         <vc xsi:type="mvc:OutputText"
         value="IME_PostIt.summary_:_String"/>
         <vc xsi:type="mvc:OutputText"
         value="IME_PostIt.content_:_Text"/>
         <vc xsi:type="mvc:OutputText"
         value="IME_PostIt.creationDate_:_Date"/>
          ...
  </vModelElem>
  ...
</mvc:MVC>
```

Figure 5. An excerpt of the ecore MVC design model for the Post-it application.

`IMA_Author_Associations`. The first two classes represent domain classes for the designed application (post-it notes and authors), with their attributes and methods (both not shown in the figure for space reasons). The latter four classes represent utility classes including methods to recover lists and associations of instances of the first two types. The diagram also shows by means of UML references, the associations between the `IME_Author` and the `IME_PostIt` classes. The bottom part of the diagram in Figure 6 shows seven UML classes characterized with the `<<View>>` or `<<SubView>>` stereotype and representing classes of the View component of the MVC architecture. In particular the shown classes are the `PMP_Home` and `PMP_PostIt` views and their component subviews. For two of the subviews the diagram also shows the included VisualComponents (user interface widgets) and the name of the model class to which they are associated, through the Controller component of the MVC architecture.

Figure 5 reports a fragment of the ecore source code associated to the Post-it model graphically shown in Figure 6. In particular the shown code fragment refers to the definition of the `IME_PostIt` model class and the `PMP_PostIt` view.

B. *The Post-it Generated Web Application Prototype*

The ecore MVC design model of the Post-it application described above was used as input for automatic code generation step. This step was executed by the code generator tool wizard which produced as output the WA prototype by generating all the software artifacts required to build

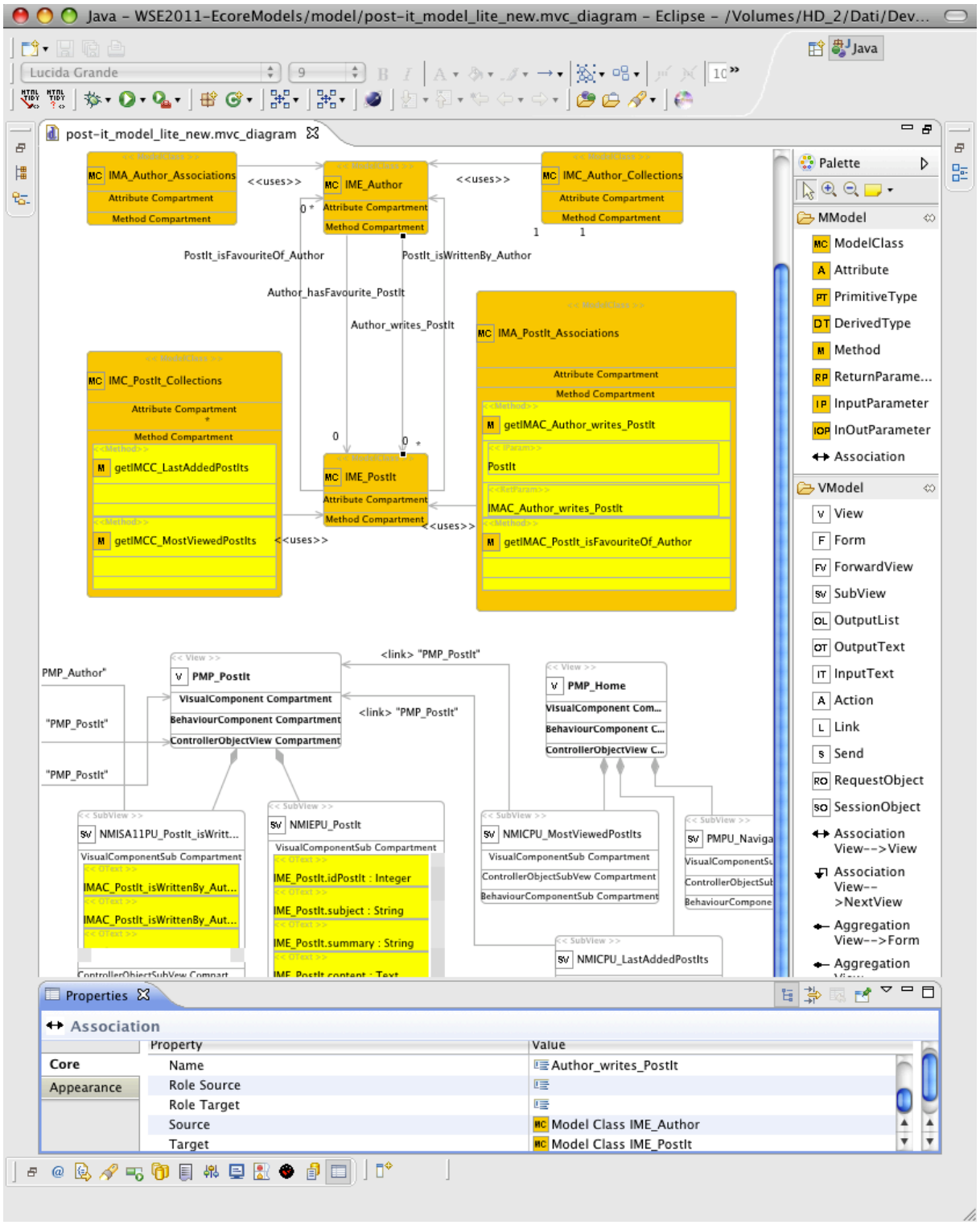

Figure 6. The Graphical Modeling Editor showing an excerpt of the design model of the Post-It note sharing Web application.

it (HTML/JS resources, Java source code, SQL scripts, and project metadata). The generator tool also takes care of copying into the prototype generated Web project the required libraries, such as JDBC drivers, JSF implementation

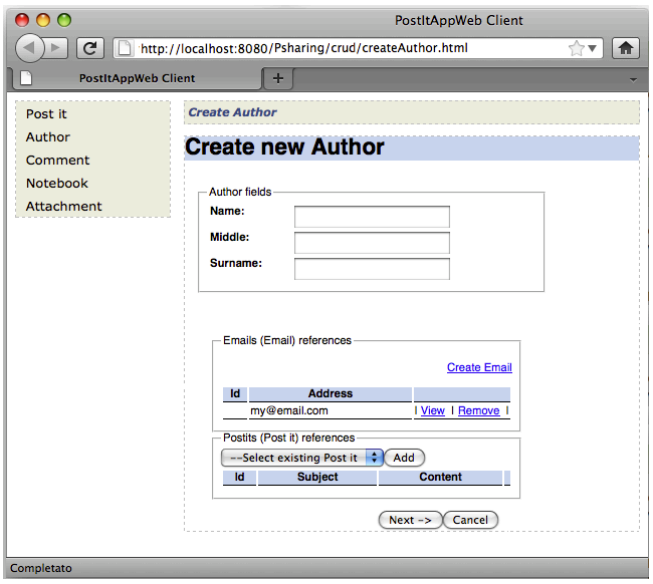

Figure 7. One of the CRUD interfaces enabling populating the database of the generated Post-it Web application prototype.

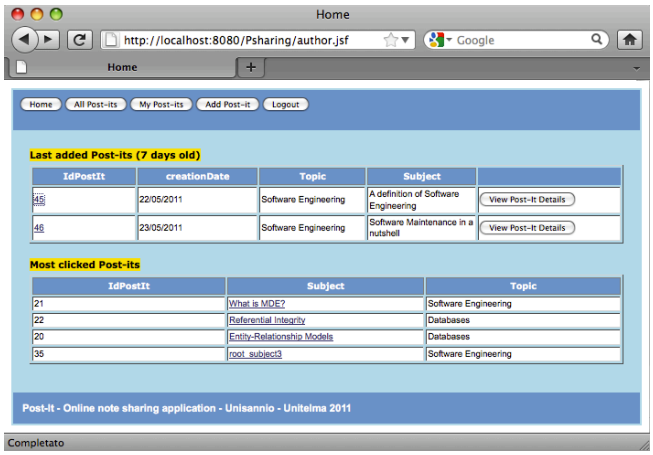

Figure 9. The homepage of the Post-it automatically generated Web application prototype.

library and tag libraries. Furthermore, the tool also generates a set of Create-Read-Update-Delete (CRUD) user interfaces which enable to easy populate and update the database of the application prototype. Figure 7 shows the CRUD interface enabling to create, read, update and delete an author of post-its in the application prototype.

Overall, the set of artifacts produced by the code generator tool constitute a WA ready to be deployed on a servlet container, such as Apache Tomcat, using MySQL as DBMS for data storage. Figure 8 shows the Post-it application prototype loaded into the Eclipse IDE as a Web Dynamic Module project. In particular, the Eclipse package explorer view (on the left side in the figure) shows the list of code artifacts produced for the Post-it application prototype, while the editor view (main window) shows the source code of the JSP implementing the `PMP_PostIt` View.

Figure 9 and 10 show the screenshots of two different pages of the front-end of the generated application prototype: the homepage and the page of the details of a post-it note. The homepage includes a navigation menu and shows two lists of post-it notes: the list of the last added notes (the notes not older than seven days) and the list of the most clicked notes. Each of the entries of these lists provides a preview of a post-it note and enables navigating to a page similar to the one shown in Figure 10. The page of an author (not showed due to space constraints) shows the details of an author of post-its and provides a preview of the post-it notes written by the author. Finally, the page of a post-it note (Figure 10), in addition to showing the details of a post-it, shows a preview of the author and enables navigating to the detail page of the author.

C. Final Considerations, Limitations and Lessons Learnt

The conducted case study confirmed the correct functioning of the tools developed to support our fast prototyping approach and to validate the approach itself. Both the MVC model editor used to design the application to implement and the code generator tool worked properly during their usage and the produced Post-it WA prototype was deployable and run on Tomcat 6.0. Being the code generation phase completely automated and producing as output a "ready to run" application prototype, we were able to go through different "modeling-generation-validation" cycles easily and effortlessly. The fast and automatic generation of a fully functioning prototype of the designed application makes it possible to verify and validate the design itself and to undertake a design refinement process effortlessly. The only activity that the designer is required to carry out manually after generating the source code of the prototype is the database population. In particular, the approach at the moment recreates the database of the prototype application if any change is required to it by the new design of the application. This aspect of the code generation process can be improved by trying to adopt an incremental approach for the SQL script used to create the database. Our fast prototyping approach could be improved in the portion concerning the definition of the application prototype layout and look & feel. In fact, while the user interface aspects assume nowadays a crucial role to the success of a Web site, our approach is currently mainly focused to validating the content and navigation design aspects.

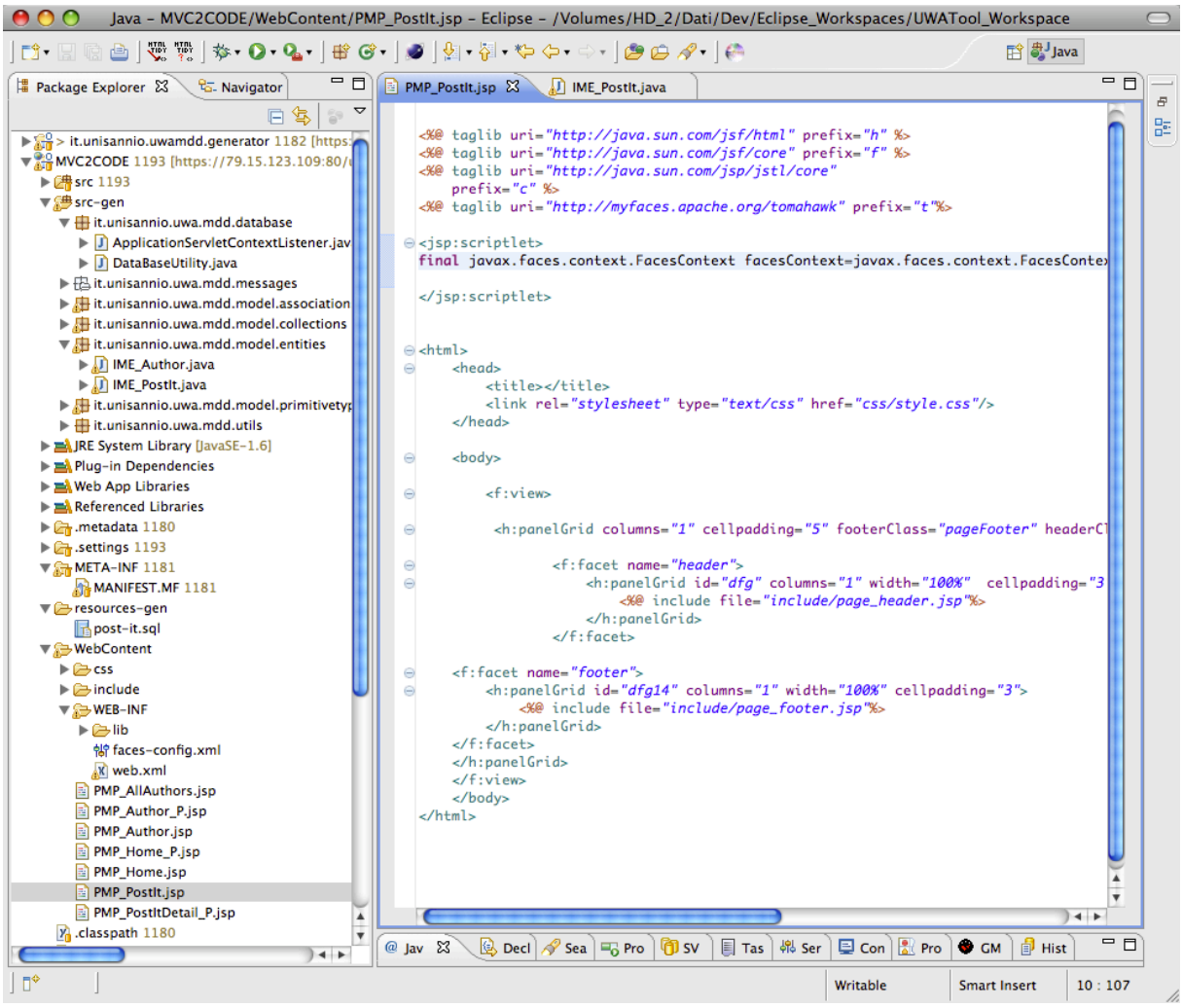

Figure 8. The Post-it Web application prototype as an Eclipse Web Dynamic Module project.

IV. Related Work

In the Web Engineering context, MDE principles are being used to successfully address the development and evolution of WAs. In [10], Nora Koch et al. show how the Model-Driven Web Engineering (MDWE) discipline has arisen and how MDD/MDA principles are applied in the Web Domain to define models and metamodels, to specify model transformations, to manage interoperability issues and to build tools that support the development process.

In particular, Model-Driven Engineering paradigm has been applied successfully by a number of web engineering methods, namely UWE, OO-H, OOHDMDA, and WebML. These methods use models to separate the platform-independent model (PIM) design of web systems from the platform-dependent (PSM) implementations as much as possible. Usually, they include supporting development environments for code generation from model specifications, either fully or partially automated.

UWE [8] follows the MDA principles and uses the OMG standards [12]. The process makes use of model transformations defined at metamodel level and specified by general purpose transformation languages, such as QVT [12] and graph transformations. UWE4JSF [9], a variant of the UWE approach, builds a WA upon a set of meta-models (both for conceptual and low level design) and targets the JSF platform. It shares with our approach the transformations of conceptual models (by means of ATL rules) and provides fully automatic code generation.

OOHDMDA [14] generates servlet-based web applications from OOHDM models. The OOHDMDA approach follows MDA principles by employing the OOHDM conceptual and navigational scheme of a web application as the basic PIM for the MDA process, using any UML-based design tool

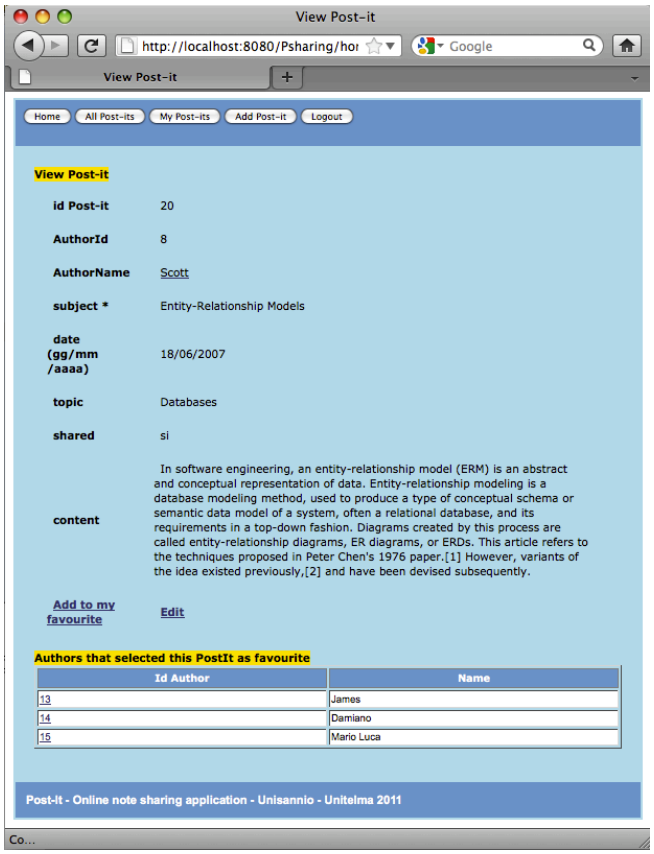

Figure 10. A screenshot of the Post-It Web application prototype presenting the details of a post-it note.

which produces an XMI-file as output. WebML [4], [3], [16] is a model-driven method for the development of data intensive web applications, with an associated supporting CASE tool called WebRatio. WebML follows an MDD approach for mapping its modelling elements onto the components of the MVC Model 2 architecture, which can be transformed into components for different platforms.

Similar to our approach, UWE, and OOHDMDA adopt an MDD process that follows MDA principles for the models. WebML differs from our and other considered approach in that its process is MDD but not MDA. Similar to our approach, WebML uses MVC as architectural pattern for its PIMs.

In [11] the HyperDe environment is presented; it supports the rapid prototyping of Web applications by combining the Model Driven Development approach with the use of Domain Specific Languages. This combination allows the designer/developer to write code by directly manipulating the models that specify the application. HyperDe shares with our approach the use of a meta-model and of the MVC pattern to design a WA, while it uses different technologies to implement the application to deploy.

The main difference between the proposed approach and the considered related ones is that they enable different technologies to be used for the implementation of the PSMs. Our choice of adopting MVC as architecture for the PIM logical model and JSF for the PSM guarantees the availability of a wide range of open-source and commercial technology frameworks to choose from for the different platforms, such as J2EE, .Net and PHP.

V. Conclusions

In this paper we have presented an approach for the model-driven fast prototyping of Web applications developed using Eclipse technologies and frameworks such as EMF, GMF and Xpand2. The approach consists of a two-step process, modeling-generate, and is accompanied by two supporting tools: a modeling tool for defining the design of the application by adopting the Model-View-Controller architectural design pattern, and a generator tool that transforms the defined design model into a "ready to run" prototype of the application. The code generation phase is fully automated and produces an Eclipse faceted Web dynamic project that uses the J2EE JavaServer Faces implementation technology and is ready to deploy on a Tomcat-MySQL platform. A case study conducted on designing and fast prototyping a Web application for online note taking and sharing has shown that the approach is valid and the supporting tools work properly. In particular, the approach enables effortlessly repeating the development cycle "modeling-generating-validating" to verify and incrementally improve the design of the application.

The process and the technologies adopted to implement our approach can be reused to develop the fast prototyping approach for a different design model and/or a different target technology platform. At the same time, our approach can be combined with a conceptual design method for Web applications to realize a fast prototyping approach for this method. We are actually working to this objective with the Ubiquitous Web Application (UWA) design methodology [6]. Other ongoing work is intended to improve the fast prototyping approach itself, in particular with regard to:

- Supporting round trip engineering of generated artifacts, so that manual customizations are taken into account and merged by the generator. This is needed to support manual customization of the generated source code.
- Improving persistence management by supporting JPA and incremental updates for SQL schema scripts.
- Supporting other interesting target platforms (e.g., for mobile devices);

REFERENCES

[1] J. Bezivin. In Search of a Basic Principle for Model Driven Engineering. *The European Journal for the Informatics Professional - Joint issue with NOVATICA*, Vol. 2, pp. 21–24, April 2004.

[2] J. Bezivin. Model Driven Engineering: An Emerging Technical Space. In R. Lammel, J. Saraiva, and J. Visser, editors, *Generative and Transformational Techniques in Software Engineering*, *Lecture Notes in Computer Science*, Vol. 4143, pp. 36–64. Springer Berlin / Heidelberg, 2006.

[3] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[4] S. Ceri, P. Fraternali, and M. Matera. Conceptual Modeling of Data-Intensive Web Applications. *IEEE Internet Computing*, 6(4):20–30, 2002.

[5] J.R. Cordy. The TXL Source Transformation Language. *Science of Computer Programming - The fourth workshop on language descriptions, tools, and applications*, 61(3):190-210, August 2006.

[6] D. Distante, P. Pedone, G. Rossi, and G. Canfora. Model-Driven Development of Web Applications with UWA, MVC and JavaServer Faces. In *Proc. of the 7th International Conference on Web Engineering*, ICWE'07, pp. 457–472. Springer Berlin / Heidelberg, 2007.

[7] R. J. E. Gamma, R. Helm and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[8] N. Koch and A. Kraus. The expressive power of uml-based web engineering. In *Proc. of the 2nd International Workshop on Web Oriented Software Technology*, IWWOST'2002. Springer Verlag, 2002.

[9] C. Kroiss, N. Koch, and A. Knapp. UWE4JSF: A Model-Driven Generation Approach for Web Applications. In *Proc. of the 9th International Conference on Web Engineering*, ICWE '09, pp. 493–496, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] N. Koch, S. Melia-Beigbeder and J. Vara-Mesa. Model-Driven Web Engineering. *European Journal for the Informatics Professional - Joint issue with NOVATICA*, IX(2):40–45, April 2008.

[11] D. A. Nunes and D. Schwabe. Rapid Prototyping of Web Applications Combining Domain Specific Languages and Model Driven Design. In *Proc. of the 6th International Conference on Web Engineering*, ICWE '06, pp. 153–160, New York, NY, USA, 2006. ACM.

[12] OMG - Object Management Group (MOF, MDA, XMI, QVT, UML, MOFM2T) - http://www.omg.org/.

[13] K. I. R. Sridaran, G. Padmavathi. A Survey of Design Pattern Based Web Applications. *Journal of Object Technology*, 8(2):61–70, 2009.

[14] H. A. Schmid and O. Donnerhak. OOHDMDA - An MDA Approach for OOHDM. *Lecture Notes in Computer Science*, 3579/2005:569–574, 2005.

[15] D. C. Schmidt. Model-Driven Engineering. *Computer*, 39:25–31, 2006. IEEE Computer Society

[16] C. Stefano, D. Florian, M. Maristella, and F. F. M. Model-Driven Development of Context-Aware Web Applications. *ACM Transactions on Internet Technology*, 7(1):2, 2007.

[17] JSR 127 - JavaServer Faces 1.0. http://www.jcp.org/en/jsr/detail?id=127 Java Community Process, 2004

[18] The Eclipse Modeling Project. http://www.eclipse.org/modeling