Generating Blogs from Product Catalogues: A Model-Driven Approach

Oscar Diaz[a] and Felipe M. Villoria[b]

[a]The Onekin Group, University of the Basque Country
oscar.diaz@ehu.es
[b]Donostia Digital
fmartin@donostiadigital.com

Abstract. Blogs can be used as a conduit for customers opinions, and in so doing, building communities around products. We attempt to realise this vision by building blogs out of product catalogues. Unfortunately, the lack of standards for blog APIs, and the limited experience in virtual communities, make this endeavour risky. This refrains small-and-medium companies from setting such blog-based communities. This paper presents a model-driven approach to alleviate these drawbacks. To this end, two abstract models are introduced: *the catalogue model*, based on the standard *Open Catalog Format*, and *the blog model*, that elaborates on the use of blogs as conduits for virtual communities. Blog models end up being realised through blog engines. Specifically, we focus on two popular platforms: a hosted and a standalone blog platform, both in *Blojsom*. The paper outlines blog construction as an instance of the MDD process, provides some transformation samples, and concludes by comparing MDD and direct manual coding of blogs.

1 Introduction

Companies know the importance of creating consumer communities around their products: users come together to exchange ideas, review and recommend new products, and even support each other. As stated in [6] *"more than one large company has discovered that external customer communities provide better support to their customers than dedicated tier-II and tier-III customer service representatives*". These communities are frequently supported through blogs. Initially thought as personal diaries, blogs' scope has broaden to become a medium for professionals to communicate [12], leading to the so-called *corporate blogging*. Distinct studies [10, 8] endorse the use of blogs to market products, build stronger relationships with customers, and obtain customer feedback.

Based on these observations, we aim at building a software factory that delivers "catalogue blogs" out of "product catalogues" in a cost-effective way. *The challenge is then not on feasibility but cost effectiveness.* Indeed, the advantages of virtual communities around products are well-known, but so far they are only enjoyed by large companies that can afford the price of ad-hoc development. We do not claim blogs to be the ultimate support for virtual communities but

one with the lowest entry barrier. Reducing the cost of entry, can encourage companies to become aware of the benefits of direct and sustainable customer interaction using blogs. This experience can eventually lead companies to move to more sophisticated blog engines or other solutions (e.g. Customer Relationship Management (CRM) tooling). Unfortunately, the lack of portability among blog engines can lock companies to a specific vendor.

There exists two main stumbling blocks. First, the plethora of blog platforms of our customer base. Blogs can be hosted by dedicated blog hosting services or be available for users to download and install on their own systems. Such diversity prevents experiences from using a given blog engine to be extrapolated to a different engine. The second impediment is the youth of virtual community development. The lack of clear guidelines for building such systems makes perfective maintenance more than likely.

Therefore, our endeavor (i.e. building blogs out of product catalogues in a *cost-effective way*) tackles three concerns: catalogue description, virtual communities and blog engines. Although catalogue description is quite stable (standards exist), this is not the case for neither virtual communities nor blog engines. This jeopardises reuse which, in turn, hinders the fulfilment of the cost-effective requirement.

This situation is addressed by using Model Driven Development (MDD) [14]. MDD achieves reuse by introducing distinct models of a system at different levels of abstraction, and, what is most important in our context, models consolidate design decisions in the sense that changes in lower layers should not affect higher models.

The contribution of this work is then two-fold. From a practitioner viewpoint, it reports an experience on using MDD in a medium-size software organisation. The focus is on providing a general view of the whole MDD process rather than implementation details that would have required more space. From a research perspective, the paper provides, to the best of our knowledge, original insights about supporting virtual communities of consumers through blogs (i.e. *the cataBLOG model*).

The rest of the paper is organised as follows. Section 2 looks at blog construction as a MDD process instance. The (meta)models, i.e. *the catalogue model, the cataBLOG model,* and *Blojsom*, as well as the transformation rules are introduced in sections 3, 4, 5 and 6. Section 7 discusses the approach and conclusions end the paper.

2 Blog construction as an instance of the MDD process

When developing blog-based consumer communities, a large number of functional concerns emerge: what products to include, how they are related, how consumers interact, how information is made persistent, what API to use, etc. Additionally, other non-functional issues might be important: cost (e.g. software license of the blog engine), maintainability, existence of in-house development expertise, etc. MDD provides a way to stratify these concerns by introducing

distinct models of a system at different levels of abstraction [14]. In our case, the system is a "catalogue blog" whose development is layered as follows:

- *the catalogue model*, a Platform Independent Model (PIM), which addresses the following questions: Which products are to be commented upon? Which kind of cross-selling relationships are involved? To this end, this work takes a standard to catalogue definition: *Open Catalog Format* (OCF) [9],
- *the cataBLOG model*, a PIM which faces how user feedback can be captured through blogs, and other virtual-community issues,
- a *hosted blog platform*, a Platform Specific Model (PSM) which raises issues concerning the realisation of the consumer community through a *remote* blog using a specific blog engine: *Blojsom* [4],
- a *standalone blog platform*, also a PSM which realises the community through a *local* blog using different functionality of *Blojsom*.

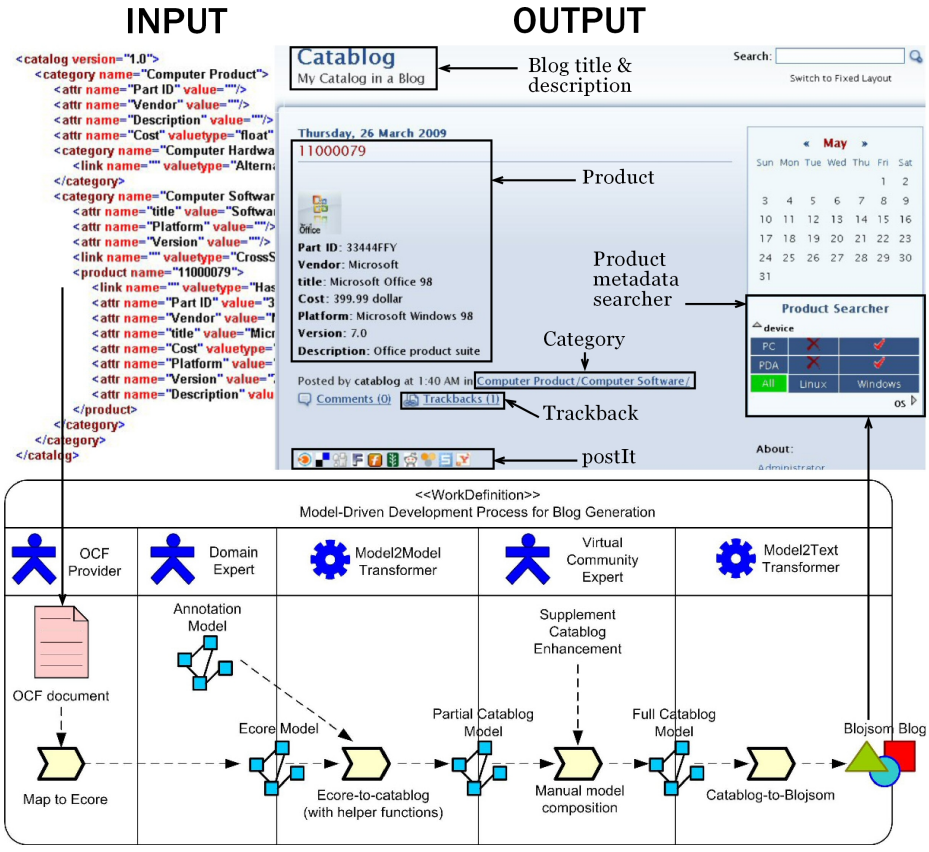

Fig. 1. From *OCF* catalog description to blog generation.

Now, manual coding is substituted by first modeling, next transforming. Model transformation is the process of converting one or more input models (a.k.a. source models) to one output model (a.k.a. the target model) of the same application. Although the ultimate MDD scenario is automatic code generation, our experience is that this seldom happens. Manual intervention is frequently required at two different stages:

- *before the transformation*, the designer frequently needs to intervene to disambiguate the transformation. This is known as *"annotation"*. Annotations are not about the source model as such but on how to transform the source model. Hence, it is possible to have different outputs for the very same input model based on indications on how the transformation should proceed,
- *after the transformation*, the designer often needs to complete the target model with data that can not be generated from the source model. It is not a question of determinism but a lack of information about the target model.

Therefore, the following actors are distinguished: *the transformer* (i.e. the software that maps from source model to a target model), *the domain expert* (i.e. the designer that provides hints about how the transformation should proceed in case of ambiguity), and *the virtual community expert* (i.e. the designer that supplements the generated model with additional details about the domain at hand).

Figure 1 provides an example of an *OCF* description and its *Blojsom* counterpart. The *OCF-to-Blojsom* process is depicted in Figure 1 as a *SPEM* process [11]. First, the XML description of *OCF* is mapped to *Ecore* (i.e. the Eclipse-based realization of MOF) that can now be processed as a model. This model is then enriched by a *domain expert* that provides some hints about how to render this catalog as a blog. On these grounds, the *Ecore-to-cataBLOG* transformation outputs a partial *cataBLOG* model which basically holds product content. This model needs to be supplemented with navigation and presentation concerns regarding virtual-community support. This rises a problem similar to "protected regions" for PSM, where experience advices to use model-composition techniques rather than inlaying this data directly into the source code. Likewise, additions to a generated PIM are preferable to be provided as a separate *"supplement model"* that is latter composed with the generated target model. This information is provided through a separated model by a *virtual community expert*. Both models can now be composed together to come up with a *Full cataBLOG Model* ready to be transformed into the selected PSM (e.g. *Blojsom*). Unfortunately, we did not manage to keep the supplement model independent of generation-dependent data (e.g. product IDs), so we finally decide to do the composition manually.

Figure 1 serves as a roadmap for the next sections. Space limitations make us focus on the most representative parts of the process.

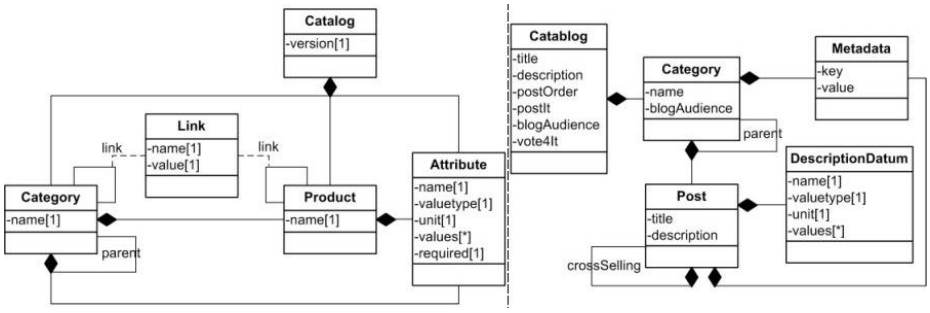

Fig. 2. *OCF* elements (left-hand side) and *cataBLOG* model (right-hand side)

3 The catalogue model

The catalogue model is based on the Open Catalog Format (OCF). OCF is an open standard for describing product catalogs [9][1]. This work takes OCF version 1.0. Figure 2 describes the main notions behind OCF: **catalog**, which consists of a hierarchy of product categories; **category**, which contains attributes, parameters, links, products and subcategories. Each category has a name. A category defines a set of parameters which specify certain special information about the category. A category also defines a set of links (see below). Categories are arranged along parent-child relationships where a child category inherits all the characterisation of its parent category; **product**, which belongs to a category and defines values for attributes of this category. Besides those attributes, a product can have attributes on its own; **attributes**, which describes a property of a product; **link**, which indicates the existence of an association between either categories or products. Links have a name that describe the nature of the link. For instance, two categories can be *"alternative"* whereas two products can hold a *"compound"* or *"cross-selling"* relationship between them. Since catalogue information is natively provided as an XML file, it needs to be first converted to Ecore.

4 The cataBLOG model

Web design methods promote a separation of concerns among content, navigation and presentation models [3]. A blog is a Web application. Hence, it could be drawn that blogs should be described along these three different models. However, in our opinion, blog simplicity makes too convoluted the use of three separate models. Blogs restrict the freedom available for presenting and navigating, namely, (1) content should be arranged in terms of *"post"*, (2) navigation is limited to category and chronology-based search, and (3) presentation is mainly

[1] OCF is readily transformable to other major catalog standards like *xCBL, Punchout, xCBL, CIF, CUP, cXML, OCI* or *Rosettanet*.

template based. Of course, more sophisticated forms of presentation and navigation could be envisaged (e.g. using some configuration or facet-based search for product indexing) but, then, we would have moved away from the realm of simple blogs.

Therefore, our proposal is to use a single model: the **cataBLOG model**. A *cataBLOG model* stands for the essence of what a blog is. It is an attempt to abstract away from the peculiarities of the myriad of blog engines currently available. Additionally, this model is enriched with properties that capture virtual community concerns. Next subsections delve into the details.

4.1 cataBLOG as an abstraction of blogs

A *cataBLOG* model captures *the content model* through the following construct[2]: *cataBLOG, post, category* (which serve to structure the content of the blog), *metadata* (that permits specific processing for posts, e.g. indexing or querying) and *descriptionDatum* (i.e. a description of each of the product attributes that become part of the content of the post[3]). Associations can be defined between categories as well as among posts. Figure 2 depicts the $<<parent>>$ association used to describe category hierarchies, and the $<<crossSelling>>$ association to reflect the namesake relationship between post (i.e. products). This content model provides the hook for adding navigation and presentation hints.

Navigation concerns. The *"cataBLOG"* construct includes a *"postOrder"* property which indicates whether posts are ordered in descending chronological order (i.e. the newest at the top), ascending chronological (i.e. the oldest at the top), alphabetical (by post title) or by post category in the index page of the blog.

Presentation concerns. Blogs use themes[4] for rendering. Each blog engine has its own collection of themes. This makes *"theme"* a platform specific concern whereas *cataBLOG* is a PIM. Hence, the notion of *"theme"* needs to be abstracted into those concerns that will eventually guide the selection of the theme at transformation time, once the blog engine is selected.

For the purpose of this paper, this concern is *"blogAudience"*. Five values are considered to capture the expected age of potential consumers: *child, teenager, young, adult* and *senior*. This property characterises both the *"cataBLOG"* and the *"category"* elements. The *blogAudience* set at the *cataBLOG* level can be overridden for specific categories whose products are targeted to a different market segment. Notice that it is up to the transformation to map each *blogAudience* value to the concrete theme supported by the blog engine at hand.

This decision rests on the assumption that the age segment is the main criteria for guiding the aesthetics of the blog. However, other concerns include

[2] Notice that comments are left outside this models since they are not generated from the catalogue, but dynamically provided by the end users.

[3] Characteristics such as the *"valuetype"* or *"unit"* are kept from the catalogue description since they can impact the rendering of the post content.

[4] A theme is *"a preset package containing graphical appearance details"*, used to customise the look and feel of the blog.

demographic (e.g. sex, age, education, etc), geographic, attitudinal (e.g. interest in lifelong learning) or behavioral (e.g. product usage rate) data [1]. The important point is that the *cataBLOG* model captures those criteria explicitly through properties: *ageAudience, educationAudience, sexAudience*, etc. And even more important, the impact that these criteria have, better said, the impact that the *combined interaction* of those criteria have in the aesthetic or other aspects of the blog, are captured through *cataBLOG-to-blojsom* transformations. For instance, expressions such as *(ageAudience="adult", educationAudience="BSc", sexAudience="female")* can characterise a market segment with a specific rendering theme. These expressions can be explicitly captured through transformation rules. Therefore, transformation rules embody design criteria about how market segments impact the blog, and in so doing, they are true repositories of design expertise.

4.2 cataBLOG as a virtual community model

Full-fledged virtual communities involve a broad range of mechanisms that are outside the scope of blogs. This subsection restricts itself to those aspects that fall within the blog realm. Specifically, we focus on interest sharing and participation promotion.

Interest sharing. Social bookmarking is becoming a popular practice for sharing interests. Tagging sites such as *del.icio.us, Digg, Fark, Newsvine, Reddit, Simpy* or *Spurl*, permit to share URL-addresable resources. Since products are now realised as blog entries (hence, URL addressable), it is now possible to affix these entries into tagging sites, and in so doing, sharing product information with a wider audience. This is an important difference with other approaches such as CRM products. Every new post in a blog is actually a new page which has a *permalink* (i.e. a permanent URL identifier to a specific post). The favours the indexing of post by *Google* (i.e. potential customers can come across with your product posts through *Google*), and allows for bookmarking using tagging sites.

To this end, the *cataBLOG* model is enlarged with a *"postIt"* property. This property holds a list of the tagging sites to which blog entries can be posted to. Rendering wise, this property is realised as a set of icons at the bottom of each entry (see Figure 1). By clicking on the e.g. *Digg* icon, the content of the entry is published at *Digg*. By clicking on *del.icio.us*, the permalink of the current post is stored at this tagging site. The list of tagging sites is up to the designer where the value *"null"* indicates that posting is not allowed.

Participation promotion. In a blog setting, participation is realized through commenting on a blog post. So far, commenting requires users to access directly the blog. However, users can hold their own blogs, wikis or portals through which they comment about products of your catalogue, and hence should be reachable through the *cataBLOG*.

Trackbacks turn out to be a very useful mechanism to this end. They enable websites communicate via "pings", where each ping informs the blog that the sending site has made a reference to a post on the blog [13]. As a product is

now realised as a post, this product can be *trackbacked* from other websites, i.e. comments on this post can appear outside the blog. Instead of forcing customers to comment only at the *cataBLOG* place, customers can now simply send a "ping" to your *cataBLOG* every time they have something to say about your products without having to leave their company website or personal blog. This wides the scope of the community outside the blog itself, and serves to measure the impact of a product: the larger the trackbacks, the stronger the impact.

However, trackbacks just indicate that there is a reference to the post, but not its intention. Such intention can be expressed through *VoteLink* . This microformat proposes a set of three new values for the *rev* attribute of the *<a>* (hyperlink) tag in HTML. The new values are *"vote-for" "vote-abstain"* or *"vote-against"*, which are mutually exclusive, and represent agreement, abstention, and disagreement, respectively. Now, when a customer describes a product at his own company's website, he can indicate his likes to the product in a machine-understandable way, e.g.

> *This is the customer's website. We have successfully installed Acrobat 2.3, after all difficulties with Taborca 3.2*

The value of the *href* attribute, i.e. the URL of *Acrobat*'s post at the blog, stands for the object of your vote. By sending a trackback to the blog, the vote gets included into the results. Basically, this means a push rather than a pull approach to voting: votes are spread all over the web, and it is the blog itself ("the ballot box") the one that goes to your website to collect the vote (see [7] for supporting *VoteLink*-aware trackbacks). This approach can be extended to account for different perspectives. For instance, software products can be ranked based on configurability, usability, interoperability and the like. These different perspectives of the product can be embodied as distinct values of the *measure* parameter in the URL of the post describing the product (e.g. *http://.../Acrobat2.3.html?measure=interoperability*).

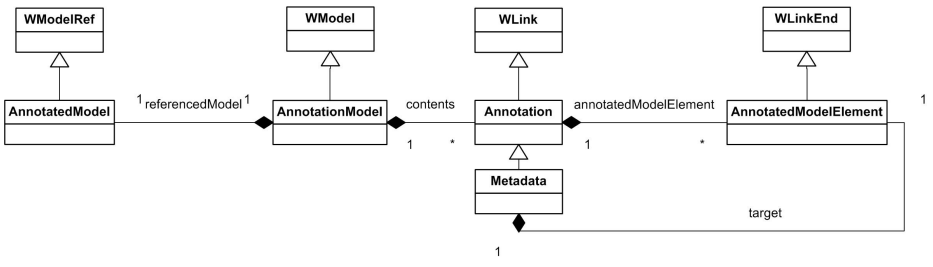

Fig. 3. AWM Metamodel Extension.

This feature is captured in the *cataBLOG* model through the *vote4It* property. This property holds a set of strings that denote each of the criteria to assess the blog products (e.g. *vote4it: [configurability, usability, portability]).* Notice, however, that this extension departs from the *VoteLink* standard, and assumes an agreement between the blog and the voting sites about the qualities to be assessed.

5 *catalog2cataBLOG* transformation

Broadly, the transformation engine takes a catalogue as input, and delivers a blog model. This process needs to be extended with two additional steps, before and after the transformation.

Before the transformation, the *domain expert* provides some hints about how the transformation should proceed. Specifically, *catalogue attributes* serve to obtain the *cataBLOG description.* Additionally, some of them can become *cataBLOG metadata* which are used as an indexing mechanism to recover blog posts (see Figure 1). These decisions (i.e. which attributes will become metadata) are not relevant to the *cataBLOG* model itself. To prevent the catalogue model from being polluted, we use a distinct model to collect these decisions. Along the lines described in [15], a weaving model is used to capture the relationships between elements of the annotated model (i.e. the catalog attributes) and the annotation as such (i.e. becoming metadata). Then, each link in the weaving model represents an annotation for the woven model. ATLAS Model Weaver (AMW) is used for this purpose [5]. Figure 3 shows the weaving metamodel.

Annotations (i.e. the weaving model) can then be consulted during transformation through helper functions. For instance, to know whether a given attribute will become metadata, the helper function in Figure 4 looks for *Metadata* annotations, whose *target* element coincides to the attribute being passed as the parameter (such woven element is identified through the *_xmiID_* property. This helper function is then used by the transformation rule mapping catalogue attributes (see Figure 4).

After the transformation, the designer often needs to complete the target model with data that can not be generated from the source model. Indeed, presentation and navigation properties cannot be obtained from the *catalog* model. Hence, the *virtual community expert* provides a *supplement model* where properties *postOrder, postIt, blogAudience* and *vote4It,* are specified for the blog. Once the *cataBLOG* model is completed, it is ready to be mapped to *Blojsom.*

6 Blojsom as a PSM

The heterogeneity in blog engines firstly motivates this work. A major distinction is that of standalone blogs vs. hosted blogs. Standalone blogs are those where the engine is available for users to download and install on their own systems. In this case, blog creation involves the population of a database or the creation of a folder structure that records the blog entries. *LifeType, b2evolution* and *Blojsom*

```
rule Product {
    from
        cp : Catalog!Product
    to
        bp : Blog!Post (
            title <- cp.name,
            description <- cp.attr,
            metadata <- cp.attr->
                        select(a | not a.getMetadataAnnotationLink().oclIsUndefined())->
                        collect(at | thisModule.MetadataProduct(at)),
            trackback <- cp.link
        )
}
lazy rule MetadataProduct {
    from
        ca : Catalog!Attr
    to
        bm : Blog!Metadata (
            key <- ca.name,
            value <- ca.value
        )
}
helper context Catalog!Attr def: getMetadataAnnotationLink() : AMW!MetadataAnnotation =
    AMW!MetadataAnnotation.allInstances()->asSequence()->
        select(link | link.target.element.ref = self.__xmiID__)->first();
```

Fig. 4. *catalog2cataBLOG* mapping: this ATL rule only applies to catalogue attributes that play the role of blog metadata. The helper function checks this out by consulting the weaving model.

are some examples of this type. Unfortunately, the database schema or the folder structure can differ greatly among engines.

On the other hand, hosted blogs are those where the engine resides remotely, and blog creation is achieved through API calls. *Movable Type, Blogger, LiveJournal* and *Blojsom* are some examples of this type. Unfortunately, current situation is characterised by lack of standards (the same functionality is termed differently), unstable APIs (APIs tend to change) and incomplete APIs (very often the blog engine has to resort to distinct APIs). Even the protocols can differ (e.g. XML-RPC in the case of post creation, whereas REST is used for trackback and comment creation).

This work focuses on *Blojsom*[4]. Even within a single vendor like *Blojsom* different alternatives exists: file-based blogs (*Blojsom 2.x*), database blogs or hosted blogs *(Blojsom 3.x* through *Blogger API 1.0).* Due to space limitations, we focus on file-based standalone blog engines.

File-based blog platforms rely on folders and files to embody blog elements. Figure 5 outlines the file structure for our sample case. The mapping goes as follows:

– **categories** are embodied through system folders which are named after the category name (e.g. *Computer Software* folder stands for the namesake category). The location of the root category (or root folder) is configured by a blog installation property. Subcategories are supported as nesting folders.

/Computer Product/blojsom.properties
/Computer Product/Computer Hardware/
/Computer Product/Computer Software/11000078.html
/Computer Product/Computer Software/11000078.meta
/Computer Product/Computer Software/11000079.html
/Computer Product/Computer Software/11000079.meta
/Computer Product/Computer Software/blojsom.properties
/Computer Product/Computer Software/.trackbacks/11000078.html/1201455548880.meta
/Computer Product/Computer Software/.trackbacks/11000078.html/1201455548880.tb
/Computer Product/Computer Software/.trackbacks/11000079.html/1201455552085.meta
/Computer Product/Computer Software/.trackbacks/11000079.html/1201455552085.tb

Fig. 5. Directory file in *Blojsom*.

- **category metadata** are kept in the so-called *blojsom.properties* file, whose content is a set of *(property,value)* pairs. This file is located inside the folder category. For example, the metadata of the category *Computer Software* could be found out at */Category Software/blojsom.properties*.
- **posts** are supported as text files with *html* extension, whose name is the title of the associated post. The content includes the title of the post (first line), and the description of the post. These files are kept inside the post's category folder counterpart.
- **post metadata** turns out into a *.meta* file, named after the title of the post. In this way, *myPost.meta* holds the metadata of *myPost*, whose content is a set of *(property, value)* pairs. These files are kept inside the post's category folder counterpart,
- **trackbacks** are supported as *.tb* files, named after the trackback creation timestamp.
- t**rackback metadata** are contained in a *.meta* file, named after the corresponding trackback. For instance, *1201455552085.meta* will contain the metadata of the *1201455552085.tb* trackback, whose content is a set of *(property, value)* pairs.

Figure 6 shows a *MOFScript* snippet that realises this model-to-code transformation. For each post the rule *"createPost"* is called and an html-typed namesake file is created. Its content (e.g. title, description) is obtained from the post attributes.

7 Discussion

MDD aims at raising the level of abstraction in application specification and increasing automation in program development. On the grounds of this project, this section reflects on the methodology to achieve abstraction, and the potential Return of Investment (ROI) brought by automation (i.e. code generation).

```
    ec.Post::createPost(categoryName : String) {
        file f (BLOJSOM_BLOG_HOME + blogName + "\\" + categoryPath.get(categoryName) + "\\" + self.title
                                + BLOG_FILE_EXTENSIONS)
        var valueList : String
        var unit : String
f.println(self.title)

        self.description->forEach(ecpd:ec.DescriptionDatum) {
            valueList = ""
            unit = ""
            if (ecpd.valuetype.equals("/image/jpg")) {
                valueList = valueList + '<br /><img src="' + ecpd.value.first().val + '" title="' + ecpd.name + '"/>
            } else {
                valueList = valueList + "<br /><b>" + ecpd.name + ":</b>";

                if (!ecpd.unit.equals("")) {
                  unit = " " + ecpd.unit
                }

                ecpd.value->forEach(ecpdv:String) {
                  valueList = valueList + " " + ecpdv + unit + ","
                }
            }
f.print(valueList.substring(0, valueList.size() - 1))
        }
    }
```

Fig. 6. *MOFScript* snippet that generates the post structure for *Blojsom*.

7.1 On the way to abstraction

Most MDD literature seems to suggest starting from abstract PIMs to then introduce gradually more platform specifics, till, finally, the code can be generated. The structure of this paper reflects this state of mind. However, this project follows the other way around. Our business partner has already a set of blog engines he works with. From then, we abstract into a *cataBLOG* model. This explains why *cataBLOG* is not a general-purpose model for Web applications, but a model that abstracts from *existing* blog engines. Your options are limited to those available through blog engines. This is in contrast to general Web models where navigation and rendering is fully modeled since their PSM platforms can be as general as HTML [2]. This project however, does not take a general-purpose programming language as the target PSM but blog engines. Introducing PIM features which could not be eventually mapped to our targeted set of PSM would have been a waste of time. Our experience then confirms the insights of Markus Volter: "In my experience, it is better to start from the bottom: first define a DSL that resembles your system's software architecture (used to describe applications), and build a generator that automates the grunt work with the implementation technologies" [16].

Another issue is how many abstraction levels are needed. In our case, the existence of the catalogue model provides the upper limit. Catalogues are application-independent resources, already available at the company. Abstracting from blogs to catalogues permit to capitalise on an asset easy to understand for the customer, and available for free. Taking catalogues as the starting point of the generative process was really the decision that makes this experience profitable. Next subsection delves into the details.

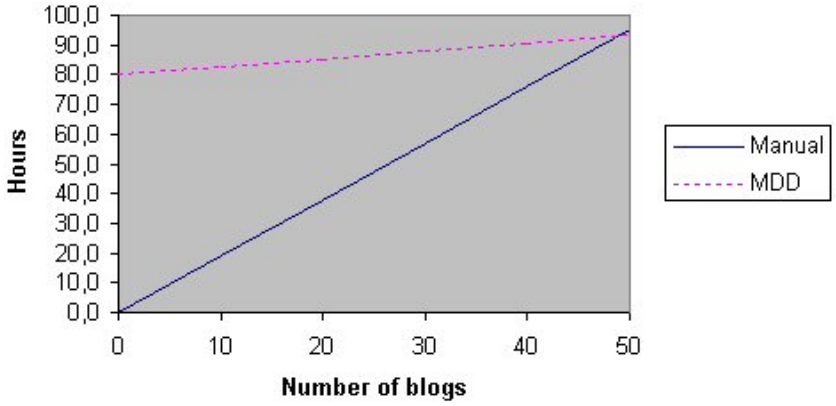

Fig. 7. Breakeven in terms of 50-product blogs: 48,91 *cataBLOG* projects are needed for ROI.

7.2 Measuring ROI

We aim at producing blogs out of product catalogues in a cost-effective way. Although the notion of cost is multifaceted, we focus on labour cost. Other *main* MDD benefits such as quality improvement (i.e. models are easier to validate/verify than raw code) or knowledge retention (i.e. PIMs and transformations as means for knowledge consolidation) are not considered here.

An empirical study was conducted comparing labour hours involved in obtaining the final blog through either manual coding or code generation (i.e.MDD). Figure 7 shows the results for a 50-product catalogue. The different inputs include:

- cost of directly coding in *Blojsom*. This involves coding the products as blog posts one by one. The cost of learning *Blojsom* has not be included as this is an asset already available at the company. Total costs: 1,9 hours using *Blojsom* wizards to create blog defaults.
- cost of generating the code out of the OCF document. The existence of the OCF catalogue is taken it from granted. Therefore, the labour cost is restricted to adding the virtual community properties through the *supplement model*. Total costs: 0,25 hours.
- upfront investment. It includes both the cost of building the MDD infrastructure (i.e. meta-models and transformations), and the learning of the MDD tooling (in our case *ATL*, *AMW* and *MOFScript*). For an experienced and motivated developer, this accounts for 80 hours. The company incurs in this base cost no matter the number of *cataBLOGs* being generated.

The ROI for this upfront cost much depends on two factors: the average size of the product catalogue and the number of *cataBLOG* projects. The catalogue size mainly influences the cost of directly coding in Blojsom. By contrast, it has

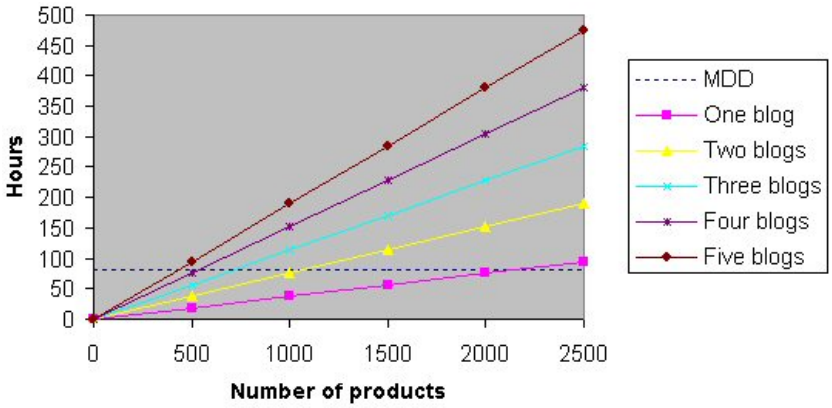

Fig. 8. Breakeven in terms of catalogue size and number of blog projects.

almost no impact on MDD where data is directly obtained from the OCF file. For instance, a sample catalogue including 25 categories and 50 products and an average of 7.2 properties per product, took 1,9 hours to obtain the *cataBLOG* counterpart. By contrast, MDD achieves the same result involving only 0,25 hours of labour work (mainly, the elaboration of the *supplement* model and the *annotation* model). Figure 7 keeps constant this catalog size, and obtains the breakeven as the number of *cataBLOG* projects needed to payoff the upfront investment. These figures show out the role of MDD as a reuse technique where ROI is gradually obtained throughout distinct projects.

Both dimensions (i.e. catalogue size and number of *cataBLOG* projects) are combined in Figure 8. The dotted line stands for using MDD to generate 5 *cataBLOGs*. The slope, almost imperceptible at this scale, goes from 0,25 hours (for 50-product *cataBLOGs*) to 2,35 hours to get five blogs of 2500 products. By contrast, the manual approach has a broader dispersion since all the product data has to be introduced manually as blog properties. It is most important to note that in our case, the MDD approach is specially favorable since it allows to capitalise on an existing asset: the catalog. We do not need to come up with a catalogue model but it is just obtained from the OCF specification. However, other MDD scenarios spend considerable efforts in obtaining these PIMs.

8 Conclusions

Based on the premise that blogs can be a suitable solution to support consumer communities, this work faces the heterogeneity of blog engines and the lack of clear guidelines to support virtual communities. These issues are tackled using MDD. The "felt benefits" include: abstracting away from proprietary blog engines, consolidating through intermediate models (which requires the introduction of the *cataBLOG* model), and capitalising on existing product catalogues.

Future work includes enriching the *cataBLOG* model with additional aspects such as security or permission. Another main issue is evolution. Catalogues evolve, and this evolution percolates their blog counterparts. However, our feeling is that the notion of evolution within model-driven development is not yet understood enough to be used in real applications.

References

1. BPIR.com and Massey University. Customer market segmentation. http://www.bpir.com/customer-market-segmentation-bpir.com/menu-id-72/expert-opinion.html.
2. Stefano Ceri, Piero Fraternali, and Stefano Paraboschi. Data-Driven, One-To-One Web Site Generation for Data-Intensive Applications. In *VLDB*, pages 615–626, 1999.
3. María José Escalona Cuaresma and Nora Koch. Requirements Engineering for Web Applications - A Comparative Study. *Journal of Web Engineering*, 2(3):193–212, 2004.
4. David Czarnecki. Blojsom. http://wiki.blojsom.com/wiki/display/blojsom3/About+blojsom.
5. Marcos Didonet del Fabro, Jean Bézivin, and Patrick Valduriez. Weaving Models with the Eclipse AMW plugin. Eclipse Modeling Symposium, Eclipse Summit Europe, Esslingen, Germany., October 2006.
6. Dion Hinchcliffe. A checkpoint on Web 2.0 in the enterprise, Part 2, August 2007. http://blogs.zdnet.com/Hinchcliffe/?p=135.
7. Steve Ivy. Votelinks + trackback: Voteback?, December 2006. http://redmonk.net/archives/2006/12/20/votelinks-trackback-voteback.
8. Nicholas S. Lockwood and Alan R. Dennis. Exploring the Corporate Blogosphere: A Taxonomy for Research and Practice. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences - HICSS*, 2008.
9. MartSoft, Inc. OCF - Open Catalog Format. http://www.martsoft.com/.
10. Carr N. Lessons in corporate blogging. 2006. Business Week Online.
11. OMG. Software Process Engineering Metamodel Specification (SPEM). Adopted Specification, January 2005.
12. Martin Röll. Distributed KM - Improving Knowledge Workers' Productivity and Organisational Knowledge Sharing with Weblog-based Personal Publishing. *BlogTalk 2.0*, July 2004.
13. Six Apart. TrackBack Technical Specification, 2004. http://www.sixapart.com/pronet/docs/trackback_spec.
14. Thomas Stahl and Markus Voelter. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, 1 edition, May 2006.
15. Juan M. Vara, Mª Valeria de Castro, Marcos Didonet del Fabro, and Esperanza Marcos. Using Weaving Models to automate Model-Driven Web Engineering proposals. In *XIII Jornadas de Ingeniería del Software y Bases de Datos (JISBD/ZOCO 2008), Gijón, Spain, Oct 7-10*, 2008.
16. Markus Völter. MD* Best Practices, December 2008. http://www.voelter.de/data/articles/DSLBestPractices-Website.pdf.