Large-Scale Model-Driven Engineering of Web User Interaction: the WebML and WebRatio Experience

Marco Brambilla and Piero Fraternali

Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio, 34/5. 20133 Milano, Italy

Abstract

This paper reports the experience of WebRatio, a company focusing on Model-Driven Engineering (MDE) tools (WebRatio and WebRatio BPM) and services since 2001. The adopted MDE approach is based on the transformation of models expressed in a Domain Specific Language called WebML (Web Modeling Language) into running applications, with the unique feature of creating not only the back-end data and business logic, but also the Web/RIA front-end, without posing any limitation on the graphical and interaction quality of the user interface. WebRatio has been applied in many industrial projects, some of which have delivered large-scale enterprise applications, generated and maintained completely through MDE practices over the years. In this paper we present the lessons learned within this experience, we describe some success stories and show some quantitative information and evaluation on the usage of the approach.

Keywords: Model-Driven Engineering, Code Generation, Web Engineering, Service Oriented Architecture, BPM, MDE, Model-driven Engineering, WebML, Web Modeling Language, IFML, Interaction Flow Modeling Language, Experience, Lesson Learned, Evaluation, Software Design, Visual Models, Software Engineering, Web, User Interaction

1. Introduction

Model-Driven Engineering (MDE) is the approach to system development that exploits as main artifacts *models*, defined as abstract representations of the system/product under construction. Models are customary in traditional engineering disciplines, where they support the design and verification of a

system/product before its realization and deployment. The application of models to the software engineering industry is as old as the software itself, if one considers that Data Flow Diagrams have been used since the inception of programming, but it is not as pervasive as in other disciplines. Software models are normally used in the analysis and design phases, and then abandoned in favor of programming during implementation and maintenance.

The resistance of software developers to the use of models during implementation and maintenance is motivated by the nature itself of the software product and by the lifecycle of its development process: unlike other engineering artifacts, software can be modified at low cost after deployment, which makes it natural to perform change management directly on the source code; on the other hand, very short development cycles and the difficulty to formalize requirements and change requests, often stated qualitatively, favor a trial-and-error approach where a system is released and then adaptively fixed while in use.

Despite the above mentioned factors, in the past two decades MDE has slowly made its way in the software industry, under the impulse of several forces: software applications have become more pervasive, distributed, and multi-platform than ever, which has called for approaches capable of factoring out system knowledge from the source code, for greater reuse; the scarcity of skilled developers in front of an increasing demand has prompted for a quantum leap in productivity; last but not least, the industry has started converging around a few mature and agreed upon standards (most notably, OMG MDA and recently BPMN) favoring the creation of solid businesses centered on MDE product and services.

However, MDE in the software industry is still far from meeting the goal of supporting development across all the tiers of the application and all the phases of development. Code generation from software models is taking place, but is confined only to those aspects of the system where the model is very close to the implementation (e.g., generating SOA technical artifacts from the signature of a procedure) or where the semantics of the model to code transformation is well understood (e.g., transforming a UML class diagram into class skeletons, or an ER diagram into the SQL code for schema creation).

The end-to-end generation of the application code of a general-purpose application at the same level of quality of a hand-crafted solution, from the user interface to data, business logic, and service connection logic is still considered a very hard task, due to the excessive complexity and cost of

creating models detailed enough to be usable in practice for generating the entire application.[1] However, limiting the generation of code from models only to one tier of the application (e.g., the data tier) or to a partial view of the software (e.g., the interfaces of classes) greatly reduces the potential benefits of MDE. The manual implementation of the non-modeled parts of the application occurs independently of the model, which creates the well-known problems of model-to-code alignment; tracing changes from the model to the code and from the code to the models is overly complex, and breaks most of the benefits of models during implementation and maintenance [Sel03].

This paper reports the experience of WebRatio [ABB+08], a company in the MDE tool market, in facing the challenges of deploying MDE solutions in the industry, with a focus on the model-driven design of user interaction and on code generation across all the tiers of Web/SOA applications. The paper is organized as follows: Section 2 provides an overview of WebRatio and of its accompanying DSL for Web application design (WebML [CFB+02]); Section 3 overviews the parallel evolution of the WebML language and of the WebRatio development environment; Section 4 discusses the lessons learnt from the joint design of the DSL and of its support tool; Section 5 presents a sample of customer histories and reports some quantitative measures on the WebRatio usage, together with some statistics on WebML models size and development effort; Section 6 reflects on the success and failure factors for MDE emerged from the WebRatio experience; finally Section 7 draws the conclusions.

2. Model-Driven Engineering with WebRatio and WebRatio BPM

WebRatio is a spinoff company of an academic institution (Politecnico di Milano). It was born during a European cooperative research project in a period of time (1999-2001) when the Web was changing the way in which software was produced. The company produces a tool (called WebRatio itself) for model-driven development of enterprise Web applications [ABB+08].

When, in the early 2000's, companies faced the challenge of deploying an

[1] Life- or mission-critical applications are obviously an exception; for these systems, the importance of verification justifies the investment in models accurate enough for complete code generation. In the sequel, we refer to non life/mission-critical applications, where the use of models should meet cost-effectiveness criteria not driven by the unacceptable costs of detecting errors after deployment.

entirely new class of applications (B2C web sites), the motivation of WebRatio was changing the way in which people developed Web applications, by creating a simple model capable of expressing any kind of Web applications (B2C, B2B, B2E) and a code generator powerful enough to produce from the model the same code that a skilled Web programmer would have written by hand. At the time, this idea was considered quite eccentric and against the market trend, which was seeing the birth and growth of large monolithic Web do-it-all products, which the users had to customize to their requirements.

2.1. The Vision

The vision of the MDE tool WebRatio is to enable fast and cost-effective delivery of custom enterprise solutions by means of development tools that integrate *application modeling* and *component-based development*.

Application modeling refers to the capacity of expressing the functional and non-functional requirements of the application and of the architecture design in a simple yet precise manner, which is amenable for both communication with stakeholders, granting early validation and less re-design cycles, and for automatic code generation across all architecture tiers, so to ensure model-to-code consistency along the development lifecycle.

Component-based development refers to the capacity of the model and of the transformations (model-to-model and model-to-code) to reflect and reuse any artifact that is deemed valuable by the customers to preserve pre-existing IT investment. Examples of artifacts that must be reused include: database schemas and content, legacy systems, business components, specific software functions at any tier (from database stored procedures to JavaScript functions) interface widgets and style templates.

2.2. The Development Process and Roles

Figure 1 illustrates, using the BPMN notation, the application development process and highlights the tasks supported by the WebRatio BPM and WebRatio tools.

In the requirement analysis phase, the business stakeholders and the software analysts jointly collect data, functional, and non functional requirements. Non-functional requirement incorporate the user interface layout and look&feel, which has a prominent role in Web development; such requirements are collected in the form of mock-ups, produced by a graphic designer

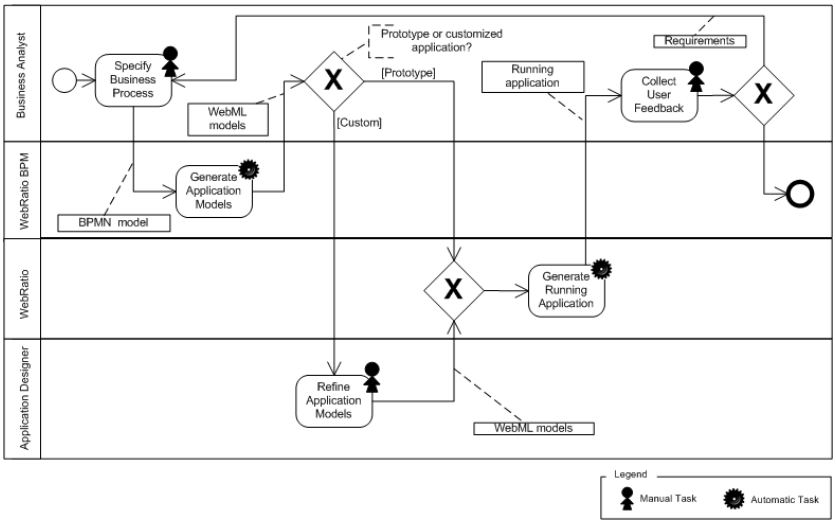

Figure 1: The development process with WebRatio and WebRatio BPM.

and expressed in a Web compatible format, typically HTML. When the application is process-driven, requirements include the business process modeling phase, expressed in BPMN 2.0.

The design phase maps requirements into the design of the front-end and back-end of the application, the latter addressing the connection with the data tier and with the business services. The application design is expressed in the Web Modeling Language (WebML) [CFB00], a Domain Specific Language for general-purpose Web/SOA applications. In the implementation phase, WebRatio transforms the WebML models into the application code.

Development follows an iterative cycle, with strong emphasis on early prototyping. Prototypes can be produced at two levels.

- *Business process prototypes* are generated directly from the BPMN 2.0 schema of a business process and incorporate the complete control logic of the process, including the user roles, the precedence constraints and task activation logic, a user interface that permits each role to submit the parameters required by human tasks, and a Business Analysis Monitor for the process manager to check the status progress.

- *Application prototypes* are generated from WebML models and graphical mock-ups and, depending on the completeness of the model, represent at a variable degree of precision the application user interface

and its interaction with the back-end data and services. In WebRatio, there is not a technical distinction between a prototype and the final deployed version of the application: when the prototype covers all functional and non-functional requirements, it is used for the delivery.

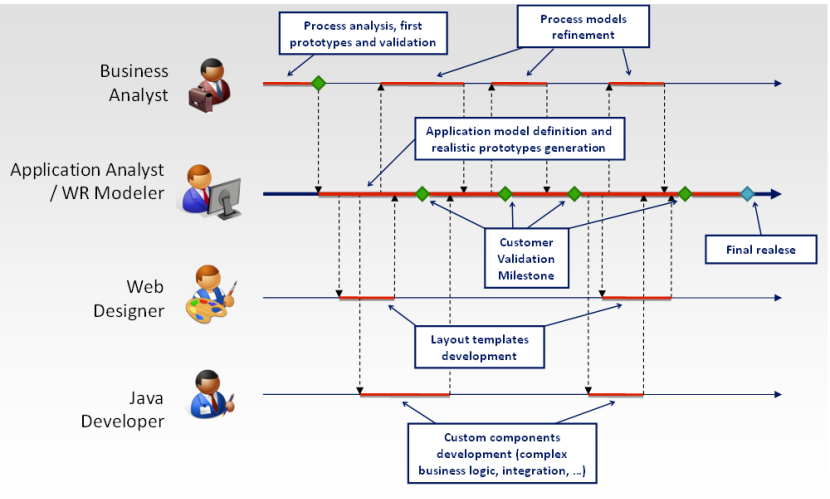

Figure 2: The Development Roles with WebRatio and WebRatio BPM.

Figure 2 shows the roles that participate to the development process, the artifacts they produce, and the typical timing of their engagement in the release cycle of an application.

2.3. The Models and Notations

Application modeling in our approach exploits the Web Modeling Language (WebML), a Web-oriented DSL designed with the initial inspiration of replacing the manual sketches of Web site maps made by designers with an equivalent yet formal notation amenable to code generation. As a consequence, WebML builds upon concepts close to Web developers, such as domain objects, pages, links, data and business logic components.

For the purpose of illustration, we will consider as a running example a simplified Web application consisting of a product catalog, with a Web interface for content publishing, directed to the general public, and an administrative interface for content management, used by the content owners.

Content objects are specified using the Entity-Relationship data model (or, equivalently, UML class diagrams), comprising classes, entities, attributes, single inheritance, binary relationships, and data derivation expressions.

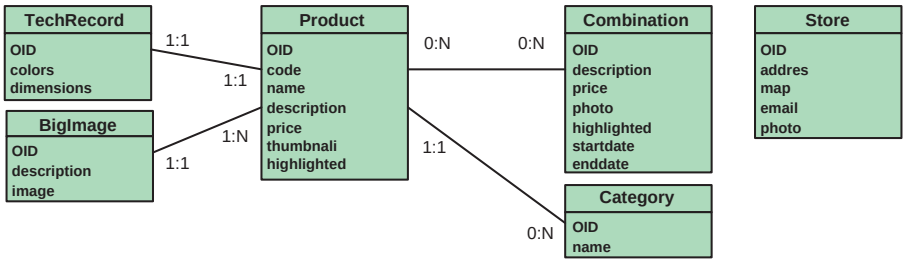

Figure 3: The data model of the running case.

Figure 3 shows the data model of the running example, with the objects mentioned in the requirements and their associations.

The application front-end is specified using the *hypertext model*, which has a hierarchical organization: each application corresponds to a *site view*, internally structured into *areas*, which in turn may contain sub-areas and *pages*.

Multiple site views can be associated to the same data model, e.g., to represent applications delivered to different actors or designed for different access devices (Web, mobile phones, PDAs). Site views, areas and pages are not only units of modularization, but also govern access, which can be selectively granted to each individual module based on the user identity and subscription to groups. A first kind of navigation, which does not depend on page content, can be expressed over site views, areas and pages: if a page or area is marked as "landmark" (L), it is assumed to be reachable (through suitable navigation commands) from all the other areas and pages in the same module; if a page or area is marked as "default" (D), it is assumed to be displayed by default when the enclosing module is accessed; if a page is marked as "home" (H), it is displayed by default when accessing the application.

Figure 4 shows a small excerpt of the hypertext model of the public site view of the running example, with the areas and pages mentioned in the requirements, annotated with the navigation markers.

The public site view contains one top-level area (*ProductArea*) and two top level pages (*Home* and *Stores* Page). Page *Home* is marked as the home page of the site view (H) and is also a globally reachable landmark page (L). The *Stores* page and the *Product* areas are marked as globally reachable (L) too.

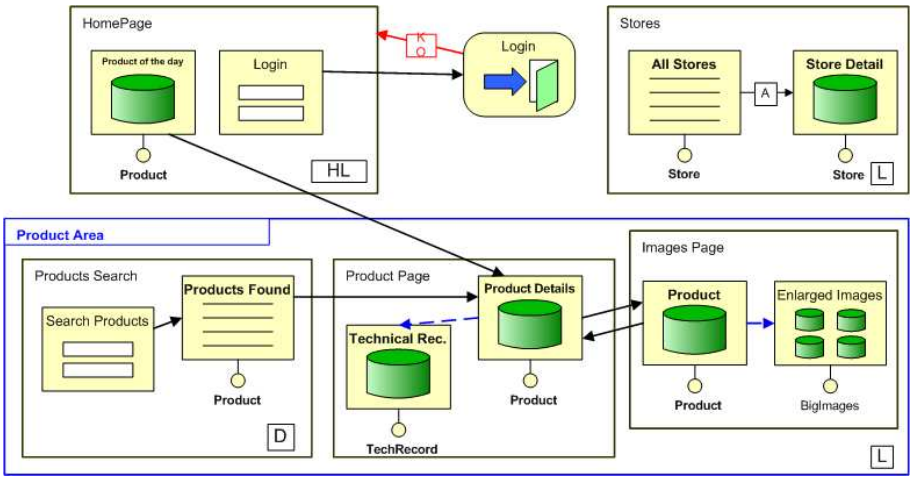

Figure 4: The hypertext model of the public site view.

Pages are the basic interface modules: they can be structured in subpages and comprise content units. A *content unit* is defined as a component that publishes some content in a page; the published content can be extracted dynamically from the objects specified in the data model or specified statically in the hypertext model (e.g., an entry form consisting of multiple input fields). In addition to content units, WebML comprises *operation units*, defined as components for executing arbitrary business logic. Operation units, unlike content units, do not publish content and thus are positioned outside pages.

Components (content and operation units) may have *input and output parameters* (e.g., the OID of the object to display or modify, the username and password for authenticating the user, etc.). Parameter passing is expressed as a side effect of navigation: components are connected by *links*, which have a threefold purpose: enabling the user's navigation, supporting the passage of parameters, and triggering the execution of components. Therefore, a WebML hypertext can be essentially described as a graph of parametric components, connected by links, in which some components publish content and are contained within pages, some other components perform business actions, and are triggered from links emanating from pages. Links express the "wiring" of the application. Five kinds of link are defined: *normal* links, denoted by solid arrows, allow both navigation and parameter passing; *trans-*

port links, denoted by dashed arrows, allow only parameter passing and are not rendered as navigation devices; *automatic* links, denoted by the `[A]` symbol, are normal links automatically "navigated" by the system on page load; OK and KO links are output links of operations, respectively followed after execution success or failure.

In Figure 4, the home page contains one data-driven content unit (*ProductOfTheDay*), displaying the product of the day, and one static content unit, the *LoginForm* entry unit, which comprises two fields for inputting the username and the password; the entry form is connected by a link to a login operation (*Login*) placed outside the home page, for authenticating the employees and forwarding them to the protected content management application (specified as a distinct site view). The outgoing link of the entry unit shows the parameters transferred from the entry form to the login operation unit. The *Stores* page contains two other content units: the *All Stores* index unit shows a selectable list of stores and the *Store Detail* data unit displays data on a chosen store. Interaction is expressed by the link connecting the two units, which permits the uses to choose the store to display. Since the link is tagged as automatic ([A]), on page load one default store is displayed in the *Store Detail* component.

The rest of the hypertext in Figure 4 has a similar organization, with pages comprising several kinds of content units connected by links.

2.4. The Tool and Architecture

Application development with WebML is supported by WebRatio. WebRatio supports all the technical development tasks: declaring the data sources and Web services used in the project, forward and reverse engineering of the data model, presentation specification through page mockups, code generation for the Java Enterprise Edition platform, and application deployment onto a private infrastructure, a public, or a hybrid cloud.

3. Language and Modeling Tool Evolution

The history of WebRatio spans across a decade that has seen a dramatic change in the way software applications are built, which can be summarized in three fundamental factors that impacted the evolution of WebML and WebRatio:

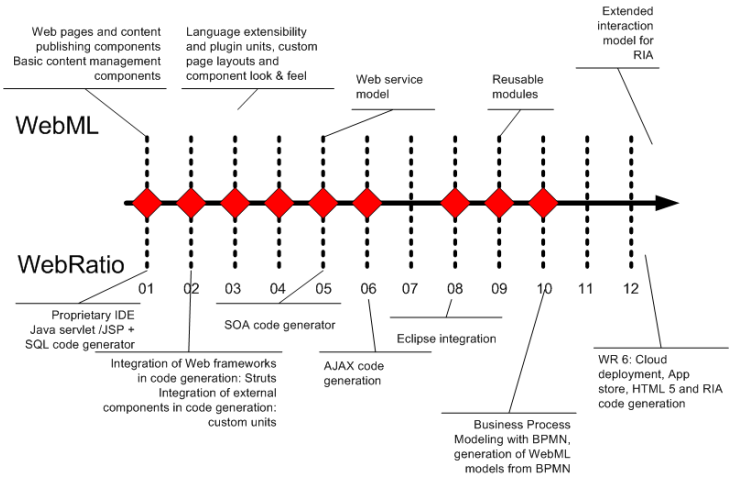

Figure 5: Evolution of WebRatio and WebML

- The progressive consolidation of the Web as an application development demanded the full support and follow-up of the HTTP, HTML and XML, to ensure universal access and application interoperability.

- At the front-end, the multiplication of access devices led to the increase in the number of interaction styles to be supported (from the original point-and click interaction of the Web 1.0, to the current gestural interfaces of tablet and mobile applications).

- At the back-end, Business Process Models emerged as a uniform way of representing cross-organization functionality, and Service Oriented Architecture as the technical vehicle for deploying process enactment on top of heterogeneous IT infrastructures. This trend intersected with the advent of application frameworks, which progressively became an important tool to better support cross-organizational architecture and development uniformity.

These change drivers put much strain on a DSL born for "*capturing the features of the Web*", and produced the timeline shown in Figure 5.

The evolution of the language has been rather intensive in the first years (1999-2001). As noted elsewhere [KP09], it is difficult for language designers to get it right at the first attempt, and practical usage of a DSL is an essential source for improvement. WebML has undergone several major revisions.

The initial version of the language was put to work, even before the spinoff of the company, on a medium scale industrial application, with very stringent time requirements [FCTT06]. This effort validated the core nucleus of the language concepts (especially, the content and operation units), which were profoundly revised to increase expressive power and orthogonality. The lesson learnt at this stage was: *covering all the functional requirements of the application at hand and doing so without a proliferation of constructs.*

The second stage of evolution was marked by the start of commercialization of WebRatio and by the corresponding inception of usage by customers (2001). The key revisions were induced by the need of accommodating domain concepts (mostly business and presentation logic) generated outside the DSL design team. The corresponding major release, which included the notion of plug-in components in the DSL and plug-in rules in the code generator, was probably the most profound revision and took place when an already substantial installation base was deployed. The use of XML to encode WebML projects allowed WebRatio to use standard document transformation technology (first XSLT, then Groovy) to support the migration of very large and complex projects. The Acer B2C and B2B portals and content management applications, described in [FCTT06], were successfully ported to the new version of WebML, and since then hundreds of projects have been maintained over the years across the new releases of the DSL. The lesson learnt at this stage was: *plan for extensibility since the beginning of the DSL design and provide support for automatic and seamless project migration to subsequent language versions.*

The third wave of revision was aimed at better support to enterprise application integration (from 2005). Experience with customers demonstrated that most often the development of a Web front-end was accompanied by the need of migrating or integrating back-end legacy systems, an effort that demanded the capacity of representing the interplay between the user interaction, the data tier and the services that surrounded the legacy or third party applications cooperating to a business process. This shift of focus from pure Web front-end modeling to a hybrid model of the Web front-end and of the SOA architecture backing the user interface was addressed by extending the notion of component, already present in WebML in the form of business functions called synchronously, to the broader case of loosely-coupled services, called synchronously or asynchronously. Here, the lesson learnt was: *do not try to fit in the model a feature that is too far from the initial concept; if necessary represent unmodeled components (e.g., complex SOA orchestra-*

tions) at a high level and refer to them as black boxes.

The last round of change was the long-awaited introduction of a modularization construct (2009), to overcome the model cut&paste behavior observed in large projects, a work-around frequently adopted by modelers to overcome the absence of a linguistic mechanism for defining a reusable piece of model and including it by reference in a larger model. Such an addition was far from trivial, because a reusable module could be anything, from a piece of interface, to a piece of business logic, or more interestingly, a hybrid of the two things, expressing the interface and business logic necessary to perform a function. The lesson learnt was: *as a DSL matures, plan for model reuse and cross-cutting concerns.*

Parallel to the evolution of WebML, also the WebRatio tool suite and the company's market positioning have progressed. The first important improvement was the incorporation of a standard Web framework as the target of code generation (Struts was adopted due to its popularity and stability). This demonstrated to customers the openness of the runtime; if code could be generated for Struts, it could be generated for any other software architecture. A second major revision, which took place in 2007, was the replacement of the code generation technology (from XSLT to Groovy) and the adoption of Eclipse as the base for the IDE. The former choice was motivated by scalability of the code generators. The latter was instead a repositioning w.r.t. code developers: adopting the Eclipse workbench created a seamless environment where modelers and component programmers shared the same technical space. Component developers could create their custom units from within the model-based environment, include them in the project, generate the code and debug the component, all in one place. At the same time, modelers could double-click on any model component and immediately see the code corresponding to the selected concept. This integration, although may not appeal to the purist, favored the adoption of MDE by programmers, who could see WebRatio and MDE in general as a way to organize, and not to replace, their work. This also paved the way to implementing a component reverse-engineering functionality, whereby programmers could easily transform an implementation level artifact into an abstract model construct, under the guidance of a wizard that assisted them in the not-so-familiar task of abstracting the concrete code into a high-level WebML content or operation unit. The development of the Eclipse-based version of the tool has been a major development effort, because it required to re-implement basically from scratch all the design-time structure of the tool. The tool has been

developed on top of Eclipse GEF[2], since at the time the limited stability of EMF-based development prevented the company from investing on that.

The last important line of evolution was the integration of Business Process Management (in 2010), which expanded the reach of WebRatio to the business analysis domain, a sector where models are already well-accepted by companies but generative MDE is still at the infancy, because process models are most frequently interpreted by ad-hoc runtime engines and GUIs for process enactment are created using implementation-level form editors. Here, the challenge was the construction of a consistent MDE approach, based on fast prototyping and full code generation. The solution adopted by WebRatio (illustrated in Figure 6) is to keep the two modeling languages orthogonal and to generate default WebML models from process models. This approach, which is still unique in the market, exploits model-to-model transformations: the BPMN process schema is transformed into a set of WebML application models that represent the user interfaces for the process actors to perform their task.

Process control and user interaction design are kept orthogonal, so that the two models can be evolved independently. Orthogonality is obtained by separating the concerns between: the process design and control (pertaining only to the BPM models), and the user interaction (pertaining to the WebML models). The BPMN model describes the process control and the granularity of the business activities, while the WebML model expresses the detailed design of the internal behaviour of each activity in terms of user interface and back-end logic invocations. The model transformation from BPM to WebML does not embed the process control in the WebML logics; it simply generates the appropriate user interaction containers (placeholders) for implementing each activity in the process. These containers though are not left empty, but are generated already with a set of default pages, page components and operations that depend on the activity type and parameters (e.g., WebML units for updating process flow objects, for sending messages across process pools, for asking the needed user input based on the activity parameters). The WebML model of the activity execution interface also contains the links for suspending/terminating the activity: these links trigger components that inspect suitable process advancement metadata and decide the activity to activate next, based on the BPMN process schema.

[2]http://www.eclipse.org/gef/

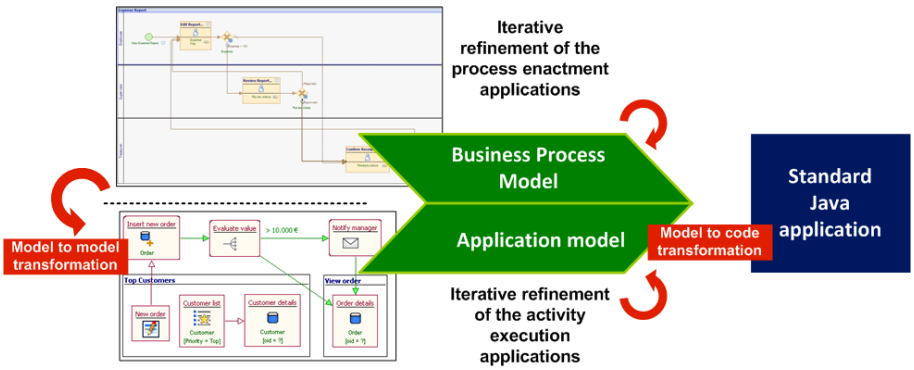

Figure 6: The two stage model-to-model and model-to-code approach of WebRatio BPM

Furthermore, the WebML model comprises standard process configuration and process monitoring pages implementing BAM (Business Activity Monitoring) features, which are independent on the specific process model. The generated default user interactions can already enact the specified business processes or, if needed, can be refined and updated by the WebML designer, without altering the control flow of the process. In this way, the process schema and the Web application interface for executing activities are kept separated and can be evolved independently.

As a last remark on the history of WebML and WebRatio, we underline how Figure 5 also shows a case where the tool evolution anticipated the DSL evolution. This happened with the AJAX code generator, which was delivered before the RIA extension of WebML. The reason is that AJAX was initially perceived as incremental addition to the capability of HTML, mostly for better interface usability, and thus relegated to the role of yet another technology for presentation templates, which are not modeled but used as part of the code generation process. However, it soon become clear that AJAX was part of a more substantial revolution, the unification of Web and desktop applications, that demanded a thorough reconsideration of the front-end modeling language. The results of the study on the WebML RIA extensions are documented in [FCBC10] and the engineering effort for implementing the required changes in the model and in the code generators are expected to produce a major release in 2012.

4. Lessons learned on DSL design

The original principles adopted in the design of WebML put special emphasis on three aspects, which were considered prominent for achieving developers' acceptance: 1) *expressive power*: the model should be capable of expressing Web applications comparable in complexity to industrial-strength systems developed manually; 2) *ease of use*: the model should be easy-to-learn for developers not skilled in software engineering; 3) *implementability*: the model should encompass enough information to permit the generation of code for all the tiers of a dynamic Web application. Code generation should produce optimized code as far as possible. In the following paragraphs we comment on the essential choices taken in the design of the language.

4.1. Model Boundaries.

The approach to defining model boundaries has been rather crude: the model contains the minimal number of concepts necessary to generate code. This led to the partition of the concepts into the three perspectives of the data model, hypertext model, and presentation; the latter perspective gathers all those aspects that have to do with aesthetics and interface usability and is not expressed diagrammatically, but by means of annotated examples incorporated in the code generation process. A notable feature of WebML is the absence of a separated sub-model for the business logic. Server-side business logic is partly encoded in the data model (in the form of declarative specifications of derived data) and partly in the hypertext model (in the form of black-box content and operation units, which can represent components with arbitrary functionality). This choice simplified dramatically the model, to the price of two (not necessarily negative) unforeseen phenomena: 1) the need of an extension mechanism for introducing in the DSL custom units encoding user-defined business logic; 2) the complexity growth of the presentation code generator, which was used to capture, beside aesthetics, also client-side behavior (e.g., JavaScript event handling), not expressible otherwise.

Another essential aspect of model definition is the compact representation and efficient management of inter-model references, which are ubiquitous: the hypertext diagram references the data model (e.g., for data extraction and update), and the presentation refers to the hypertext model (e.g., for content positioning in pages). Figure 7 shows a reference from the hypertext model to the data model: the *AllStores* index unit and the *StoreDetails* data unit

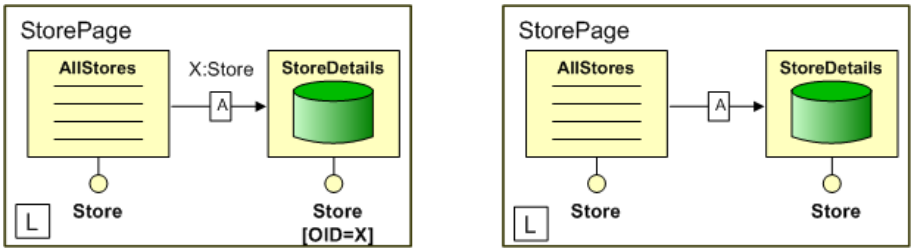

Figure 7: A page with an index unit for selecting a store and a data unit for displaying its details (left). The same page with abbreviated notation (right).

refer to entity *Store*, which provides the components' content. The reference to the data model element is simply realized by means of a typed property in the hypertext model.

Level of Abstraction. A WebML specification is far more abstract than an object-oriented representation of the generated code. A model concept is normally mapped onto multiple software artifacts, possibly residing at different tiers. For example, a WebML page, its unit and links are indeed a compact representation of multiple artifacts: 1) The data extraction code in the data tier (stored procedures, queries, EJB finder methods, etc..); 2) The object(s) storing the content in the business and/or presentation tier (entity EJB, javabeans); 3) The action class decoupling the request from the business tier objects; 4) The business service orchestrating the computation of the page content; 5) The JSP executable tags translating object content into page markup. The decision of keeping a high level of abstraction traded realism for compactness. To the developer's eye the WebML model of a page mixes things that in reality stand apart. However, if the ultimate purpose of modeling is to generate the code, the ease of building up a complete model should prevail on the realism of the representation, provided that the meaning of each abstract concept is known and the behavior of element composition predictable. Considering that a small-size project may typically consist of tens of pages comprising hundreds of components, it is easy to see that raising the level of abstraction, while preserving semantics, is crucial to achieve practical usability. Realism is reintroduced at the tool level: double-clicking on any model element shows the code that has been generated for that component, bringing back the level of detail natural for programmers.

Behavioral Semantics. A quite radical aspect of model design, which

departs WebML from traditional OO modeling, is the absence of a separate behavioral model. Behavioral aspects are embedded in the hypertext model, which can be regarded as a mix of a structural and of a dynamic diagram: on one hand, it represents the composition structure of the front-end (e.g., nesting of pages and containment of components within pages); on the other hand, it expresses event handling (the navigation of links that changes the state of the application).

All WebML models have a fixed operational semantics for representing event handling and state change, which therefore needs not be specified explicitly by the designer application by application. In essence, a hypertext model is equivalent to an extended finite state machine [BCF02], in which events capture user's interactions and system generated events, and transitions describe the propagation of computation from one component to another one. Conditions on transitions express the flow of control, for example the order in which components must be executed depending on the availability of user's input or of system-provided parameters. The provision of a standard, application-independent operational semantics is a cornerstone of the design of WebML: the "in the large" semantics describes the general functioning of the Web application as a network of cooperating components; the "in the small" semantics describes how individual components work. The former is standardized and developers need not address it; the latter is treated as a plugin to the model: developers are requested to understand how the predefined WebML components work and define their own components by respecting a few basic rules requested by the "in the large" semantics (essentially, each component should declare its input and output parameters and optional default rules). This approach resembles the notion of framework at the coding level, raised to a more abstract level. Yet the purist, when faced with a WebML diagram, will always ask: "where is the dynamic model"?

Ergonomics. The success of a model-driven method is tightly connected with its practical usability. If editing the model is more cumbersome than implementing functionality in the code, developers will hardly switch to MDE. The evolution of WebML is characterized by innumerable revisions aimed at optimizing the performance of modeling, whose common principle is "*everything that can be reasonably inferred from the context must not be specified explicitly*". A notable example occurs with parametric components and parameter passing along links. Such a feature is the backbone of hypertext modeling and occurs repeatedly in any application model. To alleviate the developer's task, WebML provides default rules whereby 80% of parameter

passing is inferred from the context. The notation on the right part of Figure 7 exemplifies this idea. The *AllStores* index unit has a single output parameter (`X`) of type (*Store*), the entity underlying the component; the *StoreDetails* data unit has a parametric selection condition (i.e., a query) determining the object to display (`[OID=X]`), which makes use of the parameter associated with the input link: in this way, the data unit shows exactly the object chosen in the index unit. By inference, the model in the left hand side of Figure 7 is equivalent to that in the right hand side of Figure 7, in which the parametric selection condition and the parameter passing on the link are omitted. This simple default rule spares two editing actions per link, which results into a substantial reduction of the model editing effort.

Openness. The last far-reaching design choice concerned the evolution of the model. Being a DSL, WebML was initially conceived as a closed language, comprising all the primitives deemed necessary for building Web applications. This choice soon proved ineffective: the extension of application functionality constantly demanded for language revisions. A major breakthrough was achieved with the definition of a standard model extension mechanism, which transformed WebML into an open component-assembly language. The semantics was revised so that developers could define their own components, by wrapping any existing piece of software, and mix them freely with the built-in WebML units. This required a simple standardization of the component external behavior to make it comply to the (very minimal) requirements of the standard component orchestration semantics; each content and operation unit must simply declare its input and output interfaces, and the rules for inferring parameter passing and content selection conditions. After such revision, the standard component orchestration semantics, coupled to libraries of domain-specific components, proved adequate to the modeling and automatic implementation of applications in the most disparate domains: mobile, digital television, Web service interaction, workflow management, and so on.

5. Success Stories and Quantitative Measures from the WebRatio User Base

5.1. Success Stories

In 12 years, WebRatio has been used by many developers in the industry and academia. The sectors where WebRatio is currently used span from financial and banking to utility, fashion and furniture, public transportation, software integration, and education. In this Section we summarize some of

the most important industrial success stories and the advantages perceived by the customers in various scenarios. Success story reports on various customer cases are available online in the company's Web site.[3] The histories in this subsection do not constitute sufficient statistical proof for generalized claims on productivity, but we think are useful at least as anedoctal evidence on the advantages of the approach.

5.1.1. Acer

The Acer Group is a family of four brands – Acer, Gateway, Packard Bell and eMachines – whose multi-brand strategy allows each brand to offer a diverse set of characteristics that targets different customer segments in the global PC market.

WebRatio has been applied to address two needs: the first one is to quickly and continuously adapt its business processes to the latest enterprise marketing strategies. Acer deployed a BPM-based Web application developed with WebRatio for dealing with the complex (and continuously evolving) international approval procedures for marketing and commercial materials of the enterprise. This application has been exploiting both BPM modeling and WebML modeling. The online application is able to support a huge traffic rate (11+ million visits per month).

The second need is to provide high quality B2B and B2C Web based customer services. Acer created with WebRatio a B2B and B2C website with CMS and support in 12 languages (the current version is visible at http://www.acer.com). The first version of the application was developed in just 22 weeks from the first brainstorming to the final deploy. A detailed report on this experience is available in [ABB+07], which quantitatively highlights also the advantages in terms of effort and cost of maintenance.

Estimates and assessments after 10 years of experiences with WebRatio show that with respect to the previously adopted tools and methods (non MDE-based) the productivity indexes are three times higher. The development process and the deployed system are currently controlled and managed entirely within the company by a very small group of people able to handle complex applications and easily deal with every request for changes. With respect to non-MDE techniques used in other parts of the IT infrastructure and with respect to externalized maintenance and evolution activities, the

[3] http://www.webratio.com.

maintenance cost is reduced by 80%.

5.1.2. GTT

GTT (Turin Transportation Group) is a public company owned by the City of Turin in Italy. It provides local public transport and more in general mobility management serving 190 million passengers every year.

WebRatio has been used for developing a new e-ticketing system (available at http://ecommerce.gtt.to.it and serving 64,000 daily passengers) published on-line in only 2 months. The application comprises 100 page templates (WebML pages) and 1215 WebML units. The iterative and quick prototyping approach supported by WebRatio allowed GTT to implement a virtuous cycle of development and testing, and to make a web solution tailored to the company needs with limited cost and time-to-market.

GTT customers can log-in in the reserved area by typing the microchip code printed on their personal card and they can buy the pass with their credit card, can manage their own personal data, check pass validity and browse an archive page containing information about their subscription history.

5.1.3. A2A

A2A is a multi-utility company buying and selling wholesale electric power. The management of electric power was previously handled by traders who relied excessively on individual productivity tools and/or poorly integrated local functional applications.

WebRatio has been used for developing the Integrated Energy Management System that replaced individual productivity tools used by traders for the management of electric power. The quick prototyping approach offered by WebRatio allowed the involvement of actual users in the development process. A2A produced an application able to manage the lifecycle of energy contracts only 6 months after the initial meeting with WebRatio, with a time to market that took one-third of the time estimated in case of adoption of a traditional development.

The essential functions of the application are: the management of electric power contracts; the management of "green package" contracts; the management of natural gas contracts; the management of the variable rate curve of both electric power contracts and gas contracts; the management of market indices; and the exporting/importing of ad-hoc statistical reports from different systems.

5.2. Quantitative Measures

This Section presents some quantitative information on the projects size and effort, at the purpose of providing some intuition on the scale of the addressed projects, on the size of the corresponding models and on the number of modeling artifacts used in the modeling, and on the way developers use WebRatio and WebML in their activities.

5.2.1. Industrial Projects

In this section we report some measures of 14 typical industrial projects developed using WebRatio. Projects have been chosen in a way that allows to appreciate the diversity in size and required effort, as described below. Our aim here is not to assess statistically valid measures of performance for the approach, but instead to give some hints on the experiences achieved in these years. Notice that project size is measured in terms of WebML pages and WebML units, only roughly corresponding to the traditional concepts of page template and widgets, because the MDD approach, the highly reuse-oriented WebML primitives (modules, master pages, and so on), and the tendency towards the one-page-application paradigm can make a single WebML page correspond to several templates developed by hand.

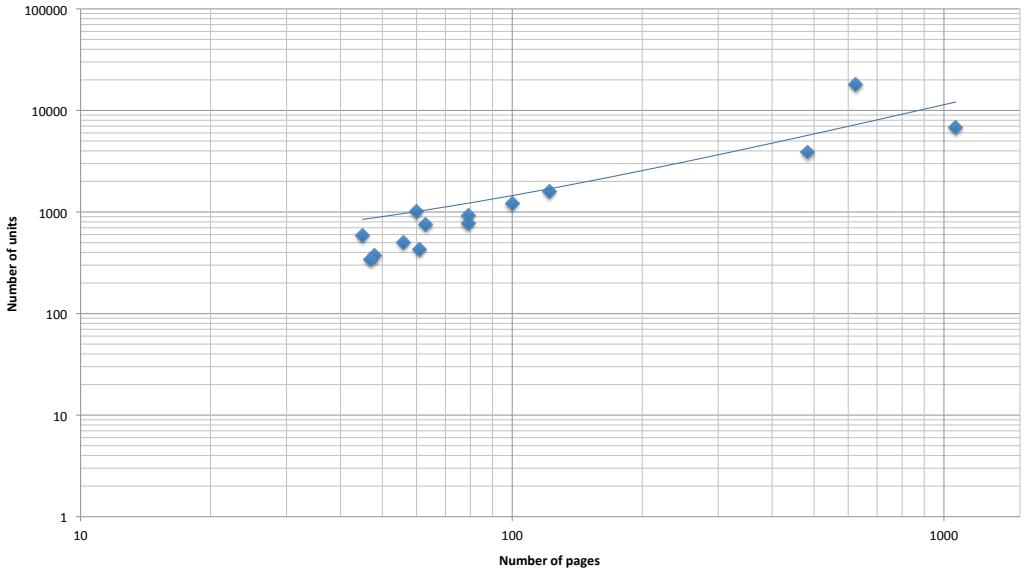

Figure 8: Relation between the number of WebML pages and number of units in a set of 14 industrial projects (logarithmic scales).

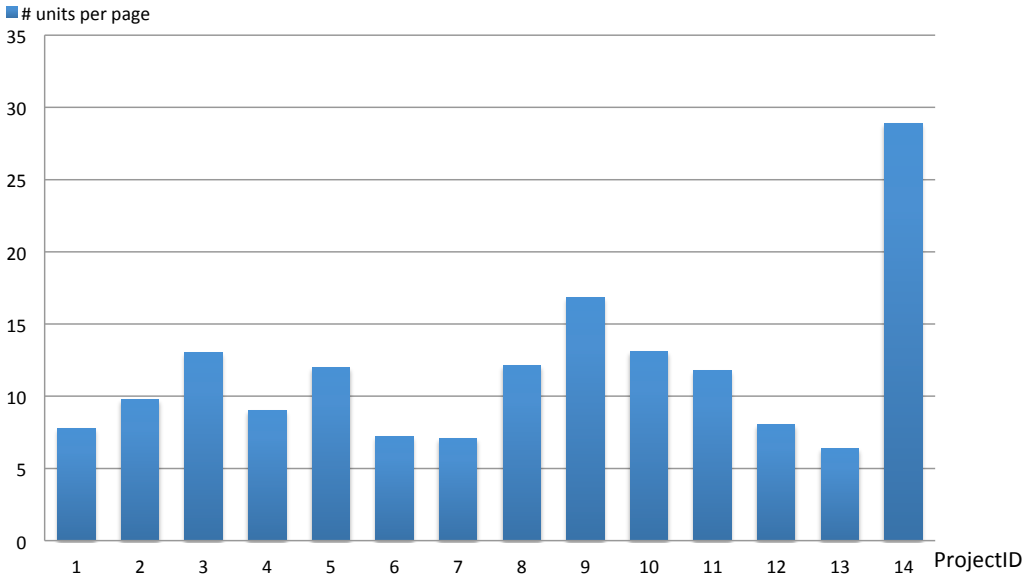

Figure 9: Average number of WebML units per page in a set of 14 mid- to large- scale industrial projects considered in our analysis.

To give a hint on the size of typical industrial WebML projects, we graphically represent some dimensions: Figure 8 shows the relation between the number of WebML pages and of WebML units in each project; Figure 9 shows the average number of WebML units per page in each project. By looking at the diagrams one can see that the two sizes are basically proportional, although variations with respect to the linear growth are significant. The number of units per pages varies a lot depending on the application: this is basically due to the high diversity in the UI complexity in the different applications. This represents one of the major difficulty for correct estimation of effort, as we will discuss later.

Figure 10 shows how the cost of development of modeling items (namely, WebML units and pages) varies depending on the overall effort dedicated to the project, measured in man-days (after removing the outliers). As one can notice, there is no direct correlation between the total effort in a project (which is also a rough measure of its size and complexity) and the average effort put in the development of each page or unit. On one side, projects with very similar (or even identical) size may encompass very different efforts per page or per units, while projects whose size is significantly different may

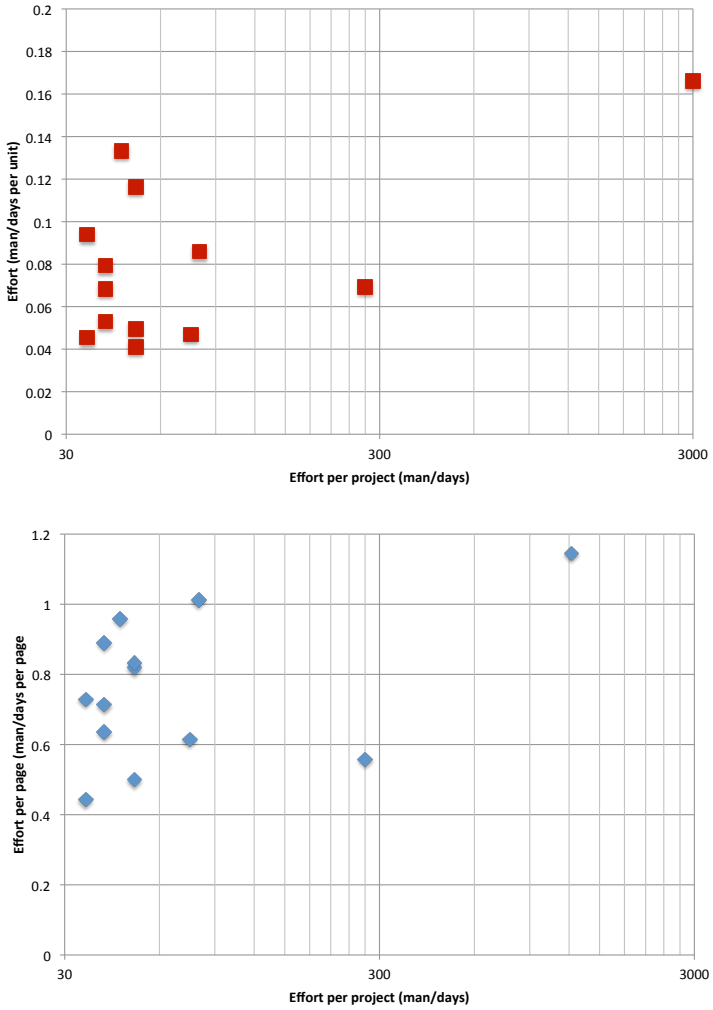

Figure 10: Cost of pages and units in terms of man-days, depending on the overall effort associated to a project.

require similar efforts per units or per page.

However, an important evidence is that the cost of each page always oscillates between half and one man/day, while the cost of a unit is between 0.04 and 0.14 man/days.

Generally speaking, our experience shows that the high variability in requirements undermines the possibility of extracting numerically significant trends in productivity and sizes. One of the main reasons of that is the weight

of the graphical interface design, which strongly affects the overall effort dedicated to a project and therefore extremely complex applications with simple interfaces may result in higher productivity measures than simpler applications that however require complex graphics and interaction widgets.

Despite this, the WebRatio company is now able to estimate the effort of new projects with remarkable precision (error in the order of 10-20%, significantly lower than typical estimates done in the software engineering field). This is obtained also thanks to the detailed knowledge and experience of the analysts that are able to assess the complexity of the expected graphics and user interfaces with good precision.

The precise measure of the cost of pages and units contributes to this results, as our estimates can now rely on a gross cost of 1 man/day per page and/or 0.1 man/day per unit. The expertise of the analysts stands in understanding, for each core requirement, the number of pages to be implemented, and their complexity (in terms of units). This allows to quickly estimate the total cost, thanks to the effort values above.

5.2.2. Usage Data

To complete our overview on the WebRatio tool usage, we provide also some information on how developers exploit the tools features during their work. This analysis is possible thanks to the fact that the tool lets the developer decide if they want to automatically share the usage statistics with the tool vendor through the internet.

For privacy reasons, only coarse grained information is collected. However, we think they provide interesting insights on what is more useful for the development.

The period of time we report on is from May 2010 to February 2012. The users that agreed to send their usage data are 7,864. Among them, we selected a subset of users that used the tool in at least 5 different working days during the analysis period, to remove the bias of people that only opened WebRatio a few times after they downloaded it for evaluation purposes. This reduced the set of the analysis to 1,498 users. The resulting statistics are reported in Table 1. Numbers show that people are using intensively the code generation feature (either for generating excerpts or entire projects), while they use much less the one-click complete generation and deployment procedure. Also some additional features such as warning checking and documentation generation are less used. While the latter could be expected, because documentation is not something one generates at every step of the

Description	Value
# of times the tool has been opened daily	1.79
# of daily code generations	11.76
# of 1-click generation and publishing of the application	0.26
# of checks of the modelling warnings	2.09
# of checks of graphical layout warning	0.11
# of automatic generations of the documentation	0.02

Table 1: Average daily values of usage statistics for the WebRatio MDE tool.

modeling, the very limited use of warning checking is somehow surprising. This may either mean that users are experts of the WebML language, or that they feel it as simple and thus do not sense the need for going through the warning analysis, or they rely on the code generation and they tend to identify errors while executing the generated application. This also entails the fact that the way or granularity in which warnings are reported is not deemed useful or productive.

6. Influential Factors and Issues in Deploying MDE in the Industry

In this Section we summarize the factors that have been most important in achieving the acceptance of MDE by customers, considering both technical and commercial aspects.

6.1. Technical Factors

The technical factors that we deem crucial in achieving the acceptance of our MDE approach by customers are:

- *Extensibility of the model*: given the nature and goal of WebML, i.e., representing *any* Web/SOA application with domain specific concepts, this aspect has been a key enabler. The main trade-off to face was between the excessive proliferation of built-in concepts, which would have made the language difficult to learn, and the over-generalization, which would have led to a lack domain expressivity similar to that of general purpose OO models. The solution we pursued was to specify *as early as possible* the base semantic rules (a kind of common denominator) of domain components, and then let adopters expand the language with their own concepts respecting such rules.

- *Quality of the generated code*: generative MDE faces a barrier of mistrust in the quality of the generated code, which is inevitably perceived as less performing than the highly optimized code that developers can write using all the tricks of the trade. This obstacle was overcome in two ways. First, the code generation used as targets two of the most popular and appreciated software frameworks: Struts for the presentation tier and Hibernate for the data tier. This choice reduced the Not Invented Here objections, to the price of a narrowing of the customer base to the JEE market. Second, not only the model but also the code generator was re-engineered to become open and modular. In essence, today WebRatio has no hard-wired code generation logic. All the rules in the model-to-code transformations can be replaced, which permits developers to incorporate their optimizations in the code generator and not in the generated application. As a side effect, optimizations become part of the MDE environment and thus are more reusable across projects and teams.

- *Capturing the user interface*: being able to reproduce *exactly* any user experience required by customers was another mandatory feature. Given the sheer variety of graphical and layout properties that characterize the look & feel of the user interface, it was immediately apparent that modeling the presentation aspects at an abstract level was impossible. Therefore, the approach was to delegate presentation to the code generator and implement *template-based model-to-code transformation*. With this technique, the model-to-code generator is modularized in separate generation layers: one for the data extraction logic, one for the business logic, and one for the presentation logic. The presentation generator consists of *template rules*, i.e., examples of presentations (at multiple granularity levels: page, content component, down to the individual input field or data value) marked-up with placeholders for dynamic code injection. This capability enabled WebRatio to win the acceptance of reluctant customers by grabbing their presentation style directly from their own web sites and building in front of them a custom application dressed with such a presentation.

- *Usability of the modeling language and tool*: as already explained, a great effort was devoted to equip the DSL with inference rules reducing the burden of model editing. This was not enough, though. After a

period of usage, we noticed that customers started creating projects far more complex and larger than we expected, which put under strain the usability of the model diagram and of the editor. Apparently trivial issues like propagating in real time to the hypertext model changes in the data model became major bottlenecks and required a thorough re-engineering of the tool. Also, WebML diagrams were conceived to fit completely in one screen, a habit that customers appreciated as a way to keep the global structure of the front-end always in view. However, with diagrams consisting of hundreds of pages and components, many usability measures became mandatory, including: fast element search, commands to jump from one concept to a logically related one, context-sensitive model checking, and selective code generation.

- *Maintenance support at the model level*: the deployment of MDE to large teams and long-lived projects must face the comparison with all the facilities that code-based development offers off-the shelf to support the maintenance phase. Among the mandatory features model-based versioning, teamwork and change impact analysis were prominent. Change impact analysis must be able to traverse the abstraction layers of MDE. In WebRatio, it never happens that developers modify the generated code, and thus we had not the problem of tracing changes from the code to the model. But organizations do update the schema of their databases after an application is deployed , and thus we had to introduce functionality for tracing a change in the data format to the content model and from the content model to the hypertext model.

- *Delivery support*: prototype MDE tools that provide coverage of the development but leave the developer alone in the deployment phase face dramatic reduction in adoption. Automation, configuration and continuous deployment on different platforms (locally for testing purposes, on enterprise application servers, and on the cloud) are considered relevant benefits by developers.

6.2. Non-technical Factors

The other factors that do not strictly depend on technical features, but that have been raised either from the customer base or by the sales personnel and business partners that try to sell the product to customers include:

- *Considering the attitude of IT staff towards MDE:* developers are typically skeptical because they see the risk of being limited in their freedom and of finding sooner or later some limits in the approach that would prevent them to cover detailed requirements; on the other hand, analysts prefer to adopt their less formalized approaches to requirements and design specification and are typically worried that a detailed model design can require them a much higher effort.

- *Identifying the right role to discuss with:* the main difficulty has been to identify the right role in the customer company to speak to. This may be a general problem for software vendors, but we think it's particularly critical for model-driven design tools: especially for large projects , the people in charge of the purchase decision power typically do not have the competencies and the capability for judging the quality and real impact of the tools, and thus rely on IT staff for an opinion. In turn, for IT people the attitude problem may play a role in the comments they provide.

- *Avoiding the vendor lock-in syndrome:* one of the main concerns of customers is the risk of getting locked into a set of technologies and models that are not standard and/or mainstream. This is perceived as highly dangerous both for the enterprise and for personal reasons, because the people taking a decision like this are worried about possibility of being considered liable for a decision taken without considering the impact on the company. To avoid this risk, WebRatio has lead a standardization initiative regarding the WebML language within the Object Management Group (OMG)[4]. This action is resulting in standardizing the Interaction Flow Modeling Language (IFML) notation, which directly descends from WebML.

- *Identifying the right size for a starting project:* the real challenge is to have the opportunity of making the customer able to perceive and experiment the advantages of MDE on real projects in some real application scenario. For achieving that, it is crucial to identify a good, limited application scenario where to put MDE at work. This first scenario should neither aim at disrupting the activities of the company,

[4]http://www.omg.org

nor being considered too much collateral. The typical size is small and it usually consists of a small extension upon a large-scale running application that the customer has already being using for a while. This allows the customer to have a clear estimate on the cost of the extension, and hence appreciate the advantage of the new approach. After a step like this, the customer is typically more open to adopting the approach to address more challenging or bigger projects.

- *Motivating the company and the developers to address the learning curve:* since most of the IT staff do not have expertise and knowledge on MDE. Therefore, the learning curve of the approach and of the specific modeling language is usually rather challenging and expensive. Indeed, one should consider that working with MDE implies not only learning new techniques and technologies, but also adopting a new way of working and addressing problems.

In this Section we discuss some issues that we felt as controversial during the development of WebML and WebRatio, and report on the way in which we addressed them.

6.3. General Purpose or Domain Specific?

One of the core decisions of any MDE approach is the choice between the two principal ways of conceiving a modeling language: making it *universal*, so that it can describe a very broad (almost infinite) variety of subjects; or making it *speak the language* of a particular domain, so that it resounds familiar to the average person educated in that area.

The former class of languages goes under the denomination of General Purpose Modeling Languages (GPMLs), of which the most popular one is probably UML. The latter class includes Domain Specific Modeling Languages (DSMLs or DSLs), which are built with a specific application area in mind; their mission is to be natural and easy to use for domain experts, or even for non-experts. Probably, the best known DSLs are HTML, SQL, and MatLab. Even if GPMLs and DSMLs concur to the solution of the same problem, a direct comparison between them must be taken with care, as there are overlaps between the two families.

On their positive side, GPMLs have a broad applicability, widespread adoption, and good portability across tools. The flip side of the coin is that GPMLs cannot be precise in the description or tend to be too verbose.

Trading expressive power for generality has an impact on code generation too. If the language lacks domain-specific focus, code generation can only be partial in real settings and the missing domain-specific semantics must be added "by hand" by the domain expert.

DSMLs have strengths and weaknesses quite symmetrical to GPMLs. DSLs can be used only for the domain they are designed, which also entails having a smaller community of adopters, the absence of a shared standard, or the proprietary nature of the language. The greatest pros are: a more precise semantics (which is a key to achieve so-called total code generation [Ise10]); and the closeness to the domain, which turns into more intuitive and easy to learn concepts and notations and more precise semantics. As an example of the former aspect, Figure 11 compares the representation of a Web page containing an index of products, using a standard UML class diagram and WebML. In WebML, which is natively a DSL for Web applications, some model features can be left implicit in the notation, because they are part of the very definition of the language primitives.[5] These are the reasons that lead us to defining a new DSL for the Web.

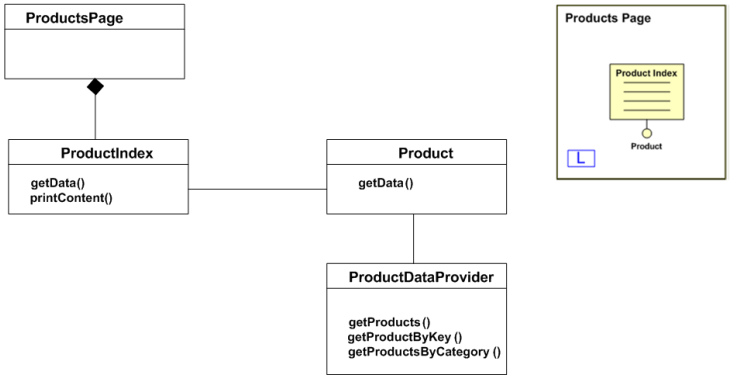

Figure 11: Model of a Web page containing an index of products in a standard UML class diagram and in WebML

[5]The example of Figure 11 refers to the usage of a standard UML class diagram. Obviously, it is possible to use UML extension mechanisms to reproduce exactly the same notation and conciseness of WebML, but this would amount to designing a DSL in UML.

6.4. Would you rather code-generate or model-interpret?

When it comes to turning a model into something executable, two approaches are possible: *code generation* and *model interpretation*. The election of code generation by WebRatio has been quite clear since the beginning, but it has been corroborated through the years also by the interaction with customers and developers, who allowed us to distil the three main reasons for preferring code generation over model interpretation.

Reason 1: customers can choose their execution environment. With model interpretation customers are required to install a new proprietary runtime platform (the interpreter) in their IT infrastructure. This means they have to add a new subject in their IT architecture policies and rely to a proprietary platform for managing non-functional requirements like security and performance. On the other hand, with generated applications are completely separate from the modeling/design environment. Thus one can create and fine-tune a code generator that is able to build applications perfectly fitting a particular runtime architecture. In WebRatio, for example, generated artifacts are standard 3-tier JEE applications, built only by standard Java libraries. In this way, any company already using a Java application server can easily deploy and use them. This allows customers to:

- be compliant to their IT architecture policies.
- choose the best execution environment among a wide range of very popular options instead of being limited to a single platform.
- satisfy security, performance and other non-functional requirements by exploiting the capabilities of a standard execution environment, without having to rely on a proprietary solution.

Eliminating any proprietary runtime has been central in our experience; any organization having strict IT architecture policies, and almost all the public bodies and companies working in the financial sector simply did not consider MDE, if this approach entailed changing their runtime management policies. So we learnt to *keep MDE at design time, and make it disappear at runtime*.

Reason 2: two degrees of freedom are better than one. Code Generation gives you two degrees of freedom: the model and the generator (while with model interpretation you have only the model). Thus, in order to meet all the functional and non-functional requirements of the final solution

you can work both on the model and the generation rules. The second degree of freedom is particularly useful when addressing the layout and the visual identity of the application. Normally these aspects cannot be modeled, so they are coded in some place of the MDD environment. In a model interpretation approach these aspects are managed by the runtime platform (the interpreter) usually by the means of a set of predefined templates and CSS files, while in a code generation approach these aspects are managed by a subset of generation rules. By creating custom generation rules for the presentation layer, customers have many more possibilities to get a final application tailored to their visual requirements.

Reason 3: no vendor lock-in in the deployed application. Although this is an issue every vendor wants to avoid, it may happen that a customer no longer wishes to use that particular MDE environment (or MDE at all). In this case, the only way for a customer to avoid maintenance problems is to have the possibility to open and change any single line of code of its applications without the development environment. This is possible only through a code generation approach (this condition is necessary but not sufficient: the generated code need also to be human-readable) because with model interpretation the customer will always have to deal with the proprietary interpreter. For many customers this is a key factor in the final go / no go decision.

Finally, code generation is not a panacea. To deploy generated applications, we had to face recurrent objections, which lead to a better engineering and more functionality in the WebRatio development environment.

- Code generation is a time-consuming and complex process, especially for big projects, with respect to immediate interpretation: it is not always true, if the MDD environment already includes all the needed rules, generation can be a 1-click task.

- A generated application is more difficult to deploy: it is not always true, if the generated application is for example a standard Java web application, you can immediately deploy it on any standard Java application server (the same would be valid for .NET platform)

- In a Code Generation approach you cannot debug the model: it is not always true, some MDD environments are able to open a socket connection with the application and let you debug the model while

executing the application (see for example the latest release of our MDD environment)

6.5. Is SaaS good for MDE?

One of the challenges that MDE is facing nowadays is the move of the software industry towards cloud-based software solutions and towards software as a service (SaaS) offerings. This is also a good opportunity for MDE vendors: their tools and model execution platforms could be deployed as SaaS and made available with different pricing models to customers (subscription, pay-as-you-go, consumption-based, and so on). On the other hand, MDE tools will be asked more and more to include capability for running and deploying applications on the cloud.

7. Conclusions and Outlook

In this paper we have presented our experience with the MDE tool WebRatio and the associated DSL called WebML. We discusses the basic features of the language and of the tool, and then we delved into a set of lessons learned, customer experiences and usage information.

We think our experience is a significant one in the MDE world because we describe a tool that is completely based on modeling, is oriented to a specific domain, and has been around long enough to collect an interesting set of success stories and insights that can be considered by MDE practitioners when they opt to move towards implementing their own DSL and tools.

Despite the successes, WebRatio still has to face a lot of challenges and ambitious objectives. Among others, we wish to cite three research and industrialization directions that are ongoing.

The first one is *standardization*: at the purpose of tackling the perceived risks of vendor lock-in, we are pursuing a standardization path for WebML. We joined OMG[6] two years ago, and we made OMG issue a request for proposal (RfP) for a user interaction modeling language, and we are now finalizing our standardization proposal in response to the RfP, called IFML (Interaction Flow Modeling Language)[7] and inspired by the WebML language, taken to the broader scope of user interaction description. From a technical and scientific perspective, we feel that our contribution can fit very

[6]Object Management Group: http://www.omg.org .

[7]Interaction Flow Modeling Language (IFML): http://www.ifml.org .

well into the overall OMG vision, as WebML can perfectly integrate with and complement languages well established as international standards such as UML or the domain-specific ones like SOAml, SysML, BPMN and so on (actually, integration with BPMN has been already accomplished both at the language and at the tool implementation levels). The problem of defining a generalized standard for user interaction design is definitely challenging, but we think we will be able to provide some good value on this. Obviously, we do not want the upcoming standard (and the associated standardization process) to be something supported only by ourselves. We want to gather interest on the problem and to collect contributions, ideas, and feedback on the solution as broad as possible. The partnership with OMG is crucial because it will foster the discussion among the big players in the software and modeling market.

Another objective we aim at is towards building a lively and useful *user community* around the tool. At this purpose, we have recently released the WebRatio Store, an online store completely integrated within the toolsuite, where people can download (and also contribute) components, units, graphical styles useful for their work.

Finally, we are facing the big technical challenge of enabling our code generation platform to cover efficient cloud-based deployment of applications. Future plans include also porting our services to the clouds, so as to enable SaaS usage scenarios for WebRatio.

References

[ABB+07] Roberto Acerbis, Aldo Bongio, Marco Brambilla, Massimo Tisi, Stefano Ceri, and Emanuele Tosetti. Developing eBusiness Solutions with a Model Driven Approach: The Case of Acer EMEA. In Luciano Baresi, Piero Fraternali, and Geert-Jan Houben, editors, *Intl. Conf. on Web Engineering (ICWE 2007)*, volume 4607 of *Lecture Notes in Computer Science*, pages 539–544. Springer Berlin / Heidelberg, 2007.

[ABB+08] Roberto Acerbis, Aldo Bongio, Marco Brambilla, Stefano Butti, Stefano Ceri, and Piero Fraternali. Web applications design and development with webml and webratio 5.0. In Richard F. Paige and Bertrand Meyer, editors, *TOOLS (46)*, volume 11 of *Lecture Notes in Business Information Processing*, pages 392–411. Springer, 2008.

[BCF02] Marco Brambilla, Sara Comai, and Piero Fraternali. Hypertext Semantics for Web Applications. In *SEBD*, pages 73–86, 2002.

[CFB00] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. In *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking*, pages 137–157, Amsterdam, The Netherlands, 2000. North-Holland Publishing Co.

[CFB+02] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann, USA, 2002.

[FCBC10] Piero Fraternali, Sara Comai, Alessandro Bozzon, and Giovanni Toffetti Carughi. Engineering rich internet applications with a model-driven approach. *TWEB*, 4(2), 2010.

[FCTT06] Piero Fraternali, Stefano Ceri, Massimo Tisi, and Emanuele Tosetti. Developing ebusiness solutions with a model driven approach,. In *Proc. International Conference on Industrial MArketing and Purchasing*, Milan, Italy, Semptember 2006.

[Ise10] Martijn Iseger. Domain-specific modeling for generative software development. http://www.itarchitect.co.uk/articles/display.asp?id=161, July 2010.

[KP09] S. Kelly and R. Pohjonen. Worst practices for domain-specific modeling. *Software, IEEE*, 26(4):22 –29, july-aug. 2009.

[Sel03] Bran Selic. The pragmatics of model-driven development. *IEEE Software*, 20:19–25, 2003.