Assisting Blog Publication: Annotation, Model Transformation, and Crossblogging Techniques

Felipe Martín Villoria

Dissertation
presented to
the Department of Computer Languages and Systems of
the University of the Basque Country
in Partial Fulfillment of
the Requirements
for the Degree of

Doctor of Philosophy

The University of the Basque Country
Universidad del País Vasco / Euskal Herriko Unibertsitatea

Supervisor: *Prof. Dr. Óscar Díaz García*
San Sebastián, Spain, 2012

This work was hosted by the *University of the Basque Country* (Faculty of Computer Sciences). It was partially supported by the *Spanish Ministry of Education and Science* (Ministerio de Educación y Ciencia) under contract TIC2002-01442.

To my wife Marisa

and

my daughter Patricia

Summary

Although blogs exist from the beginning of the Internet, their use has considerably been increased in the last decade. Nowadays, they are ready for being used by a broad range of people. From teenagers to multinationals, everyone can have a global communication space.

Companies know blogs are a valuable publicity tool to share information with the participants, and the importance of creating consumer communities around them: participants come together to exchange ideas, review and recommend new products, and even support each other. Also, companies can use blogs for different purposes, such as a content management system to manage the content of websites, a bulletin board to support communication and document sharing in teams, an instrument in marketing to communicate with Internet users, or a Knowledge Management Tool. However, an increasing number of blog content do not find their source in the personal experiences of the writer. Thus, the information can currently be kept in the user's desktop documents, in the companies' catalogues, or in another blogs. Although the gap between blog and data source can be manually traversed in a manual coding, this is a cumbersome task that defeats the blog's easiness principle. Moreover, depending on the quantity of information and its characterisation (i.e., structured content, unstructured content, etc.), an automatic approach can be more effective.

Based on these observations, the aim of this dissertation is to assist blog publication through annotation, model transformation and crossblogging techniques. These techniques have been implemented to give rise to *Blogouse*, *Catablog*, and *BlogUnion*. These tools strive to improve the publication process considering the aforementioned data sources.

Contents

Chapter 1

Introduction

"Never doubt that a small group of thoughtful, committed citizens can change the world. Indeed, it is the only thing that ever has."

– *Margaret Mead.*

1.1 Overview

In 1997, Jorn Barger, the editor of the *Robot Wisdom* website [Bar97], coined the term *weblog* to describe the process of "logging the web". This term is often shortened as blog. Late in 1999, the software developer Dave Winer offered a free hosting service at *EditThisPage.com* [Win99]. Then, people were able to publish their own *weblogs* quickly and easily. He defined *weblogs* as *"... often-updated sites that point to articles elsewhere on the web, often with comments, and to on-site articles. A weblog is kind of a continual tour, with a human guide who you get to know. There are many guides to choose from, each develops an audience, and there's also comraderie and politics between the people who run weblogs, they point to each other, in all kinds of structures, graphs, loops, etc."* [Win01] In February 2001, he claimed to be hosting approximately 20,000 websites. The whole of blogs and their interconnections is known as *blogosphere*. This term was coined by Brad L. Graham[1], a theater publicist, as a joke in 1999 [Gra99].

The boom of blogs arose soon after with the 9/11 aerial attacks. Then, people searched information to understand that event. And sometimes, first-hand testi-

[1]Acccording to http://en.wikipedia.org/wiki/Blogosphere

monies were published on blogs before traditional mass media (e.g. newspapers, TV, etc.) [BHH02]. In 2003, the Iraq War gave way to the *warblogs* (i.e., war specialised blogs). In Baghdad, some soldiers and citizens wrote their thoughts in their blogs and became real war correspondents. It is noteworthy the Salam Pax blog. What started as a blog written by an Iraqi citizen [Sal], ended as a book [Pax03].

According to Technorati, the largest blog search engine, the number of blogs from March 2003 to July 2006 was doubled about every 6 months [Sif06]. In March 2007, over 70 million blogs were tracked [Sif07].

A key factor of blog success is its simplicity of use. Before their appearance, people had to create an HTML (HyperText Markup Language) [RHJ99] formatted file, and then, upload it through FTP (File Transfer Protocol) [PR85]. Blogs put in the layman's hands the possibility of publishing web contents without this technical knowledge: only an Internet-connected computer was necessary. As a result, the publication process becomes democratic and, from then on, everyone can have his say in the Internet [Blo02]. Two roles emerge around blogs: *bloggers* and the *audience*. While *bloggers* publish contents in one or more blogs, the *audience* reads them. Part of the *audience* (a.k.a. *participants*) also comments on blogs. Active *bloggers* are also *participants*.

Often, blogs are blurred with web pages or forums. However, they differ on how their contents are managed by the different roles. Only *bloggers* can write in a blog, and their *audience* read or comment on it. Although web pages are very similar, the *audience* cannot comment. They are restricted to read them. It is remarkable that if the blog *comments* are disabled, both systems are identical, from the point of view of these roles. Nonetheless, it is, technically, easier to publish contents in a blog than in a web page. Finally, everyone can write, read, or respond in a forum. Moreover, the contents of a blog are more personal than those of a forum.

The blog community is very active. Since 2004, Deutsche Welle[2] has been organising the BOBs[3] (Best Of Blogs) awards to encourage the use of blogs [BOB]. Furthermore, there exists workshops and conferences on this topic [Blo06][Blob], and a great deal of tools has been developed [Blo07][Wor10c].

[2]Germany's international public broadcaster.

[3]The world's largest international awards for blogs, *audioblogs* and *videoblogs*.

1.2 Blogosphere typification

The increasing number of blogs and the diversity of uses make difficult the study of the *blogosphere*, and the blogs' needs. This section dissects blogs by classifying them through three characteristics: number of *bloggers*, content type, and visibility.

Depending on the number of *bloggers*, blogs can be **personal** or **multi-author** (a.k.a. *group blogs*). *Personal blogs* are written by a single *blogger*, who expresses his opinions, deals with topics he finds interesting (e.g. day-to-day experiences), filters information, etc. By contrast, *multi-author blogs* are written by two or more *bloggers*, who sometimes cover a variety of topics.

Thus, the aim of a blog is to publish information. The variability of the contents is based on **format**, and **topic**. In *format-based blogs*, such as *photoblogs* [Pho], *videoblogs* [Roc], *audioblogs* [Aud] or, simply, blogs; the main contents are photos, videos, audios (e.g. speeches, music, etc.), or texts, respectively. When blogs are about a clearly defined topic, such as war, education, business, etc., they gave way to *topic-based blog*, such as *warblogs* (a.k.a. *milblogs*) [Mil], *edublogs* [Edu], *business blogs* [Ama], etc. For instance, the latter improves workplace conversations, knowledge and project management, and direct communication with customers (e.g. retrieving feedback from products).

Finally, the visibility of a blog defines its *audience*. For instance, *intranet blogs* improve team communication in an organisation, *extranet blogs* share information with external stakeholder of an organisation (e.g. suppliers, partners, clients, etc.), and *Internet blogs* publish information for everyone. The latter are the most popular.

1.3 Contributions

The classification of the previous section does not pretend to be definitive, but it helps to clarify the scope of this dissertation. The contributions of this dissertation can be summarised as:

BLOG AS DIARIES

Problem Statement: Blogs can be used as a means to share information with their *audience*. This information is stored up in the users' computers with different file formats. There is a gap in the publication process between the content source (i.e., files) and the target blog that is overcome in a copy&paste manner (i.e., copying a text from a file to paste it in a blog).

Contribution: A mouse-based device was developed to automatise the publication process. The mouse menu is configured by an ontology which guides the document annotation process. Bloggers personalise the ontology in their own blogs. This device aims to fulfil the requirements of user-friendliness, as well as blog-system and editor independence.

BLOG AS VIRTUAL COMMUNITY PLATFORMS

Problem Statement: Some *business blogs* publish information about products in their catalogues (hereafter, referred to as "catablogs"). The *audience* creates a consumer community to exchange ideas, review and recommend the products, and even, support each other. However, the generation of a blog from a product catalogue is manually coded, and is hindered by the immaturity of blog technology.

Contribution: A cost-effective way to generate catablogs through a Model Driven Engineering (MDE) process is specified. Along this process, models are transformed, annotated, and composed to create an application. The execution of this application creates a catablog. The cost-effectiveness of catablog creation is analysed comparing the labour-cost of the MDE approach versus the manual coding.

BLOG AS PEERS

Problem Statement: Lecturers maintain their own *edublogs* as a conduit for lecturers to communicate with students (i.e., a place where course's themes or assignments can be published). Links can be defined among *edublogs* of distinct lecturers which sustain the very same course. However, *posts* and *comments* are isolated in the blog where they were written. Reading and writing the linked blogs implies to skip from one to another.

Contribution: The interaction between blogs has been automatised through a contract-based distributed and heterogeneous crossblogging framework. Thus, any content is susceptible of being automatically published on various blogs according to a contract. Now, contents skip from one blog to another, and not lecturers and students. The contract is established by two *peer* blogs, which can be distributed in different computers and implemented by different blog engines. The term *peer* is used as an entity (i.e., a blog) with capabilities similar to other entities in the system [DSMX02].

1.4 Outline

This dissertation is composed of six chapters, including this one, namely:

Chapter 2 introduces the background of blogs. Here, the different applications surrounding blogs are described, the blog structure is examined closely, and some definitions are given. Also, the contributions of chapters 3 to 5 are contextualised.

Chapter 3 addresses the problem of publishing content kept into the desktop files with different formats. To this aim, annotation techniques are applied to automatise the publication, and make the mouse be ontology-aware.

Chapter 4 describes an MDE architecture to automatise the generation of catablogs out of catalogues. It provides some trade-off studies comparing the manual coding versus the MDE approach.

Chapter 5 defines a blog extension to support a contract-based communication between blogs. The contract specification, negotiation, and implementation are described. RDF (Resource Description Framework) format [W3C04], and ECA (Event Condition Action) rules are the keys to the contract implementation, and interpretation, respectively.

Chapter 6 shows the conclusions, and summarises the main contributions of this dissertation. Finally, the publications achieved along the dissertation, and the future work are exposed.

Chapter 2

Background

“Keep It Simple, Stupid!”
– *Kelly Johnson.*

2.1 Introduction

What initially was thought as link-driven sites have evolved to a new mean of communication. Blogs create a shared conversational space between the *blogger* and his *audience*. These conversations create a community of interest around a topic, where linking and discussing are the commonplace [KNRT04]. As a result, stronger links between the *blogger* and his *audience* are created.

In consequence, a great deal of information is created everyday. Selecting what blog to read, keeping the blog up-to-date, and promoting a blog are a daily challenge for many *bloggers*. To manage this myriad of information, a broad range of tools and characteristics are built around blogs.

Next sections describe how *bloggers* and the *audience* can interact with blogs, and how blogs satisfy aforementioned characteristics. Finally, contributions are revisited to contextualise the contributions of this dissertation.

2.2 Blog Interfaces

Blog interaction can be achieved through three different kinds of interfaces, namely, graphical-user interfaces (GUI), syndication interfaces, and programmatic interfaces. The former are targeted at customary users, and present a human-readable version of the blog. The rest are machine-readable interfaces, used to interact with other tools. More details follow.

2.2.1 Graphical-User Interfaces

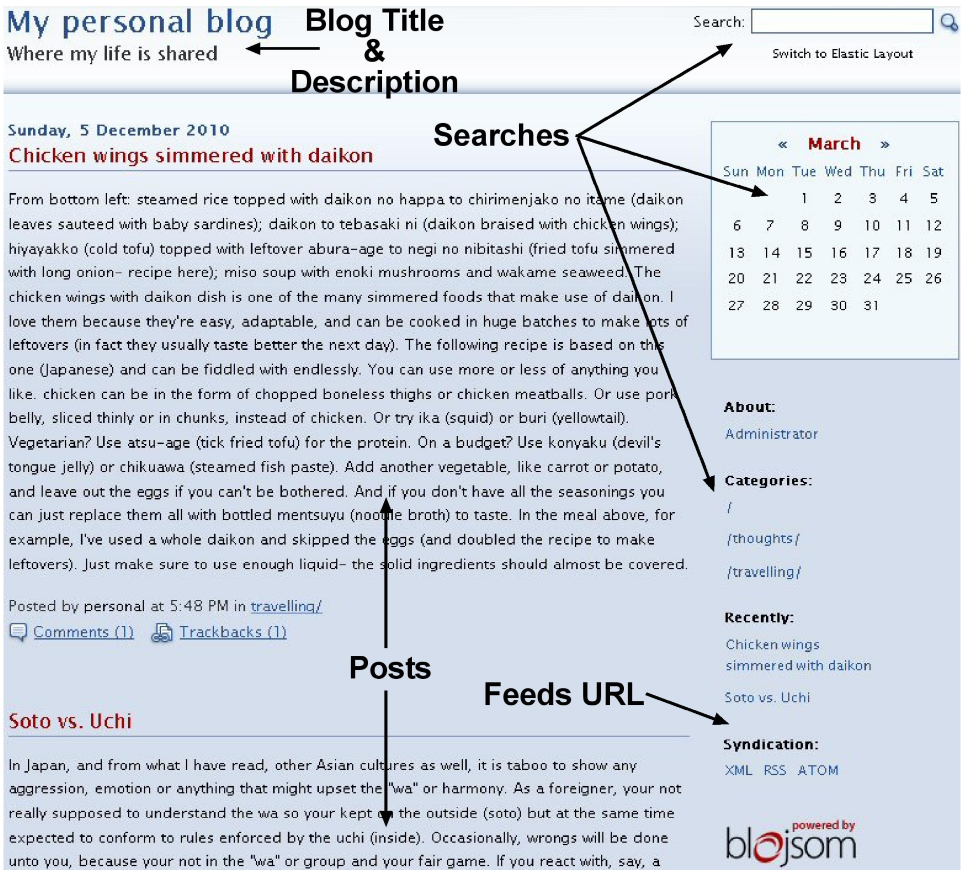
From the point of view of the *audience*, a blog is a frequently updated online journal or newsletter with different types of pages: *index page*, *archive page*, and *additional pages*. At the top of these pages, the blog *title*, and, usually, a short *description* are placed. It is common for these pages to link related blogs (e.g. *blogrolls*, *webrings*, etc.), and provide notification services (e.g. *content syndication* capabilities, *web pings*, etc.).

The *index page* is the main entry point to the blog. Its location (i.e., the blog URL) is the only one that the *audience* has to remember. This page consists of a sequence of usually short articles, namely *posts*, which are usually arranged in reverse chronological order (i.e., the most recent *post* at the top of the page, and the oldest one at the bottom) [Win01][BHH02]. The *index page* only shows the latest *posts*. Meanwhile, the rest of the *posts* remain archived permanently. The blog provides search facilities to access the archive, such as text-based, category-based, author-based, calendar-based (i.e., daily, monthly, yearly, etc.), alphabetical search, etc. [Wor10a]

The *archive page* shows a sequence of *posts* filtered by a criteria (e.g. by keyword, category, date, etc.). Just like the *index page*, the *posts* are usually arranged chronologically. Not only is this page a backup of the *index page's posts*, but also it enables the *audience* to read what was published on the blog [BHH02]. Thus, the *audience* can contextualise the *posts*.

Also, there exists *additional pages* where the *blogger* can promote himself, publish some photo galleries, receive emails through a contact form, etc. [BHH02]

All these pages share a consistent design under the notion of *theme*. A *theme* is a cascading style sheets (CSS) [W3C] based package containing graphical appearance details. It is used to customise the look and feel of the blog. The *blogger* can manage blog *themes*, changing, removing, or including a new one [BHH02]. Also, they can develop, buy, or download them from another sites [Bloc][Bloe][Blof][Blod] [Tem]. Figure 2.1 displays a blog GUI.



My personal blog
Where my life is shared

Search:
Switch to Elastic Layout

Sunday, 5 December 2010
Chicken wings simmered with daikon

From bottom left: steamed rice topped with daikon no happa to chirimenjako no itame (daikon leaves sauteed with baby sardines); daikon to tebasaki ni (daikon braised with chicken wings); hiyayakko (cold tofu) topped with leftover abura-age to negi no nibitashi (fried tofu simmered with long onion- recipe here); miso soup with enoki mushrooms and wakame seaweed. The chicken wings with daikon dish is one of the many simmered foods that make use of daikon. I love them because they're easy, adaptable, and can be cooked in huge batches to make lots of leftovers (in fact they usually taste better the next day). The following recipe is based on this one (Japanese) and can be fiddled with endlessly. You can use more or less of anything you like. chicken can be in the form of chopped boneless thighs or chicken meatballs. Or use pork belly, sliced thinly or in chunks, instead of chicken. Or try ika (squid) or buri (yellowtail). Vegetarian? Use atsu-age (tick fried tofu) for the protein. On a budget? Use konyaku (devil's tongue jelly) or chikuawa (steamed fish paste). Add another vegetable, like carrot or potato, and leave out the eggs if you can't be bothered. And if you don't have all the seasonings you can just replace them all with bottled mentsuyu (noodle broth) to taste. In the meal above, for example, I've used a whole daikon and skipped the eggs (and doubled the recipe to make leftovers). Just make sure to use enough liquid- the solid ingredients should almost be covered.

Posted by personal at 5:48 PM in travelling/
Comments (1) Trackbacks (1)

Soto vs. Uchi

In Japan, and from what I have read, other Asian cultures as well, it is taboo to show any aggression, emotion or anything that might upset the "wa" or harmony. As a foreigner, your not really supposed to understand the wa so your kept on the outside (soto) but at the same time expected to conform to rules enforced by the uchi (inside). Occasionally, wrongs will be done unto you, because your not in the "wa" or group and your fair game. If you react with, say, a

« March »

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

About:
Administrator

Categories:
/
/thoughts/
/travelling/

Recently:
Chicken wings simmered with daikon
Soto vs. Uchi

Syndication:
XML RSS ATOM

powered by blojsom

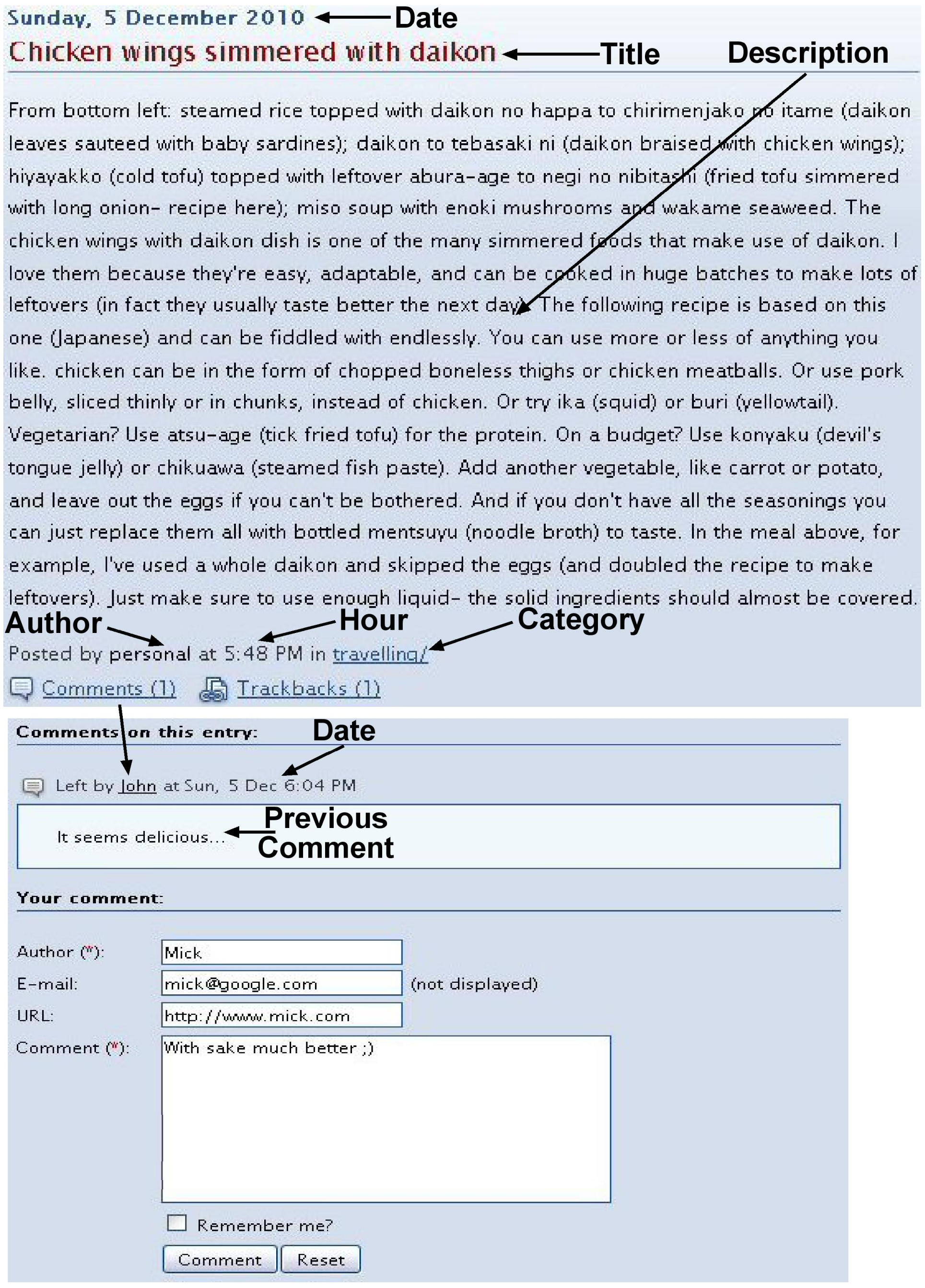
Figure 2.1: Blog Graphical-User Interface.

Post Structure

A *post* (a.k.a. *entry*) is the atomic unit of a blog. Its structure includes a *title* and a *description*. The *description* can be composed by contents of different formats. The most common format is HTML, which allows *bloggers* to use anchors to refer to any URL addressable content. Its length varies with the *blogger*. Some of them decide to show only an excerpt of *posts'* content (e.g. the first 100 characters). Thus, their *audience* can make a quick glace at the blog contents without doing long scrolls. In this case, each *post* provides a link to its complete content.

Metadata give more information about *posts*. They are a set of *(property, value)* pairs attached to a *post*. The most outstanding *metadata* are the following [BHH02]:

- **Authoring** points to the writer of a *post*, and is especially important in *multi-author blogs*.


Sunday, 5 December 2010

Chicken wings simmered with daikon

From bottom left: steamed rice topped with daikon no happa to chirimenjako no itame (daikon leaves sauteed with baby sardines); daikon to tebasaki ni (daikon braised with chicken wings); hiyayakko (cold tofu) topped with leftover abura-age to negi no nibitashi (fried tofu simmered with long onion- recipe here); miso soup with enoki mushrooms and wakame seaweed. The chicken wings with daikon dish is one of the many simmered foods that make use of daikon. I love them because they're easy, adaptable, and can be cooked in huge batches to make lots of leftovers (in fact they usually taste better the next day). The following recipe is based on this one (Japanese) and can be fiddled with endlessly. You can use more or less of anything you like. chicken can be in the form of chopped boneless thighs or chicken meatballs. Or use pork belly, sliced thinly or in chunks, instead of chicken. Or try ika (squid) or buri (yellowtail). Vegetarian? Use atsu-age (tick fried tofu) for the protein. On a budget? Use konyaku (devil's tongue jelly) or chikuawa (steamed fish paste). Add another vegetable, like carrot or potato, and leave out the eggs if you can't be bothered. And if you don't have all the seasonings you can just replace them all with bottled mentsuyu (noodle broth) to taste. In the meal above, for example, I've used a whole daikon and skipped the eggs (and doubled the recipe to make leftovers). Just make sure to use enough liquid- the solid ingredients should almost be covered.

Posted by personal at 5:48 PM in travelling/

Comments (1) Trackbacks (1)

Comments on this entry:

Left by John at Sun, 5 Dec 6:04 PM

It seems delicious...

Your comment:

Author (*):	Mick	
E-mail:	mick@google.com	(not displayed)
URL:	http://www.mick.com	
Comment (*):	With sake much better ;)	

Remember me?

Figure 2.2: The ins and outs of a *post*.

- **Date** gives a temporal context about a *post*. Thanks to this element, the *audience* can experience a strong connection with *bloggers*. For instance, if a *blogger* writes a *post* and someone reads it at once, the reader has the impression of sharing that moment with the *blogger*. Thus, the blog is seen as a shared place between the *blogger* and his *audience*.

- **Categorisation** is used to classify *posts* under *blogger* criteria.

- **Permalink**, or permanent link, is a unique and permanent URL identifier that anyone can use to point to a specific *post*. Before *permalinks*, the *index page* was the only way of referring to a *post*. However, as a new *post* was published, the oldest one disappeared from the *index page*, and the references became completely useless. The *permalink* solved this problem and, from then on, linking to a specific *post* eased the discussion between blogs [Meg02][Tom03].

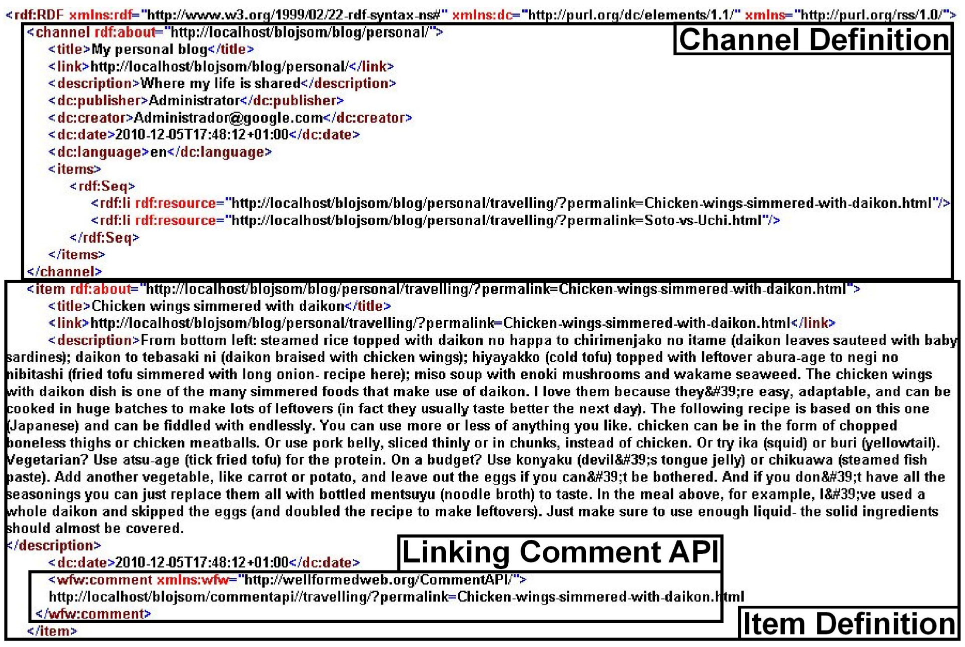
Moreover, *bloggers* can add their own *metadata* for additional processing of *posts* (e.g. for indexing or querying purposes). Also, they can be used to categorise *posts*. Basically, the difference between a *category* and a *metadata* categorisation is when each one is created. The existence of a *category* does not depends on the existence of a *post*. However, a *metadata* is always attached to a *post*. Therefore, the existence of *metadata* depends on the existence of a *post*. Also, unlike the *categories*, *metadata* are not usually shown in the graphical-user interface.

Finally, a list of *comments* are attached to each *post*. They are usually written by *participants*. They are anonymous to provide freedom of expression. However, some *bloggers* disable them to avoid feedback from the *audience*. Others moderate them to avoid publishing undesirable *comments* (e.g. spam) [Wor10a]. *Comments* are based on the *post* content, and can link to other contents. Although they have the *comment* publication *date* as single compulsory *metadata*, some blog engines ask for extra information, e.g. *authoring*, *email*, an *URL* and the *comment*.

Figure 2.2 zooms in on the first *post* of the Figure 2.1 to stand out *post*'s elements. At the bottom of the figure, the *comment* GUI with its more characteristic elements is depicted.

2.2.2 Syndication Interfaces

RSS (Really Simple Syndication) [Dav03] is a popular XML format for syndicating news, also referred to as *feeds*. The *feed* provides a machine-readable de-



```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns="http://purl.org/rss/1.0/">
  <channel rdf:about="http://localhost/blojsom/blog/personal/">
     <title>My personal blog</title>
     <link>http://localhost/blojsom/blog/personal/</link>
     <description>Where my life is shared</description>
     <dc:publisher>Administrator</dc:publisher>
     <dc:creator>Administrador@google.com</dc:creator>
     <dc:date>2010-12-05T17:48:12+01:00</dc:date>
     <dc:language>en</dc:language>
     <items>
        <rdf:Seq>
           <rdf:li rdf:resource="http://localhost/blojsom/blog/personal/travelling/?permalink=Chicken-wings-simmered-with-daikon.html"/>
           <rdf:li rdf:resource="http://localhost/blojsom/blog/personal/travelling/?permalink=Soto-vs-Uchi.html"/>
        </rdf:Seq>
     </items>
  </channel>
  <item rdf:about="http://localhost/blojsom/blog/personal/travelling/?permalink=Chicken-wings-simmered-with-daikon.html">
     <title>Chicken wings simmered with daikon</title>
     <link>http://localhost/blojsom/blog/personal/travelling/?permalink=Chicken-wings-simmered-with-daikon.html</link>
     <description>From bottom left: steamed rice topped with daikon no happa to chirimenjako no itame (daikon leaves sauteed with baby
sardines); daikon to tebasaki ni (daikon braised with chicken wings); hiyayakko (cold tofu) topped with leftover abura-age to negi no
nibitashi (fried tofu simmered with long onion- recipe here); miso soup with enoki mushrooms and wakame seaweed. The chicken wings
with daikon dish is one of the many simmered foods that make use of daikon. I love them because they&#39;re easy, adaptable, and can be
cooked in huge batches to make lots of leftovers (in fact they usually taste better the next day). The following recipe is based on this one
(Japanese) and can be fiddled with endlessly. You can use more or less of anything you like. chicken can be in the form of chopped
boneless thighs or chicken meatballs. Or use pork belly, sliced thinly or in chunks, instead of chicken. Or try ika (squid) or buri (yellowtail).
Vegetarian? Use atsu-age (tick fried tofu) for the protein. On a budget? Use konyaku (devil&#39;s tongue jelly) or chikuawa (steamed fish
paste). Add another vegetable, like carrot or potato, and leave out the eggs if you can&#39;t be bothered. And if you don&#39;t have all the
seasonings you can just replace them all with bottled mentsuyu (noodle broth) to taste. In the meal above, for example, I&#39;ve used a
whole daikon and skipped the eggs (and doubled the recipe to make leftovers). Just make sure to use enough liquid- the solid ingredients
should almost be covered.
</description>
     <dc:date>2010-12-05T17:48:12+01:00</dc:date>
     <wfw:comment xmlns:wfw="http://wellformedweb.org/CommentAPI/">
     http://localhost/blojsom/commentapi//travelling/?permalink=Chicken-wings-simmered-with-daikon.html
   </wfw:comment>
  </item>
```

Figure 2.3: The RSS 1.0 syndication interface.

scription of blog contents to allow *posts* to be aggregated into a *feed aggregator* application[1]. This description follows distinct dialects of RSS [Ham03].

Figure 2.3 shows the RSS 1.0 format [Web02] of the Figure 2.1. The blog is mapped to the notion of *channel* in the RSS model (i.e., a *feed provider*). A *channel* has a *title*, *link*, *description* and another optional elements (e.g. a *language*), followed by a series of *items* (a.k.a. news), each of which, also, have a *title*, *link*, *description* and another optional elements.

In addition, the RSS 1.0 dialect [Web02] permits the use of XML Namespaces [W3C09] to accommodate additional tags (a.k.a. *modules* in the RSS parlance). The reason rests on RSS 1.0 lack of a central authority for extending the format. Instead, *modules* are used for third parties to accommodate the *feed* description to their own purposes. This is the case for the *comment* element, which uses the namespace *wfw* to define the Comment API [The03] access in Figure 2.3.

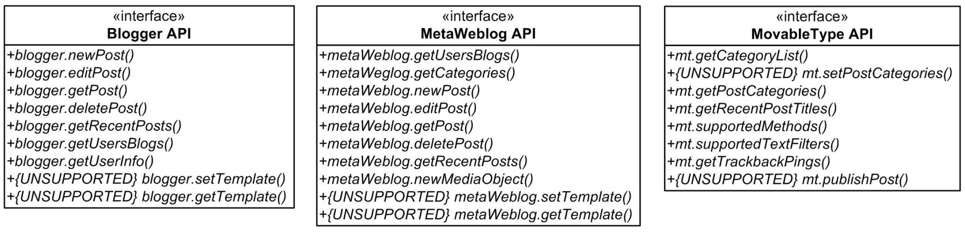

«interface»
Blogger API
+blogger.newPost()
+blogger.editPost()
+blogger.getPost()
+blogger.deletePost()
+blogger.getRecentPosts()
+blogger.getUsersBlogs()
+blogger.getUserInfo()
+{UNSUPPORTED} blogger.setTemplate()
+{UNSUPPORTED} blogger.getTemplate()
«interface»
MetaWeblog API
+metaWeblog.getUsersBlogs()
+metaWeglog.getCategories()
+metaWeblog.newPost()
+metaWeblog.editPost()
+metaWeblog.getPost()
+metaWeblog.deletePost()
+metaWeblog.getRecentPosts()
+metaWeblog.newMediaObject()
+{UNSUPPORTED} metaWeblog.setTemplate()
+{UNSUPPORTED} metaWeblog.getTemplate()
«interface»
MovableType API
+mt.getCategoryList()
+{UNSUPPORTED} mt.setPostCategories()
+mt.getPostCategories()
+mt.getRecentPostTitles()
+mt.supportedMethods()
+mt.supportedTextFilters()
+mt.getTrackbackPings()
+{UNSUPPORTED} mt.publishPost()

Figure 2.4: Blojsom APIs.

2.2.3 Programming Interfaces

The programming interfaces allow programmers to achieve most of the functions which can be obtained through the GUI, also programmatically (e.g. creating a *post*, introducing a *comment*, etc.). Although there is not such thing as a standard for blog APIs (Application Programming Interfaces), there is some consensus about the functionality to be supported. There are two types of interfaces: (1) those built on top of the XML-RPC[2] protocol [Dav99], such as Blogger API 1.0 [Eva03], MetaWeblog API [Dav02], MovableType API [Mov05] and Weblogs.Com [Dav01]; (2) those based in HTTP calls, such as Comment API [The03] and Trackback API [Six04]. These APIs permit the construction of applications for blog engines, such as publication, notification or commenting tools. Also, the blog developers can build their own APIs. The uses of these interfaces are covered more deeply in future sections. Figure 2.4 depicts the different interfaces implemented in Blojsom[3] [Dav].

2.3 Publication Tools

Blogs allow *bloggers* to publish *posts* and *comments* through web-based interfaces or desktop clients (a.k.a. blog clients) [DDJ$^+$02][Wor10a].

Blog engines (e.g. [Dav]) provide *back-end pages* to ease blog management. From these pages, *bloggers* can publish *posts* without technical knowledge (e.g. HTML, FTP, etc.), categorise them, change *themes*, etc. Because of the importance

[1]A *feed aggregator* (a.k.a. *feed reader*) periodically checks the *feed* at the blog for changes, reacts to the changes in an appropriate way, and, commonly, keeps an orderly and updated record of the *feeds* at which it is subscribed to.

[2]XML-RPC is an internet-based cross-platform remote-procedure calling protocol which uses HTTP as the transport medium, and XML as the encoding system for the communication between different platforms.

[3]Notice that the developers do not have implemented these APIs completely.

of managing such blog's elements, these pages are protected by password. So, initially, *bloggers* have to open his browser at the *back-end* login page, insert their username and password, and start working.

Publishing a *post* involves selecting the *post* creation page, and fill out a form to complete the *post* structure. In this form, some fields are compulsorily completed by the *blogger*, such as the *title* and the *description*. The *permalink* and the *author* are generated by the blog engine. Moreover, although a default publication *date*[4] and *category* are proposed by the blog, they can be changed. Optionally, *bloggers* add *metadata*, and send notifications. Thus, the content design and other technicalities (e.g. sending notifications) are up to the blog engine, and the *blogger* can focus on writing.

The desktop clients are applications that run on a computer and communicates with a blogging system, usually, through an XML-RPC protocol [DDJ$^+$02]. They provide a blog engine independent interface and, usually, WYSIWYG (What You See Is What You Get) edition capabilities (see [Odd10], and [Lim10]). The latter allows *bloggers* to easily add style to the content (e.g. bold, italic, etc.). Unlike other Windows editors, the desktop clients require to configure an XML-RPC blog connection, and buttons for content publication. Also, the desktop clients provide an environment where *blogger* can write their *posts* offline, and publish them when Internet connection is available. They can provide extra information (e.g. geo positioning) when installed on mobile devices, such as mobile phones, PDAs, etc. Also, photos, videos or audios can be published instantly. The action of publishing content from these devices is known as mobile blogging or *moblogging*. It can be used by *bloggers* who lack of computer access.

As an alternative to installing an independent application, some plugins are developed. When a plugin is installed, it appears as a new toolbar inside an application (e.g. Microsoft Word [Blo05], Firefox [Goo10], etc.). Then, *bloggers* can publish the selected content displayed by the application. This is known as the *BlogThis!* button.

In addition to the desktop clients, there exists server side solutions which allow *bloggers* to publish contents in a blog by sending an email from their email client [Wor10b], an SMS (Short Message Service) from a mobile phone [Let06], etc. Thus, it is not needed to install any software. It is up to the server to manage this kind of messages.

[4]Some blog engines allow *bloggers* to publish *posts* at a future date. In this case, the *blogger* can change the publication date.

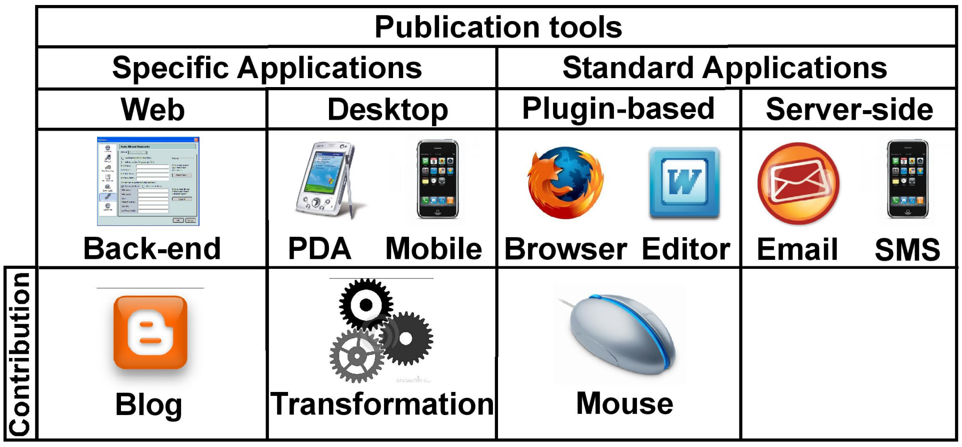


Figure 2.5: A typification of publication tools.

Although *posts* are just published after being written, some publication tools allow *bloggers* to draft it, or publish it on a fixed date [DDJ+02][Dav]. Publishing a draft of a *post* involves *blogger* confirmation. On the contrary, a *post* is automatically published at the predefined date.

2.4 Contributions revisited

Figure 2.5 summarises the publication tools, according to the previous section. Also, the contributions of this dissertation are contextualised.

The **specific applications** are independent from other applications. They are classified as web-based or desktop-based. The **standard applications** are already existing applications which extend their functionality to publish contents in a blog. This is achieved through plugins and server-side solutions. The contributions of this dissertation are classified following these criteria in the last row of the table.

Along the life cycle of this dissertation, the blog engines evolved. The work presented in this dissertation has been developed using Blojsom as blog engine [Dav], although the ideas exposed can be applied to others. Moreover, different versions of Blojsom were needed in some sections.

Blojsom is a full-featured, multi-user, multi-blog, open source web application developed in Java. It aims to retain simplicity in design while adding user flexibility in storage, *themes*, etc. Also, the Blojsom development community has been very active along this dissertation not only in programming it, but also in specifying it. These characteristics as a whole have been decisive to select Blojsom, up to the point that it has not been needed to use any additional blog engine to develop the contributions of this dissertation. By chapter, these contributions are the following:

BLOG AS DIARIES

Contribution: A mouse-based desktop client and some blog extensions to autodiscover the configuration have been developed. Also, the configuration format has been defined.

BLOG AS VIRTUAL COMMUNITY PLATFORMS

Contribution: Given a catalogue and a blog engine, a new blog instance is generated with the products of the catalogue. The key point of the generation process has been to define models and configuration elements. As programming interfaces do not cover the generation of all blog elements, new interfaces has been developed. Also, a part of the generation process can be seen as a transformation-based desktop client. The contribution is focused on *business blogs*.

BLOG AS PEERS

Contribution: A server-based publication client focused on *edublogs*, where the interaction between blogs has been automatised through a contract. Thus, blog's contents can seamlessly flow between blogs. This work describes both the contract life cycle and an RDF-based contract specification using ECA rules.

Chapter 3

Blog as Diaries

“Without the data, the tools are useless; without the software, the data is unmanageable.”
– Tim O’Reilly.

3.1 Introduction

Initially thought as personal diaries, blogs’ scope has broadened to become a medium for professionals to communicate [Röl03, Röl04]. Indeed, a report by Forrester [Li04] suggests blogging as a valuable publicity tool for companies and a means to keep in touch with their customer base. Enterprises can use blogs for different purposes: as a content management system to manage the content of websites, as a bulletin board to support communication and document sharing in teams, as an instrument in marketing to communicate with Internet users, or as a Knowledge Management Tool [Röl03].

This implies that an increasing number of *posts* do not find their source in the personal experiences of the *blogger*. Rather, the blog is used as a means to share information with others. And this information is currently kept in the user’s desktop. Regardless of the origin of the information (e.g. an Excel figure, a paragraph found in a PDF document or a slide in a PPT presentation), this content is amenable to be blogged, i.e., to be discussed and shared with the *participants*.

This scenario leads to a decoupling between the blog as a sharing platform, and the desktop as a resource container. Currently, this gap is manually traversed by the *blogger* in a copy&paste manner, a cumbersome task that defeats the blog's easiness principle. Consider the following scenario. Reading the sport newspaper on the web, you come across with some awful *comment* on the performance of your favorite soccer player on the last match. You want to collect the opinions of your blogmates about this issue. To this end, you copy the corresponding paragraph from the newspaper and you blog it. The latter implies to go to the blog, create a new *post*, edit it, clean it[1] and, finally, save it. Might be too cumbersome to be worth the effort!! Blog is an spontaneous action which can be refrained if the process is lengthy and unhandy.

Fortunately, some browsers (e.g. Firefox, Mozila) and search engines (e.g. Google) are providing the *BlogThis!* functionality. This functionality achieves "copy&blog" in an automatic way: posting a Web page is as easy as clicking on the *BlogThis!* icon. This opens a small window where a blog *post* is edited after the HTML page and a pointer are included to this source page.

Although similar functionality is also available for Word documents [Blo05], there is no knowledge about the existence for other formats. This means that interesting quotes from PDF or Excel documents can only be posted through laborious copy&paste, defeating spontaneous blogging. And even if this were the case, a *BlogThis!* functionality on an editor basis would require the user to become familiarised with distinct GUI interfaces (for Word, for Excel, for IE), putting an additional burden on the layman.

Based on this observation, this chapter describes a *BlogThis!* device for desktop resources that is editor independent: ***Blogouse***. *Blogouse* is a blog client that achieves editor independence by building on the mouse rather than the editor. By clicking on the mouse's middle button, *post* can be created from no matter which editor. Specifically, this device aims to fulfil the following requirements:

- user-friendliness. Usability is a main quality requirement in any blog interface to promote the participation of the layman. Sophisticated functionality should not be obtained at the cost of convoluted interfaces.

- blog-system independence. A myriad of tools are currently available for laymen to build up their blogs. *Blogouse* should be as independent as possible from the underlying blog system. Specifically, this blog client builds on

[1]A blog entry is basically text. Figures or rendering tags needs a special treatment.

top of the XML-RPC API which is becoming a *de facto* standard [Dav99]. Many blog systems follow this API, including *Blogger* [Bloa], *RadioUserland* [Rad], *MovableType* [Sixa] and *Blojsom* [Dav].

- editor independence. The device should work with any editor, no matter the format of the file. This includes IExplorer, Netscape, Acrobat Reader, XMLSpy, OpenOffice or any editor of the Microsoft Office suite. In this way, the source of the blog can be a PDF document, a Word document, an Excel spreadsheet, an HTML page, or an XML file.

The rest of the chapter describes the contribution (Section 3.2), *Blogouse* blog client (Section 3.3), and, finally, evaluation (Section 3.4) and conclusions (Section 3.5) are given.

3.2 Blog clients for second-generation blogs

A blog client works locally, and publishes globally. Unlike remote editing, local editing permits to incorporate additional content without caring about connexion problems. For shallow, semanticless, typed *posts*, this could not be regarded as an important advantage as editing a *post* just takes few seconds. However, second-generation blogging oversees the initial use of blogs as personal diaries to become lightweight content management systems [Röl03].

An additional argument in favour of blog clients is the tight integration with the desktop resources. Web-based editing do not permit the sophisticated GUI gadgets available in a desktop (e.g. drag&drop, mouse mobility, etc.) and to which users are accustomed. As pointed out in [MD05], *"access to other desktop applications and their data (e.g. through their public APIs), control of the clipboard, and techniques like drag-and-drop are difficult or impossible to implement in a web-based environment"* which make them also favour a desktop approach.

The *BlogThis!* functionality permits *posts* to be constructed out of existing desktop resources (e.g. a PDF file, an HTML page, an Excel spreadsheet). The *title*, the *description*, and other properties of the *post* should be mostly derived from the resource itself. Hence, posting is equated with annotating the resource along "the blog ontology".

Some of the information can be automatically extracted when the mapping from the document content is clear. However, this is not always the case. For instance, the *title* can be extracted from different places, such as the document title, a section name, or, even, the beginning of a phrase. This situation deteriorates when distinct formats and editors are considered. Automatic extraction could sacrifice editor/format independence, a hallmark of this chapter's approach.

Incorporating a *BlogThis!* button into each editor (i.e., as provided by Google) will certainly lead to an integrated solution but at the price of coupling editing and blogging. The myriad of formats which can be found in current desktop (e.g. *.doc*, *.xml*, and *.txt,* to mention a few), and the corresponding editors, vindicate the use of an editor-independent solution which relieves the user of such a burden. In this chapter, the mouse is used to attain this aim.

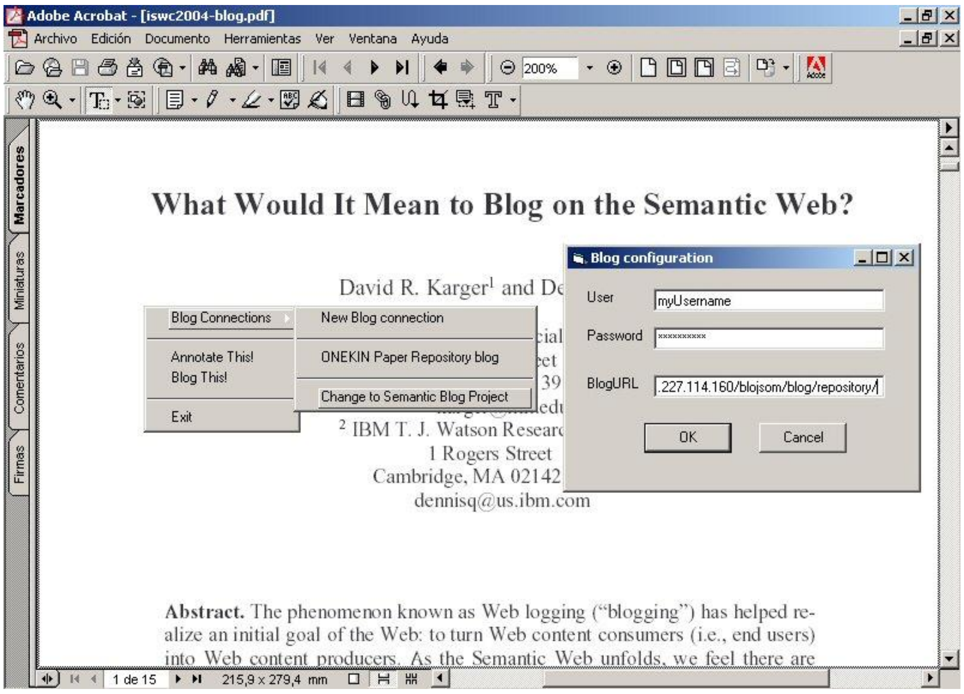
Rather than using the extensibility technology provided by each editor (e.g. *ActiveX* controls in the case of Word), the development of this chapter is moved down to the operating system so that the solution can be available to no matter which editor. The result is *Blogouse*, a *BlogThis!* device that achieves editor-independence by working at the operating-system level. By clicking on the mouse's middle button, a *post* is obtained from a resource regardless of the editor you are working with. In this way, the user does not have to move to a new editor when posting (e.g. desktop clients), nor has to learn a new GUI when resources from different formats are edited (e.g. Google).

In this way, this chapter strives to enhance the functionality without loosing the simplicity and usability that make blogware so popular. Next section describes *Blogouse* through a sample case.

3.3 Blogouse at work

BlogThis! can be regarded as an extractive process that obtains a *post* out of an external resource. If all *posts* have the same characterisation, the only source of variation in this process will be the external resource as such, i.e., whether the *post* is obtained from an HTML or a PDF document. This source of variation is faced by abstracting from the editor at hand, and working at the operating-system level.

However, we envisage a second source of variation, namely, the characterisation of the *post*. It can vary depending on the blog ontology. So far, conventional tags are used to describe no matter which *post* regardless of its content. However, authors envisage a promising scenario where *posts* can be annotated to surface their semantic content, and become integral parts of the semantic Web [KQ04]. In this setting, *post* extraction depends on the ontology at hand, and the blog client needs to be configured with the blog ontology. Next subsection describes this configura-



What Would It Mean to Blog on the Semantic Web?

David R. Karger[1] and De

[2] IBM T. J. Watson Researc
1 Rogers Street
Cambridge, MA 02142
dennisq@us.ibm.com

User myUsername
Password xxxxxxxxxx
BlogURL .227.114.160/blojsom/blog/repository/

Abstract. The phenomenon known as Web logging ("blogging") has helped realize an initial goal of the Web: to turn Web content consumers (i.e., end users) into Web content producers. As the Semantic Web unfolds, we feel there are

Figure 3.1: *Blogouse* configuration.

tion which is previous to the *BlogThis!* process.

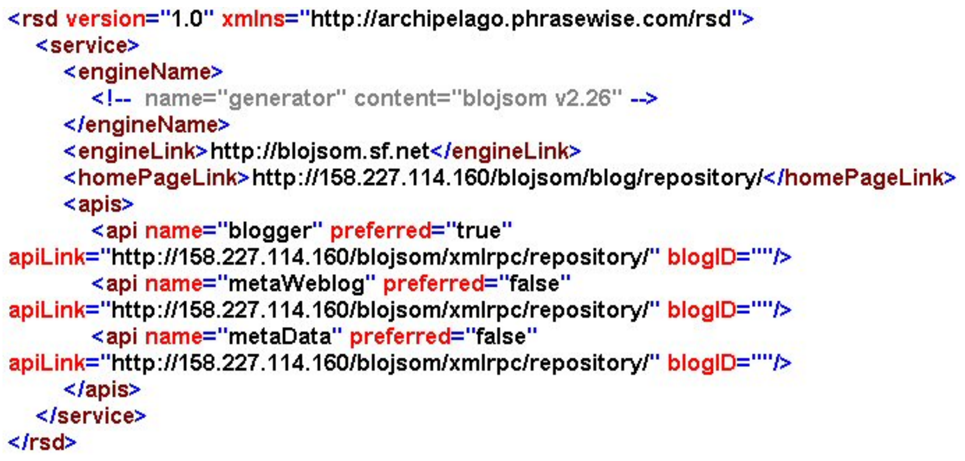
3.3.1 Blogouse configuration

Figure 3.1 describes the menus for configuring *Blogouse*. By clicking on the middle button of the mouse, the user is prompted to indicate a previous profile (e.g. *"Semantic Blog Project"*) or to enter a new one. In this latter case, a blog main page URL, an user name and a password are provided to establish a new profile. At any time, the user can change the active profile (i.e., the publishing blog).

When a blog connexion is created, *Blogouse* makes some *HTTP* requests to configure itself. Firstly, the *BlogURL* is requested. As a result, three data are obtained:

- the blog's *title* (*"ONEKIN Paper Repository Blog"*), which is used to name the profile,

- the RSD (Really Simple Discovery) URL [RSD], which holds the URI for a RSD file. RSD aims at simplifying *"the discovery of setting information by*



```
<rsd version="1.0" xmlns="http://archipelago.phrasewise.com/rsd">
  <service>
    <engineName>
      <!-- name="generator" content="blojsom v2.26" -->
    </engineName>
    <engineLink>http://blojsom.sf.net</engineLink>
    <homePageLink>http://158.227.114.160/blojsom/blog/repository/</homePageLink>
    <apis>
      <api name="blogger" preferred="true"
apiLink="http://158.227.114.160/blojsom/xmlrpc/repository/" blogID=""/>
      <api name="metaWeblog" preferred="false"
apiLink="http://158.227.114.160/blojsom/xmlrpc/repository/" blogID=""/>
      <api name="metaData" preferred="false"
apiLink="http://158.227.114.160/blojsom/xmlrpc/repository/" blogID=""/>
    </apis>
  </service>
</rsd>
```

Figure 3.2: An RSD file.

reducing the information a user must supply to three well known elements[2]. *Assuming that the RSD file is generated by the blogging software (Radio, Manila, etc.), the other required bits of information, not well known to the user, can be easily discovered and supplied"* [RSD]. The rationale for RSD rests on the heterogeneity of the formats and manners of the services available at blog engines that pose important interoperability problems to blog clients. Figure 3.2 gives an snippet of an RSD file[3].

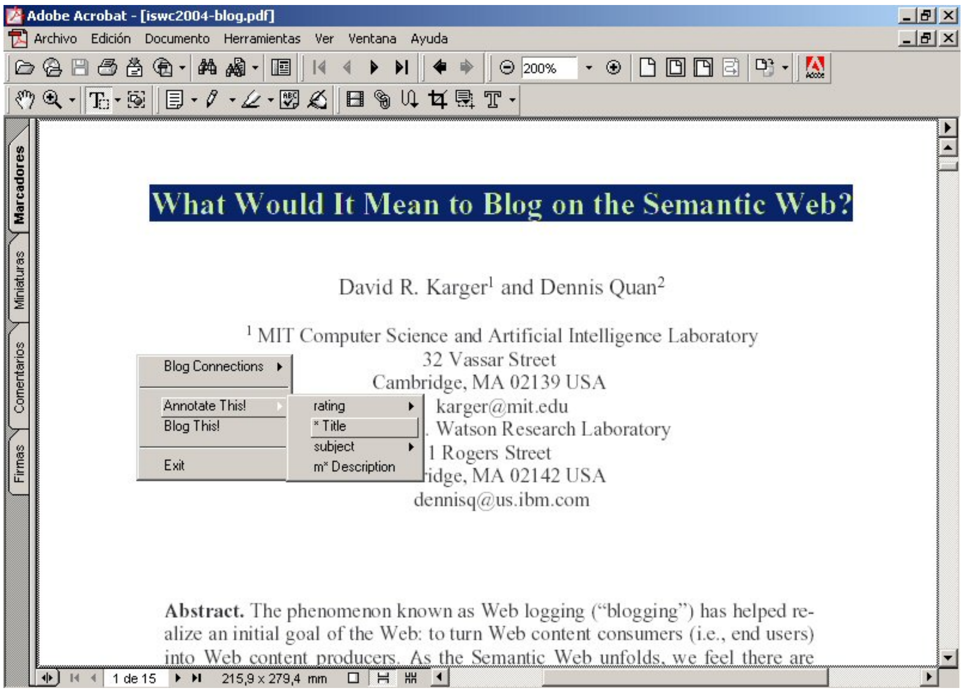
- the ontology URL, which holds a URI for the file that keeps the blog ontology. The terms and associations defined in this ontology are then used to configure dynamically the mouse menu during the *BlogThis!* process (i.e., the drop-down menus used for annotation).

All these data are embedded through *HTML* elements into the blog home page's header. Once the homepage is retrieved, the blog client filters <*title*> and <*link*> elements, and obtains the APIs as well as the blog ontology. The following code shows a <link> element definition for retrieving the ontology:

> *<link rel="xrss" type="application/rdf+xml" title="Extended RSS 1.0 Schema" href="http://158.227.114.168/blojsom/resources/repository/xRSS1.0.rdf" />*

[2]These elements correspond to the attributes *name*, *apiLink*, and *blogId*, which are necessary to a blog connection establishment.

[3]<*engineName*> holds the name of the blog engine, <*engineLink*> keeps the URL to the engine's home, <*homePageLink*> has the URL of the blog itself, <*apis*> contains a sequence of apis to be used to interact with the blog. Additional elements can be included to point to documentation.



What Would It Mean to Blog on the Semantic Web?

David R. Karger[1] and Dennis Quan[2]

[1] MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar Street
Cambridge, MA 02139 USA
karger@mit.edu
. Watson Research Laboratory
1 Rogers Street
idge, MA 02142 USA
dennisq@us.ibm.com

Abstract. The phenomenon known as Web logging ("blogging") has helped realize an initial goal of the Web: to turn Web content consumers (i.e., end users) into Web content producers. As the Semantic Web unfolds, we feel there are

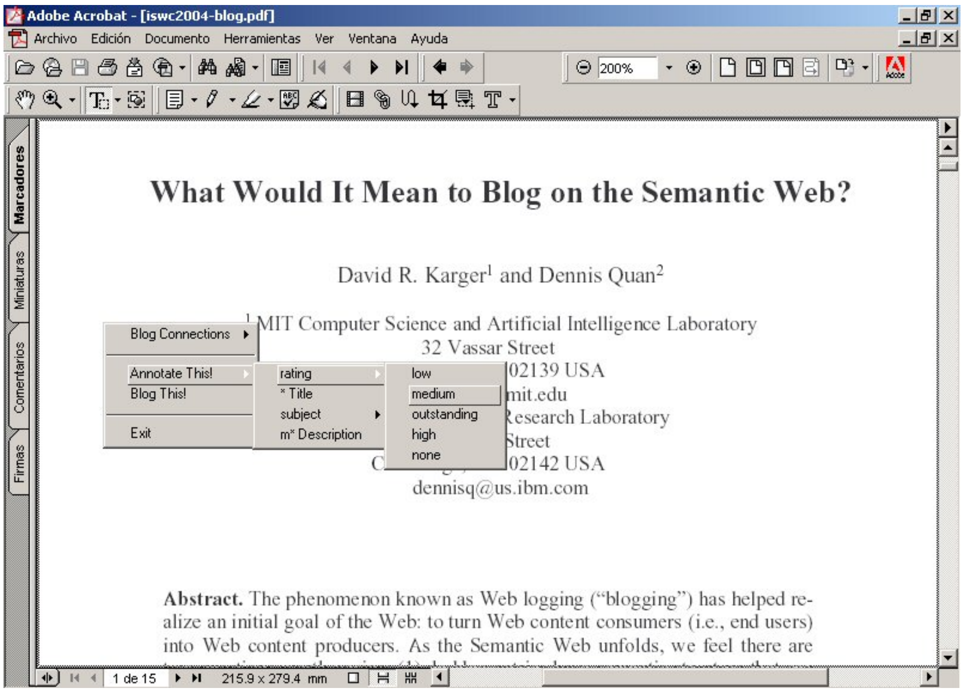
Figure 3.3: Document title annotation.

This process only occurs at configuration time. Once the configuration is set, the user deploys this configuration by just selecting the corresponding profile. The profile fixes where the *post* is going to be published, and which ontological terms are available for annotation during the *BlogThis!* process.

3.3.2 *BlogThis!* at *Blogouse*

Once a profile is selected, *Blogouse* is ready for guiding the annotation process through drop-down menus (see Figure 3.3). All the *post* properties (i.e., *title*, *description*, any *metadata*, etc.) are now obtained from the resource by selecting some text chuck, and, next, clicking on the appropriate menu. For instance,

- *title* is obtained by selecting some text chunk, and next, clicking on the menu (see Figure 3.3). Since *title* only admits one value, successive selections have a substitution effect. Moreover, this element is mandatory (denoted by the asterisk), therefore, the user has to annotate it before publishing content in the blog.



What Would It Mean to Blog on the Semantic Web?

David R. Karger[1] and Dennis Quan[2]

[1] MIT Computer Science and Artificial Intelligence Laboratory
32 Vassar Street
02139 USA
mit.edu
esearch Laboratory
treet
02142 USA
dennisq@us.ibm.com

Abstract. The phenomenon known as Web logging ("blogging") has helped realize an initial goal of the Web: to turn Web content consumers (i.e., end users) into Web content producers. As the Semantic Web unfolds, we feel there are

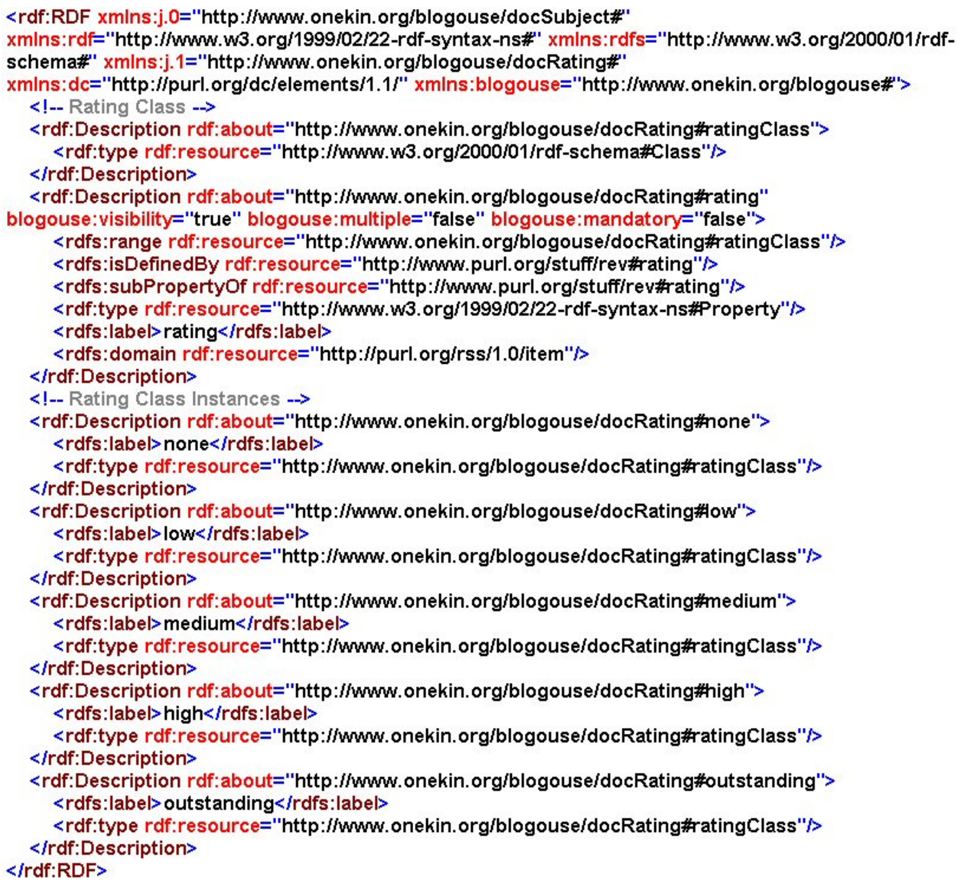
Figure 3.4: Document interest annotation.

- *description* is also obtained through selection. However, this element is multivalued, so that successive selections have a cumulative effect. This situation is denoted by prefix *"m"*.

- *subject* and *rating* are *metadata* elements which are directly provided by the user (see the namesake menu options in Figure 3.4). As the *metadata* depends on the blog at hand, this GUI is generated on the fly based on the blog profile. For instance, *rating* only admits five values, namely *"none"*, *"low"*, *"medium"*, *"high"*, and *"outstanding"*, whereas *subject* is restricted to hold *"Java"*, *"Web Service"*, and *"XML"*.

- *source* is automatically filled up by *Blogouse*. This element maintains a reference to a resource document from which the annotation is derived, in this way the annotation is decoupled from the document.

Figure 3.5 shows the syntax of the *rating* element in the ontology. In the definition of each element, the attributes *visibility*, *multiple*, and *mandatory* can be seen under the namespace *blogouse*. If the value of the attribute is true, then the attribute has an effect over the element (e.g. *visibility="true"* indicates the element is shown in



```
<rdf:RDF xmlns:j.0="http://www.onekin.org/blogouse/docSubject#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-
schema#" xmlns:j.1="http://www.onekin.org/blogouse/docRating#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:blogouse="http://www.onekin.org/blogouse#">
  <!-- Rating Class -->
  <rdf:Description rdf:about="http://www.onekin.org/blogouse/docRating#ratingClass">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.onekin.org/blogouse/docRating#rating"
blogouse:visibility="true" blogouse:multiple="false" blogouse:mandatory="false">
    <rdfs:range rdf:resource="http://www.onekin.org/blogouse/docRating#ratingClass"/>
    <rdfs:isDefinedBy rdf:resource="http://www.purl.org/stuff/rev#rating"/>
    <rdfs:subPropertyOf rdf:resource="http://www.purl.org/stuff/rev#rating"/>
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
    <rdfs:label>rating</rdfs:label>
    <rdfs:domain rdf:resource="http://purl.org/rss/1.0/item"/>
  </rdf:Description>
  <!-- Rating Class Instances -->
  <rdf:Description rdf:about="http://www.onekin.org/blogouse/docRating#none">
    <rdfs:label>none</rdfs:label>
    <rdf:type rdf:resource="http://www.onekin.org/blogouse/docRating#ratingClass"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.onekin.org/blogouse/docRating#low">
    <rdfs:label>low</rdfs:label>
    <rdf:type rdf:resource="http://www.onekin.org/blogouse/docRating#ratingClass"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.onekin.org/blogouse/docRating#medium">
    <rdfs:label>medium</rdfs:label>
    <rdf:type rdf:resource="http://www.onekin.org/blogouse/docRating#ratingClass"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.onekin.org/blogouse/docRating#high">
    <rdfs:label>high</rdfs:label>
    <rdf:type rdf:resource="http://www.onekin.org/blogouse/docRating#ratingClass"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.onekin.org/blogouse/docRating#outstanding">
    <rdfs:label>outstanding</rdfs:label>
    <rdf:type rdf:resource="http://www.onekin.org/blogouse/docRating#ratingClass"/>
  </rdf:Description>
</rdf:RDF>
```

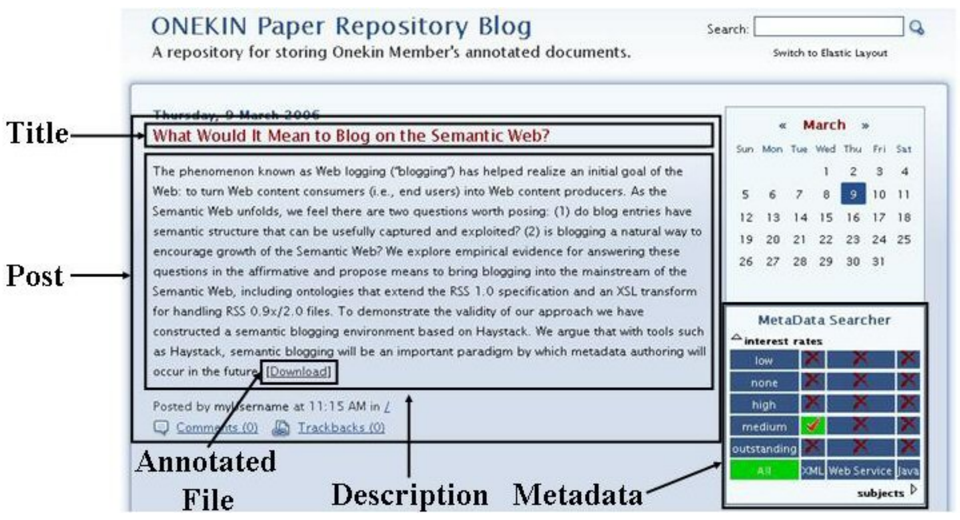
Figure 3.5: Rating ontology description.

the menu). The combination of *boolean* values in the attributes gives the semantics described for each element.

During this process, the (inconclusive) *post* is kept in a special clipboard. Once concluded, the user can publish the bright-new *post* by clicking on *BlogThis!*.

To avoid publishing a *post* by mistake, *Blogouse* provides two security systems. When user clicks on *BlogThis!*, the annotation of mandatory elements is checked. If an error occurs, an alert window is shown and the publication process is stopped. Once all mandatory elements are annotated, a preview window appears and displays the annotated content (similar to Google's *BlogThis!*). Finally, the content is published.

An option is also to upload the annotated document (i.e., the PDF file) as an additional property of the *post*. This facilitates document sharing through the blog. In this case, *BlogThis!* enacts two *XML-RPC* calls. Firstly, the annotated document



ONEKIN Paper Repository Blog
A repository for storing Onekin Member's annotated documents.

Search:
Switch to Elastic Layout

~~Thursday, 9 March 2006~~

What Would It Mean to Blog on the Semantic Web?

The phenomenon known as Web logging ("blogging") has helped realize an initial goal of the Web: to turn Web content consumers (i.e., end users) into Web content producers. As the Semantic Web unfolds, we feel there are two questions worth posing: (1) do blog entries have semantic structure that can be usefully captured and exploited? (2) is blogging a natural way to encourage growth of the Semantic Web? We explore empirical evidence for answering these questions in the affirmative and propose means to bring blogging into the mainstream of the Semantic Web, including ontologies that extend the RSS 1.0 specification and an XSL transform for handling RSS 0.9x/2.0 files. To demonstrate the validity of our approach we have constructed a semantic blogging environment based on Haystack. We argue that with tools such as Haystack, semantic blogging will be an important paradigm by which metadata authoring will occur in the future. [Download]

Posted by myUsername at 11:15 AM in /
Comments (0) Trackbacks (0)

« March »

Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

MetaData Searcher
interest rates

low	✗	✗	✗
none	✗	✗	✗
high	✗	✗	✗
medium	✓	✗	✗
outstanding	✗	✗	✗
All	XML	Web Service	Java

subjects

Figure 3.6: The annotated document as a *post*.

is uploaded to the blog, and the assigned URL is returned back. This URL is held by the *source* element, and added to the *post* description. This *post* is then also eventually uploaded, and published in the blog[4]. The result is displayed in Figure 3.6. Notice the *[Download]* anchor at the end of the *post* that is the GUI counterpart of the *source*. It is also worth highlighting the *metadata* matrix found on the lower, right-hand side corner. The same as the traditional calendar-based searches permit to locate *posts* based on their publishing date, this matrix permits locating *post* based on the *subject* and *rating metadata*. Each kind of *metadata* is shown as an axis of the matrix. Each cell shows an "V" to indicate whether at least one *post* has these *metadata*. In this case, when clicking on one of these cells, the blog filters the *posts* with the corresponding *metadata*. In other case, an "X" is shown. This is a straightforward use of *metadata*, but more sophisticated advantages can be drawn from *post* annotation (refer to [KQ04] for a detailed account).

3.4 Evaluation and discussion

An experiment has been conducted to use blogs for teaching material sharing and revision among six lecturers. Some lecturers were reluctant to participate as they look down blogs as mere diary's entries, and producing *posts* out of teaching ma-

[4]Implementation wise, resource upload is achieved through the *MetaWeblog* API whereas *post* upload uses *Blogger* API. To ensure transactional properties to these two operations, a rollback-like operation undoes the resource upload.

terial was quite labour intensive. On the other hand, most of them recognised the advantages of having a central hub where documents as well as *comments* can be widely and easily available. As pointed out in [Röl03] *"weblog can reduce the volume of e-mails received by the participants; reduce searching time looking for teaching material; effectively archive project documentation"*.

All participants agree these advantages were effective, and that *Blogouse* facilitated the ready publication of their desktop resource. Editor independence was regarded as the most outstanding feature of *Blogouse*. The variety of formats used for storing teaching material as well as the heterogeneous range of editors in such a free environment as the university, where staff is free to choose the editor he enjoys it most (e.g. *XMLSpy* for XML documents, *Visio* for UML diagrams, and so forth), makes this feature be really a must. Furthermore, participants appreciate the easiness of the interface provided by *Blogouse*.

On the down-side, the drop-down menus were found cumbersome. The annotation of some *metadata* requires three mouse clicks just for simple ontologies. This really poses a scalability problem for sophisticated ontologies where a longer "drill-down" process could be required to locate the appropriate concept.

Some authors note that semantic *metadata* should be produced *"as a by-product of tasks that a user is already used to perform on a day-to-day basis, such as entering people in an address book application, organizing events in a calendar or managing publications in a bibliographic database"* [Hen01, MD05], and that *"while the metadata added to a blog entry could in principle be hand coded or added through specific form fields in a web-based blog editor, we believe that this would be far too complicated to appeal to a non-technical user, like the average employee we are aiming at"* [MD05].

We agree on this statement. However, its feasibility is limited to structured resources (e.g. bibliographic entries, calendar entries, and the like) where wrapping techniques can be used to extract the appropriate *metadata* without user intervention. Also, it is well-known that most of the desktop resources are far from being structured, and annotation techniques and tooling are required here. In this sense, *Blogouse* offers a compromise between simplicity and more powerful approaches to *metadata* extraction.

3.5 Conclusions

This chapter describes how semantic annotation can be successfully applied to improve blog publication client tools. These improvements basically consist on an extension of mouse devices functionality, which they make the mouse devices ontology-aware. As a result, a document with its annotations (*metadata*) can be discussed and shared easily with the *participants*. As this point, annotation and document are completely decoupled, and the blog is which links them to give them a meaning.

Moreover, *Blogouse* has been designed to be user-friendly, editor-independent, and blog-independent, maintaining the blog arena simplicity. This makes *Blogouse* be a more general blog publication tool.

Chapter 4

Blog as Virtual Community Platforms

"No great improvements in the lot of mankind are possible until a great change takes place in the fundamental constitution of their modes of thought."
– *John Stuart Mill.*

4.1 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. These communities are frequently supported through blogs. One of its main proponents, Dion Hinchcliffe, states: *"most businesses should be actively exploring the use of blogs to communicate (two-way) with their customers, starting small, expecting little, and looking for opportunity"* [Hin07]. Studies endorse the use of blogs to market products, build stronger relationships with customers, and obtain customer feedback [N06, LD08]. This interaction between the company and its customers frequently pivots around the products in the company's catalogue. Such blogs are hereafter referred to as "catalogue blogs" (CBs). CBs aim to be a conduit for customer feedback as well as fostering community construction around products. However, CB development is hindered by the lack of common standards for blogs, the immaturity of blog technology, and the youth of virtual community development. This obstructs the development, the maintenance and the migration of CBs.

This chapter addresses the aforementioned situation through Model Driven Engineering (MDE) [SV06]. The main requirement is to develop CBs in a cost-effective way. *The challenge is then not on feasibility but cost effectiveness.* So far, the immaturity of blog technology prevents code and knowledge acquired for a given blog engine to be extrapolated to a different engine. This jeopardises migration and reuse which, in turn, hinders the fulfilment of the cost-effective mandate. MDE achieves reuse by introducing distinct models of a system at different levels of abstraction. 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. First, to the best of our knowledge, this chapter provides one of the few examples of using MDE for blog generation. Despite authoritative voices, such as that of D. Hinchcliffe, have long recognised the usefulness of blogs for community support, we are not aware of work on how such blogs can be developed. This chapter addresses "catablog construction" along MDE principles and, in so doing, provides concrete guidelines for both the models and the steps to be followed.

However, it is not just a question of showing that MDE-based blog development is technically feasible. Companies need some cost analysis that proves the benefits of MDE compared with current practices. This would be the second contribution. The chapter reports on labour-cost experiments comparing the MDE approach versus manual coding. The break-even point is measured in terms of both the catalogue size and the number of generated *catablogs* needed for the MDE infrastructure to pay off.

The rest of the chapter is organised as follows. A brief on the heterogeneity of blog engines is given in Section 4.2. Section 4.3 looks at blog construction as an MDE process instance. The (meta)models, i.e., the *Catalogue Model,* the *Content Model,* the *Style Model* are introduced in sections 4.4, 4.5 and 4.6, respectively. Section 4.7 addresses *Blojsom* as a Platform-Specific Model while the challenge of *Blojsom* evolution is the topic of Section 4.8. Section 4.9 discusses the approach, and the related work (Section 4.10) and conclusions (Section 4.11) end the chapter.

4.2 Blogs & Blog heterogeneity

A blog is basically a sequence of usually short, frequently updated *posts* that are arranged chronologically. A blog engine is a software for blog management[1]. This blog engine can take different forms, namely:

- *Standalone blogs*, where the engine is available for users to download and install locally. Here, blog entries are realised as either database tuples or folders in a given directory structure. *LifeType*[Lif], *b2evolution*[Fra] or *Blojsom*[Dav] are some examples of this type. Unfortunately, the database schema or the folder structure can differ greatly among engines.

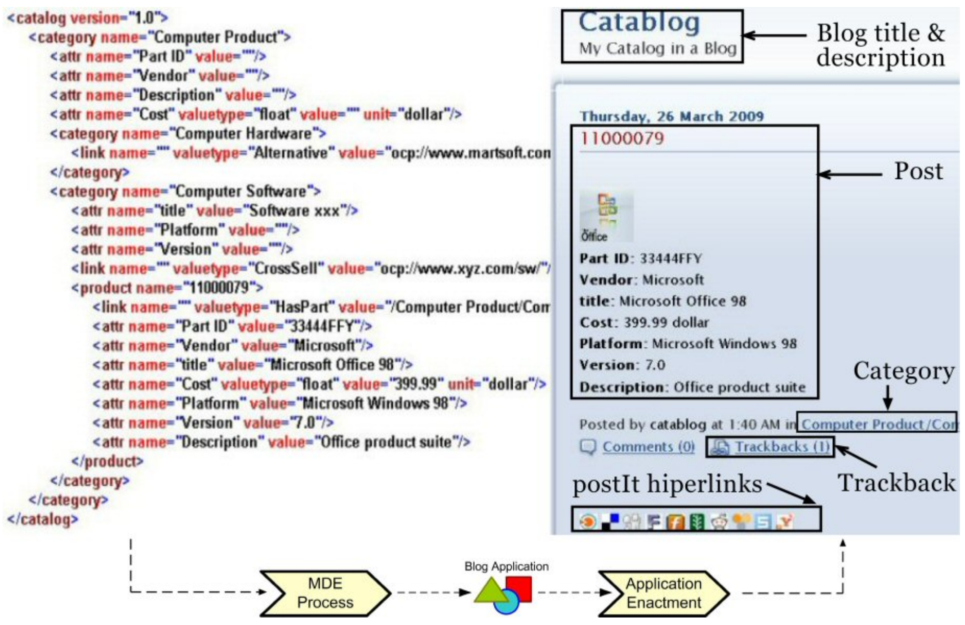
- *Hosted blogs*, where the engine resides remotely, and blog creation is achieved through API calls. *Movable Type*[Sixa], *Blogger*[Bloa], *LiveJournal*[Liv] or *Blojsom*[Dav] illustrate this approach. Unfortunately, current situation is characterised by lack of standards (the same functionality is termed differently), unstable APIs, 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*[2] and *comment* creation).

There is no one-size-fits-all solution for blog engines. The selection very much depends on the budget, infrastructure availability, and the control required over the final design. Different companies have distinct priorities for creating product communities, and their technological literacy and staff availability also influence the solution. Therefore, automating blog creation from product catalogues should take blog heterogeneity as a starting point. Decoupling from engine specifics looks as the only way out of this plethora of platforms which is constantly evolving and widening. In this setting, Model-Driven Engineering (MDE) emerges as a way to separate platform-independent design from platform-specific implementation. Next section introduces blog construction as an instance of the MDE process.

4.3 Blog generation as an instance of the MDE process

The goal is to obtain a "catalogue blog" from a catalogue specification using an MDE process (see Figure 4.1). The starting point is a specification of the catalogue

[1]Refer to [Gar05] for a detailed comparison of blog engines.
[2]Also known as external *comment*.



```xml
<catalog version="1.0">
   <category name="Computer Product">
      <attr name="Part ID" value=""/>
      <attr name="Vendor" value=""/>
      <attr name="Description" value=""/>
      <attr name="Cost" valuetype="float" value="" unit="dollar"/>
      <category name="Computer Hardware">
         <link name="" valuetype="Alternative" value="ocp://www.martsoft.con
      </category>
      <category name="Computer Software">
         <attr name="title" value="Software xxx"/>
         <attr name="Platform" value=""/>
         <attr name="Version" value=""/>
         <link name="" valuetype="CrossSell" value="ocp://www.xyz.com/sw/"/
         <product name="11000079">
            <link name="" valuetype="HasPart" value="/Computer Product/Con
            <attr name="Part ID" value="33444FFY"/>
            <attr name="Vendor" value="Microsoft"/>
            <attr name="title" value="Microsoft Office 98"/>
            <attr name="Cost" valuetype="float" value="399.99" unit="dollar"/>
            <attr name="Platform" value="Microsoft Windows 98"/>
            <attr name="Version" value="7.0"/>
            <attr name="Description" value="Office product suite"/>
         </product>
      </category>
   </category>
</catalog>
```

Catablog
My Catalog in a Blog

Thursday, 26 March 2009

11000079

Part ID: 33444FFY
Vendor: Microsoft
title: Microsoft Office 98
Cost: 399.99 dollar
Platform: Microsoft Windows 98
Version: 7.0
Description: Office product suite

Posted by **catablog** at 1:40 AM in Computer Product/Co

Comments (0) Trackbacks (1)

Figure 4.1: From *OCF* catalog to blog generation.

along the *Open Catalog Format* (OCF) standard [Mar]. The MDE process will then generate an application whose enactment delivers a blog. During this process, a large number of functional concerns emerge: what products to include, how they are related, how consumers interact, how information is made persistent, the 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. MDE provides a way to stratify these concerns by introducing distinct models of a system at different levels of abstraction [SV06]. In this case, the system is a "catalogue blog", and the following (meta)models are introduced (hereafter, capital M will denote (meta)Models):

- *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: the *Open Catalog Format* (OCF),

- *The CB Content Model*, a PIM which captures the content to be rendered through the blog. It can be automatically obtained from catalogue data,

- *The CB Style Model*, a PIM which collects those concerns that impact user interaction. It is to be provided by a domain expert,

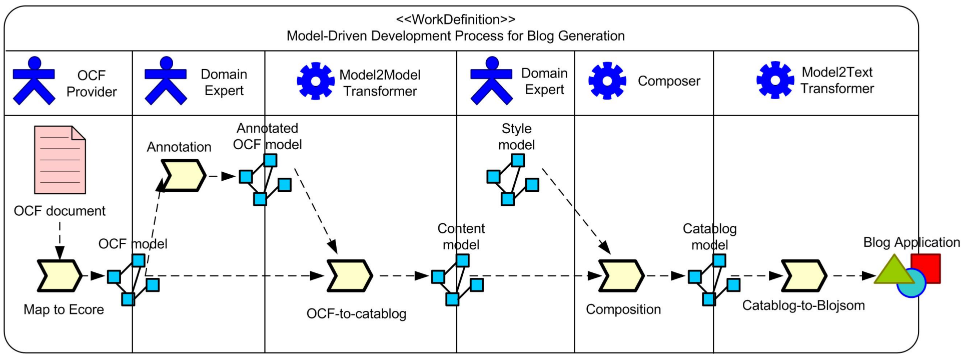


Figure 4.2: Blog generation as an instance of the MDE process.

- A *hosted blog platform*, a Platform Specific Model (PSM) for a hosted blog platform: *Blojsom* [Dav],

- A *standalone blog platform*, also a PSM for a *s*tandalone blog platform also in *Blojsom*.

These models describe the problem at different levels of abstraction. Model transformations are then used to map among models. 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. In this way, MDE strives to substitute manual coding by first modeling, then transforming.

Consequently, MDE focuses on the construction of *Models*, specification of *transformation* patterns, and the chaining of transformation (a.k.a. *the MDE process*). System development is conceived as transformation chains where the artifacts that result from each phase must be models. *SPEM (Software Process Engineering Metamodel)* is a notation for defining processes and their components whose constructs can be described in *UML* notation [OMG05]. Hereafter, *SPEM* terminology is used to specify the milestones, roles and dataflow that go with producing a blog from a catalogue description through a chain of model transformations.

The *OCF-to-Blojsom* process is depicted as an *SPEM* process in Figure 4.2. Three actors are distinguished: *the transformer* (i.e., the software that maps from source model to a target model), *the composer* (i.e., the software that takes two models as input and delivers a single model) and *the domain expert*.

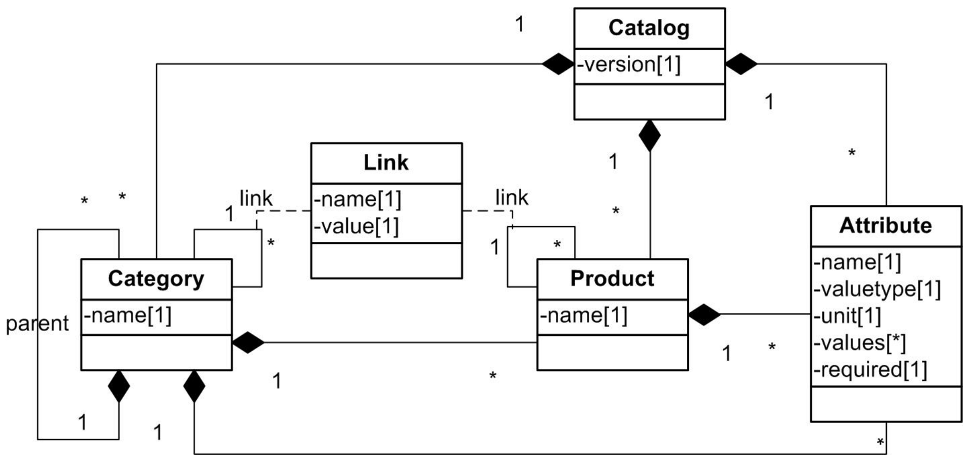


Figure 4.3: The *OCF Model.*

The process starts by mapping the XML-based *OCF* file into *Ecore* (i.e., the Eclipse-based realisation of MOF[3]) that can now be processed as a model. This *OCF* model serves to obtain the *Content Model.* However, some ambiguities exist about how *OCF* product attributes are to be mapped to properties of the *Content Model.* The *domain expert* 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. The outcome is an *Annotated OCF model* (see Figure 4.2). This annotated model is then used to obtain the *Content model.* Content is next complemented by the *Style model* provided by the *domain expert.* Together they serve to obtain a full-fledged *catablog* model. Finally, a model-to-text transformation outputs the code that realises the *catablog* for a specific PSM. Figure 4.2 serves as a road map for the next sections.

4.4 The Catalogue Model

The *Catalogue Model* is based on OCF. OCF is readily transformable to other major catalog standards like *xCBL, Punchout, xCBL, CIF, CUP, cXML, OCI* or *Rosettanet.* This work takes OCF version 1.0. Figure 4.3 describes the main notions behind OCF: **catalog**, which consists of a hierarchy of product categories; **category**, which contains attributes, parameters, links, products and subcategories.

[3]Meta-Object Facility

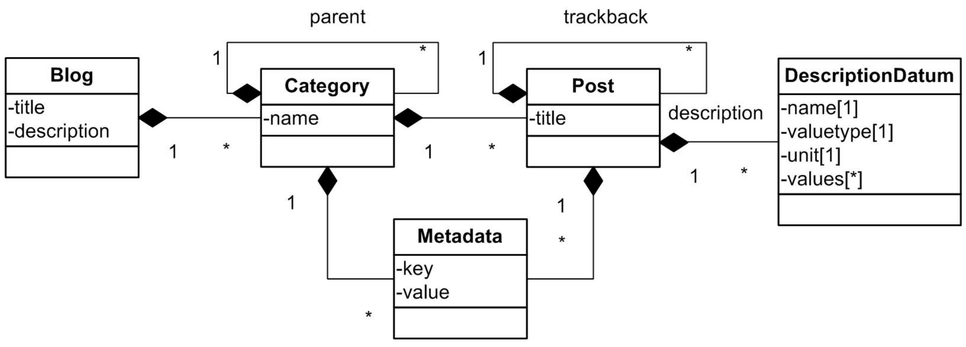


Figure 4.4: The *Content Model.*

Each category has a name. A category defines a set of attributes which specify 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*.

4.5 The Content Model

The *Content Model* is an attempt to abstract away from the peculiarities of how content is supported in different blog engines. Basically, this content is arranged along the following constructs (see Figure 4.4): ***blog*** as the content root*,* ***post*** as a blog entry*,* ***category*** as the means to classify *posts*, ***metadata*** as content not intended to be rendered but needed for additional processing of *posts* (e.g. for indexing or querying purposes), and finally, ***descriptionDatum,*** which keeps product attributes that will later be used to generate the content of the *post*[4]. Notice that *comments* are left outside this model since they are not generated from the catalogue, but dynamically provided by end users.

[4]A *Description Datum* is basically a *(name,value)* pair. Additionally, the "valuetype" and "unit" are also collected from the catalogue due to their potential impact on the rendering of this datum.

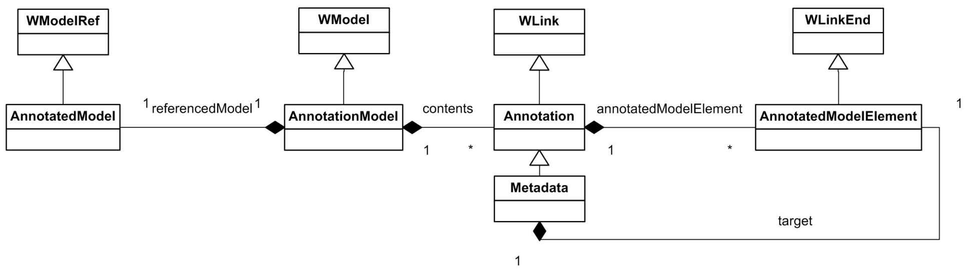


Figure 4.5: *AMW Model* Extension.

These classes can hold distinct associations (see Figure 4.4), e.g. the "parent" association, that captures *category* hierarchies; the *trackback* association, that reflects the namesake relationship between *posts*, etc. This *Content Model* provides the hook for adding style hints.

4.5.1 Obtaining the Content model

Content models are obtained from *OCF models*. Some mappings are straightforward: a *Catalog* element delivers a *Blog* element; an *OCF Category* is mapped into a *Category*; an *OCF Product* outputs a *Post*. However, there exists some ambiguity about how *OCF Attributes* should be transformed. Specifically, *OCF attributes* can be mapped into either *descriptionDatum* or *metadata* in the *Content Model*. For instance, *vendor* is a product attribute in the *OCF Model*. When mapped to the *Content Model*, this attribute can be used to generate the description of the *post*. Additionally, it can also play the role of *metadata* in the sense that *vendor* can be used for indexing *posts* so that you can render the *posts* that pertain to a given vendor. Therefore, the *Domain Expert* should provide hints about how transformation should proceed.

The *Domain Expert* annotates *OCF* attributes indicating whether they will become *metadata* or *descriptionDatum* in the *Content Model*. Annotations are not relevant to the *OCF model* itself. To prevent the *OCF model* from being polluted, a distinct model is used to collect these decisions. Along the lines described in [VdCdFM08], 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 [dFBV06]. Figure 4.5 shows the Weaving (meta)Model.

```
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();
```

Figure 4.6: *OCF_to_catablog* 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.

Now transformation can proceed. Annotations (i.e., the Weaving model) are consulted during transformation through helper functions. Figure 4.6 provides a case in point. This transformation rule invokes a helper function to ascertain whether a given *OCF attribute* will become *metadata*. The function looks for *metadata* annotations whose *target* element coincides to the attribute being passed as a parameter (identified by the *_xmiID_* property).

4.6 The Style Model

The design of this metamodel is based on two observations. First, catalogue blogs (CBs) differ from diary-like blogs in their purpose, i.e., supporting virtual communities. Unlike diary-like blogs, CBs should introduce means for community building but without losing blog simplicity. The second observation is that blog simplicity is obtained at the cost of reducing the design space. Blog engines restrict how content can be presented and navigated. Specifically, blogs are characterised by (1) content being arranged in terms of *post*, (2) navigation being limited to *category*-based and chronology-based search, and (3) presentation being mainly based

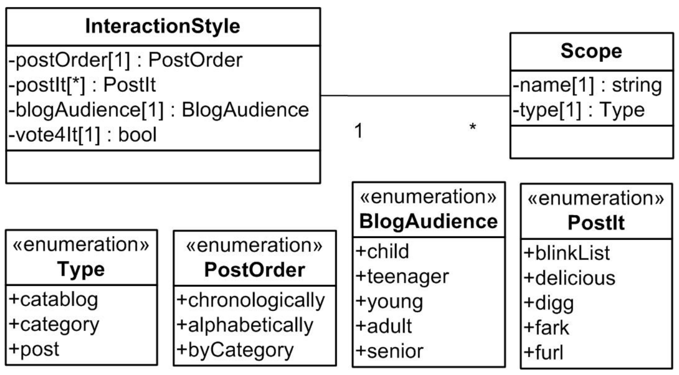


Figure 4.7: The *Style Model*.

on *themes*. Of course, more sophisticated forms of presentation and navigation could be envisaged but this would have infringed the simplicity principle.

Therefore, introducing, let's say, general-purpose navigation models as those provided by *WebML* [BCFM07] or *OO-H* [GCP01] would have been of limited use here, since some features would have *not* been possible to be realised through blog engines. Blog engines, although PSMs, provide a more abstract platform than general-purpose Web frameworks (e.g. *Struts*). As a result, we did not come up with a meta-model and then, looked at how this model could have been mapped down to technological platforms. Rather, the approach was the other way around. We departed from existing blog engines, and strove to find a set of style parameters that abstract away from specifics of how rendering is achieved in distinct blog engines.

Based on these two observations, we introduce the *Style Model*. We get inspiration from the cascading-style-sheet (CSS) way of handling rendering [W3C]. CSS classes permit to associate a name to a preset collection of rendering parameters. These parameters are low level (e.g. background color, fonts and the like). The *Style Model* abstracts away for low-level rendering details (diversely realised by blog engines) into a set of criteria adapted to CBs. Therefore, we do not pretend this model to be general but domain specific, i.e., tuned for blogs supporting community building around product catalogues.

Figure 4.7 depicts the *Style Model*. A set of parameters are provided that define the so-called *InteractionStyle*, namely: *postOrder*, *blogAudience*, *postIt* and *vote4It*. The first two parameters, *postOrder* and *blogAudience*, look at CBs as blog abstractions by describing design criteria that guide blog navigation and presentation. On the other hand, *postIt* and *vote4It* capture blogs as community conduits. Next, an *InteractionStyle* applies within a ***Scope***. A *Scope* has a *name* and *type*. The *type* range include, *catablog* (which affects the whole blog), *category* (which impacts all products of this category) and *post* (which is restricted to a single product). An order is defined among scopes (*post* < *category* < *catablog*) so that parameters defined at one scope can be overridden by lower level scopes[5]. It is worth noticing that scopes with a *catablog* or *category* type should have their counterpart at the *Content model* (i.e., the scope's *name* should be found at the *Content model*). Next paragraphs introduce each parameter of the *Interaction Sytle*.

postOrder. It indicates the order in which *posts* can be arranged in the index page of the blog. Three values are included, namely, *chronologically* (i.e., *posts* are ordered by their creation time), *alphabetically* (by *post* title) or *byCategory* (i.e., ordered by the *post category*). The scope of this property is the whole blog.

blogAudience. Blogs use *themes* for rendering. Each blog engine has its own collection of *themes*. This makes *theme* a platform-specific concern. 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. This is the role of *blogAudience*. This property captures the expected age of users as the main criteria for *theme* selection. Five values are considered: *child, teenager, young, adult* and *senior*.

postIt. The rationale for this property is based on the increasing popularity of social bookmarking. Tagging sites, such as *delicious*, *digg*, *fark*, *newsvine*, *reddit*, *simpy* or *spurl*, permit to share URL-addressable resources. Since products are now realised as blog entries (hence, URL addressable), every new *post* in a blog is actually a new page which has a *permalink*. This permits the indexing of *post* by search engines (e.g. potential customers can come across with your product *posts* through Google), and the affixing of blog entries into tagging sites (which facilitates the sharing of product information with a wider audience). 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 4.1). For instance, by clicking on the *digg* icon, the content of the entry is published at *digg*. So far, five tagging sites are considered: *blinkList*, *delicious*, *digg*, *fark* and *furl*.

vote4It. This property captures a new means for community building. In a blog setting, participation is realised through commenting on a blog *post*. So far,

[5]It can potentially be possible to define scopes based on queries over the catalogue so that the style applies to those products meeting the query criteria. This option is left for future developments.

commenting requires users to access directly the CB. 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 CB.

As an example, consider that a customer prefers to comment on your products at his own website, rather than accessing your CB. For instance, this customer can write the following HTML snippet at his website:

> *We have successfully installed <a href="http://.../catablog/Computer Software/Acrobat2.3.html" rev="vote-for">Acrobat 2.3</a>, after all difficulties with <a href="http://.../Computer Software/Taborca3.2.html" rev="vote-against">Taborca 3.2</a>*

The two anchors refer to products at your CB by including their *permalinks*. The question is how your CB can be aware of such external references. The answer is through *trackbacks*. This mechanism enables websites communicate via “pings”, where each ping informs the blog that the sending site has made a reference to a *post* on the blog [Six04]. Having a *permalink*, products can be referenced from other websites, i.e., *comments* on this product can appear outside the blog. Instead of forcing customers to comment only at the CB place, customers can now simply send a “ping” to your CB 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.

Trackback counting can then be used to measure the popularity of products. *Trackbacks* support 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. Even more, as the vote is at your site, you can change your vote at any time, and the voting is re-calculated. This is in sharp contrast with current pull approaches to voting, where you need to go to the place where the product is described, tick some rank boxes, and then, lose the control over your vote.

However, *trackbacks* just indicate that there is a reference to the *post*, but not its intention. Such intention can be expressed through the *VoteLink* microformat [Me05]. Microformats are defined as embedded annotations into HTML markup that convey metadata, i.e., describing what the content is about. For instance, the *VoteLink* microformat proposes a set of three new values for the *rev* attribute of the <*a*> tag in HTML. The new values are *“vote-for”*, *“vote-abstain”* and *“vote-against”*, which are mutually exclusive, and represent agreement, abstention, and disagreement, respectively. Now, a hyperlink (i.e., the <*a*> markup) not only points to an URL but also conveys opinions about your likings w.r.t. what this URL stands for. In our case, these URLs stand for products (i.e., *permalinks* on *posts* that represent products). A customer commenting at his company website can now include a link to your CB but now the link can be annotated with his likings (i.e., vote) to the linked product in a machine-understandable way. The previous HTML sample uses this approach to convey the likings of the customer about two products: *Acrobat2.3* and *Taborca3.2*. This feature is captured in the *Style Model* through the ***vote4It*** property. This property holds a *boolean* that indicates whether this feature is to be present in the CB (see [Me05] for supporting *VoteLink*-aware *trackbacks*).

4.6.1 Obtaining the Style model

The *Domain Expert* provides himself the *Style model*. The *postOrder*, *blogAudience* and other *Style* parameters are then consulted when the blog application is generated so that the right *theme*, order of *posts*, tagging hyperlinks, etc. are obtained for the blog engine at hand. These properties are, by no means, the only criteria that can guide, for instance, the selection of the *theme* for the blog application. Other aspects to be considered during *theme* selection can include demographic (e.g. sex, age, education, etc.), geographic, attitudinal (e.g. interest in lifelong learning) or behavioral (e.g. product usage rate) data [BU]. This very much depends on the importance of these factors on the market at hand.

The bottom line is that the *Style Model* explicitly captures design criteria through properties. And even more important, the impact that these criteria have, or better said, the impact that the *combined interaction* of these criteria have in the aesthetic of the blog are specified as transformation rules. 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 blog rendering, and, in so doing, they are true repositories of design expertise.

Finally, the *Style model* is composed with the *Content model* to deliver a full model of the blog. Model composition is achieved on the grounds that *Style scopes* should find their counterpart as either the *catablog*, *categories* or *posts* of the *Content model*. That is, scopes' names should coincide with the name/title of objects in the *Content model*. This is achieved through AMW[6]. Once the *Content model* and the *Style model* are composed, all it is ready to generate the code.

4.7 Blojsom as a PSM

Blog engine heterogeneity firstly motivates this work. A major distinction is that of standalone blogs *versus* hosted blogs. Standalone blogs are those where the engine is available for users to download and install. Here, blog creation involves the population of a database or the creation of a folder structure that records blog data. On the other hand, hosted blogs are those where the engine resides remotely, and blog creation is achieved through API calls.

This work focuses on *Blojsom* [Dav]. Even within a single vendor like *Blojsom* different alternatives exist: file-based blogs (*Blojsom 2.x*), database blogs or hosted blogs *(Blojsom 3.x* through *Blogger API 1.0).* Next paragraphs address the mapping into different PSMs.

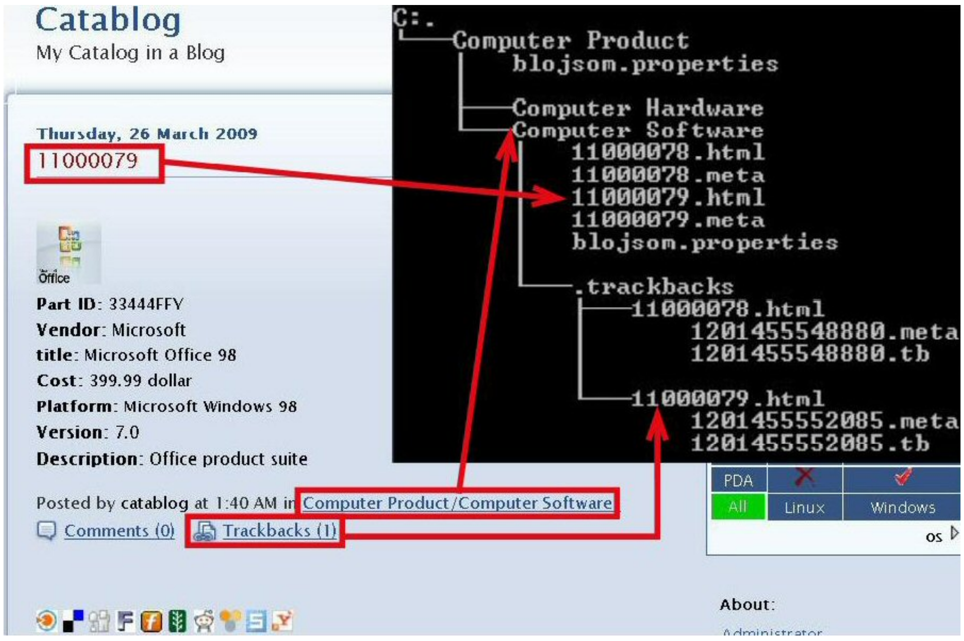
Standalone platform. Standalone platforms store data locally. Two options are available: files and databases. File-based blogs rely on both folders and files to store blog elements. Figure 4.8 outlines the file structure for our sample case. The mapping goes as follows:

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

- **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 instance, the *metadata* of the *category Computer Software* is located at */Computer Software/blojsom.properties*.

- **Posts** are supported as text files with *html* extension. The file extension is after the *post* title. The content includes the title of the *post* (first line), and the description of the *post*. These files are kept inside the *post category* folder.

[6]The composition process goes as follows. First, a weaving model captures the links between the input model elements, for instance indicating that *Post* (from the *Content* model) and *Scope* (from the *Style* model) whose name coincides should be combined into *Post*. Second, the weaving model is used to generate a transformation that achieves the composition.



Catablog

My Catalog in a Blog

Thursday, 26 March 2009

11000079

Part ID: 33444FFY
Vendor: Microsoft
title: Microsoft Office 98
Cost: 399.99 dollar
Platform: Microsoft Windows 98
Version: 7.0
Description: Office product suite

Posted by catablog at 1:40 AM in Computer Product/Computer Software

Comments (0) Trackbacks (1)

```
C:.
└───Computer Product
    │   blojsom.properties
    │
    ├───Computer Hardware
    └───Computer Software
        │   11000078.html
        │   11000078.meta
        │   11000079.html
        │   11000079.meta
        │   blojsom.properties
        │
        └───.trackbacks
            ├───11000078.html
            │       1201455548880.meta
            │       1201455548880.tb
            │
            └───11000079.html
                    1201455552085.meta
                    1201455552085.tb
```

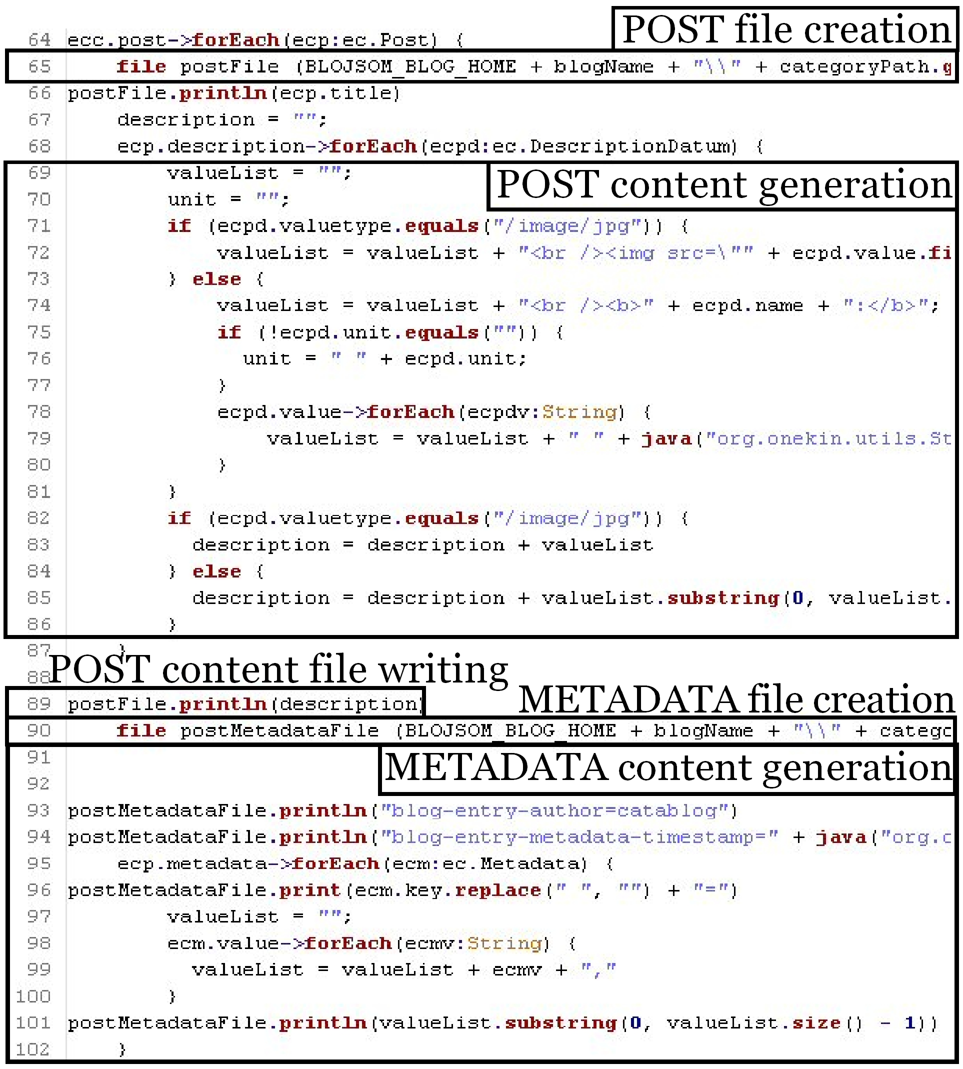
Figure 4.8: Blog directory structure and its rendering counterpart: *posts*, *categories*, *metadata* are all kept in files.

- **Post metadata** is mapped to a *.meta* file. The file extension is after the *post* title. 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 category* folder.

- **Trackbacks** are supported as *.tb* files, named after the *trackback* creation timestamp. *Post trackbacks* are collected in a folder that is misleadingly named after the *post* file (e.g. *"myPost.html"* but now as a folder name). Additionally, folders related with *posts* belonging to the same *category* are kept under the *.trackback* folder. Finally, this folder hangs from the *category* folder to which the corresponding *post* belongs. An example follows: *categoryA/.trackback/myPost.html/1201455552085.tb*

- **Trackback 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 4.9 provides a *MOFScript* snippet for creating files for *posts* and their corresponding *metadata*. For each *post*, an html-typed namesake file is created (line 65).



```
 64 ecc.post->forEach(ecp:ec.Post) {
 65     file postFile (BLOJSOM_BLOG_HOME + blogName + "\\" + categoryPath.g
 66 postFile.println(ecp.title)
 67     description = "";
 68     ecp.description->forEach(ecpd:ec.DescriptionDatum) {
 69         valueList = "";
 70         unit = "";
 71         if (ecpd.valuetype.equals("/image/jpg")) {
 72             valueList = valueList + "<br /><img src=\"" + ecpd.value.fi
 73         } else {
 74             valueList = valueList + "<br /><b>" + ecpd.name + ":</b>";
 75             if (!ecpd.unit.equals("")) {
 76               unit = " " + ecpd.unit;
 77             }
 78             ecpd.value->forEach(ecpdv:String) {
 79                 valueList = valueList + " " + java("org.onekin.utils.St
 80             }
 81         }
 82         if (ecpd.valuetype.equals("/image/jpg")) {
 83           description = description + valueList
 84         } else {
 85           description = description + valueList.substring(0, valueList.
 86         }
 87     }
 88
 89 postFile.println(description)
 90     file postMetadataFile (BLOJSOM_BLOG_HOME + blogName + "\\" + catego
 91
 92
 93 postMetadataFile.println("blog-entry-author=catablog")
 94 postMetadataFile.println("blog-entry-metadata-timestamp=" + java("org.c
 95     ecp.metadata->forEach(ecm:ec.Metadata) {
 96 postMetadataFile.print(ecm.key.replace(" ", "") + "=")
 97         valueList = "";
 98         ecm.value->forEach(ecmv:String) {
 99           valueList = valueList + ecmv + ","
100         }
101 postMetadataFile.println(valueList.substring(0, valueList.size() - 1))
102     }
```

Figure 4.9: *MOFScript* snippet that outputs code for the file-based *Blojsom* platform. (right truncation).

This *post* is associated with the *category* specified as the “ecp” parameter. Along *Blojsom* guidelines, a *post* file contains first a title (line 66) and, next, the content of the *post*. This content is obtained by formatting *DescriptionDatum* elements (lines 69-86). Once a *post* file is created, its associated *metadata* file needs to be generated. Line 90 creates the *metadata* file. Its content includes the creator and the timestamp for the associated *post* (lines 93-94). The other *metadata* is obtained from *Metadata* elements of the blog model (lines 95-102).

Rather than files and folders, *Blojsom* also permits to store blog data as tuples

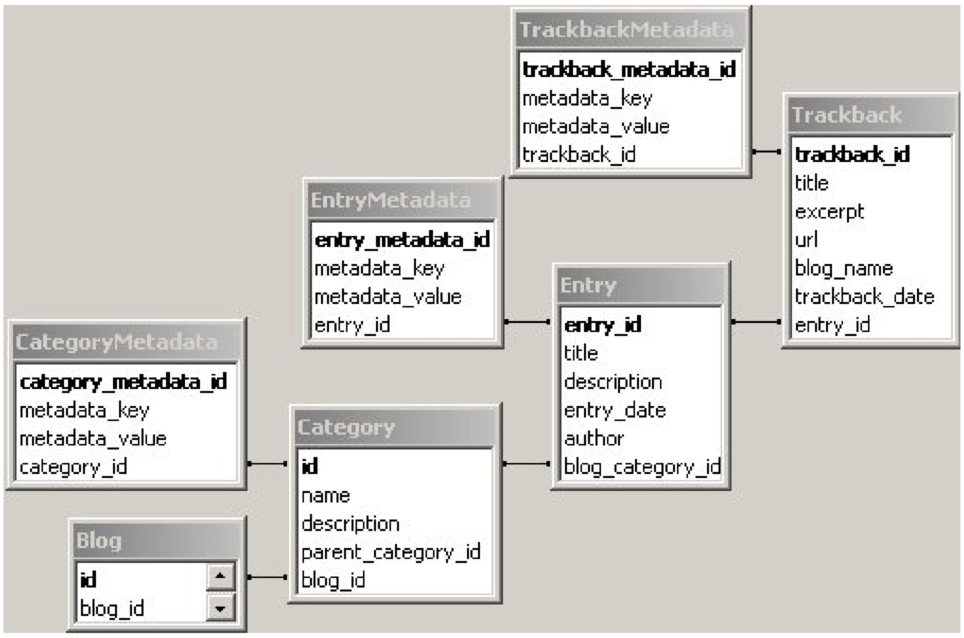


Figure 4.10: *Blojsom* database schema.

in a local database. Figure 4.10 depicts the database schema for *Blojsom*. This involves a different *MOFScript* transformation. Now, SQL scripts are generated that basically maps elements of the blog model into tables of the *Blojsom* database schema.

Hosted platform. This option permits the blog to reside remotely. API requests are then used to manage the blog online. Although there is not such thing as a standard for blog APIs, there is some consensus about the functionality to be supported. To the best of our knowledge, this functionality is captured by *Blogger API 1.0* [Eva03], *MetaWeblog API* [Dav02], and parts of the *MovableType API* [Mov05]. Many blog systems understand the *Blogger API*, including *Blogger* [Bloa], *RadioUserland* [Rad], *MovableType* [Sixa], and *Blojsom* [Dav]. All these APIs build on top of the XML-RPC protocol.

Unfortunately, current APIs fall short to generate the whole blog. APIs are thought to populate the blog but do not contemplate blog creation. Specifically, current APIs do not allow for creation of the blog itself as well as *category* and *metadata* definition. The former limitation is overcome through a script that mimics user actions when introducing new information using the blog back-end web editing tool (through HTTP calls). The second drawback (i.e., *category* and *metadata* remote insertion) is addressed through an *ad-hoc* XML-RPC plugin. It is worth noticing how this forces the *MOFScript*-generated application to handle two different protocols to create the blog.

To sum up, three different *MOFScript* applications were developed to handle the specifics of each target platform. Unfortunately, this *MOFScript* code is too exposed to *ad-hoc* details of *Blojsom*, making the code too expose to *Blojsom* evolution. Next section addresses this issue.

4.8 Facing PSM evolution

MDE suggests to create a dedicated mapping for each PSM. This is basically the approach described in the previous section. Three *MOFScript* applications cope with the idiosyncrasies of each *Blojsom* platforms (i.e., file-based, database-based and API-based). However, PSMs can be a moving target. For immature domains, PSMs tend to suffer frequent updates (new versions) which will more likely impact the transformations. For our sample case, transformations would need to be rewritten in at least three scenarios: changes in the file structure, updates of the database schema or modifications in the *Blojsom* APIs. The likelihood of such revisions much depends on the stability of the domain. Unfortunately, the current panorama is characterised by a plethora of heterogeneous blog engines where standards are far from being yet set. Therefore, it can be expected that future releases of *Blojsom* will gradually support standards as they emerge. The question is how to shield *MOFScript* code from those changes.

To this end, it is resorted to the notion of "abstract platform". An abstract platform defines an acceptable or, to some extent, ideal platform from the point of view of the application developer [ADvP04]. The aim is two fold. Firstly, the abstract platform permits *MOFScript* developers to describe the mapping in more abstract terms. Secondly, it shields *MOFScript* code from changes in the underlying PSMs whose specifics are encapsulated as part of the implementation of the abstract platform.

Therefore, an abstract platform has a dual nature. As a platform, it provides an executable environment. As an abstraction, it factors out the details of the set of existing platforms. That is, it represents a higher degree of platform-independence than that of existing platforms. Indeed, an abstract platform is defined w.r.t the existing platforms it abstracts from. Abstract platforms stand inbetween PIM and PSM. The aim is to ease PSM mappings when either the PIM-PSM gap or the PSM likelihood of evolution are important. Thus, abstract platforms are not PIMs but

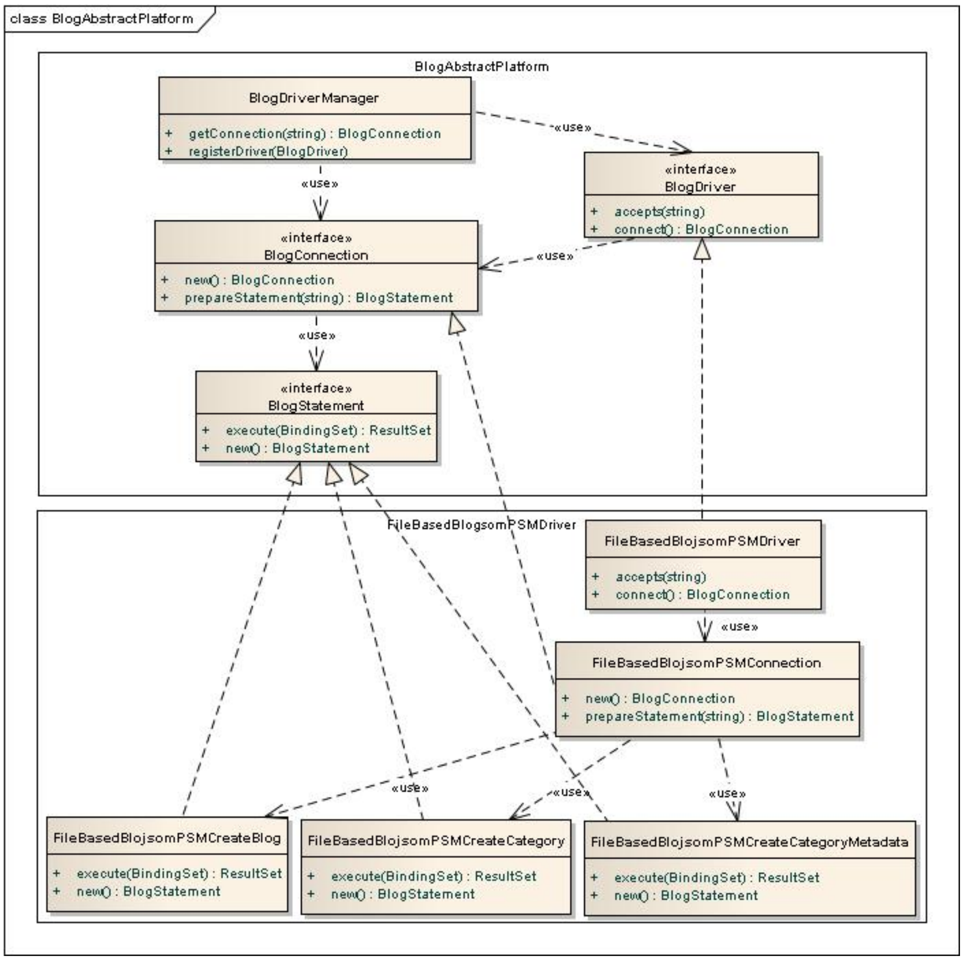

class BlogAbstractPlatform
BlogAbstractPlatform
BlogDriverManager
+ getConnection(string) : BlogConnection
+ registerDriver(BlogDriver)
«use»
«interface» BlogDriver
+ accepts(string)
+ connect() : BlogConnection
«interface» BlogConnection
+ new() : BlogConnection
+ prepareStatement(string) : BlogStatement
«interface» BlogStatement
+ execute(BindingSet) : ResultSet
+ new() : BlogStatement
FileBasedBlogsomPSMDriver
FileBasedBlojsomPSMDriver
+ accepts(string)
+ connect() : BlogConnection
FileBasedBlojsomPSMConnection
+ new() : BlogConnection
+ prepareStatement(string) : BlogStatement
FileBasedBlojsomPSMCreateBlog
+ execute(BindingSet) : ResultSet
+ new() : BlogStatement
FileBasedBlojsomPSMCreateCategory
+ execute(BindingSet) : ResultSet
+ new() : BlogStatement
FileBasedBlojsomPSMCreateCategoryMetadata
+ execute(BindingSet) : ResultSet
+ new() : BlogStatement

Figure 4.11: *Blog Abstract Platform* (partial view).

provide some kind of platform-independence for a set collection of PSMs. Next subsections present how these ideas are realised for the blog case.

4.8.1 The Blog Abstract Platform

Defining an abstract platform starts by identifying the existing platforms to abstract from. In our case, this includes: *file-based Blojsom PSM, database Blojsom PSM* and *hosted Blojsom PSM*. Next, platform-independence is defined based on these three PSMs. The peculiarities on handling blog creation among these three platforms are factored out into the abstract platform. This implies defining the architecture and operations over the abstract platform. Figure 4.11 outlines the architecture of the *Blog Abstract Platform*.

The architecture is inspired by how database applications are isolated from the peculiarities of Database Management Systems along the lines of ODBC and JDBC. A set of abstract operations are introduced to handle blog data no matter how these data end up being stored, i.e., either through hosted or standalone blogs. Specifically, the following operations are introduced: *createBlog()*, *createCategory*, *createCategoryMetadata()*, *createPost()*, *createPostMetadata()*, *createTrackback()* and *createTrackbackMetadata()*.

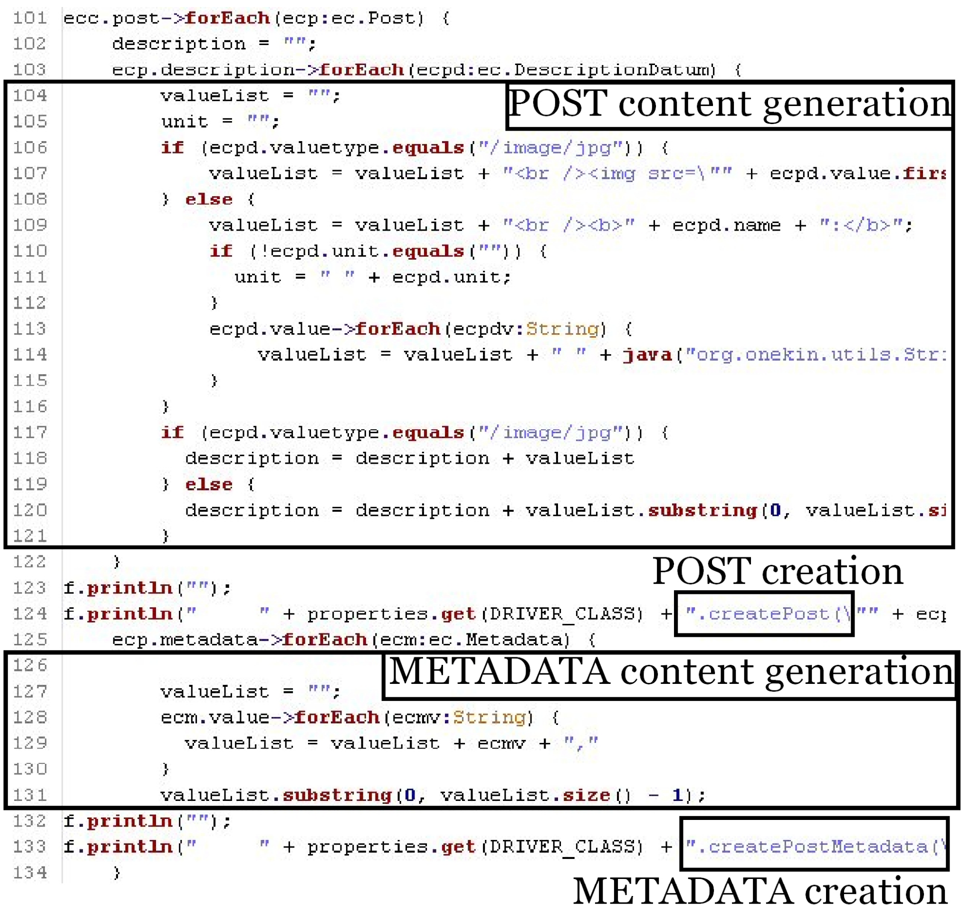
Next, *BlogDrivers* map these operations into the specificities of each blog engine. A *BlogDriver* encapsulates the peculiarities of the blog PSM at hand (e.g. protocol, data format, etc.). The management of *BlogDrivers* is realised by a *BlogDriverManager* component, which hosts the drivers, and supports the interaction with the distinct PSMs using a *BlogConnection* component.

This architecture permits to introduce new blog engines by specialising the *BlogDriver*, *BlogConnection* and *BlogStatement* interfaces. Figure 4.11 depicts this situation for the *fileBasedBlojsomPSM* platform. In this case, the driver describes how operations on the abstract platform (e.g. *createPost()*) are implemented in terms of files. This is supported by realising the *BlogStatement* interface, e.g. *fileBasedBlojsomPSMCreatePost* class. This class encapsulates the protocol and parameter details to achieve *post* creation for the *fileBasedBlojsomPSM*. The current implementation also provides drivers for the *databaseBlojsomPSM* and the *hostedBlojsomPSM*.

Once the abstract platform is in place, *MOFScript* can generate code for this platform. Next subsection outlines an example.

4.8.2 Mapping onto the Blog Abstract Platform

Applications can be generated over the *Blog Abstract Platform*. This implies that the application is described in terms of "abstract operations" (e.g. *createPost())* rather than describing how this operation is finally realised in a target platform (e.g. creating a file for the *post*). As an example, consider a *MOFScript* snippet for creating *posts* and their corresponding *metadata* out of blog models (see Figure 4.12). This script generates code for the *Blog Abstract Platform* rather than for a particular blog engine. Indeed, lines 124 and 133 generate calls to *createPost()* and *createPostMetadata(),* respectively. This is in contrast with the previous script in Figure 4.9 where the details of how the platform handles *post* and *metadata* are exposed in the transformation. That is, notions, such as files or tables, that were explicit in the previous version of *MOFScript* code (see line 65 in Figure 4.9),



```
ecc.post->forEach(ecp:ec.Post) {
    description = "";
    ecp.description->forEach(ecpd:ec.DescriptionDatum) {
        valueList = "";
        unit = "";
        if (ecpd.valuetype.equals("/image/jpg")) {
            valueList = valueList + "<br /><img src=\"" + ecpd.value.firs
        } else {
            valueList = valueList + "<br /><b>" + ecpd.name + ":</b>";
            if (!ecpd.unit.equals("")) {
              unit = " " + ecpd.unit;
            }
            ecpd.value->forEach(ecpdv:String) {
                valueList = valueList + " " + java("org.onekin.utils.Str:
            }
        }
        if (ecpd.valuetype.equals("/image/jpg")) {
          description = description + valueList
        } else {
          description = description + valueList.substring(0, valueList.si
        }
    }
f.println("");
f.println("      " + properties.get(DRIVER_CLASS) + ".createPost(\"" + ec
    ecp.metadata->forEach(ecm:ec.Metadata) {

        valueList = "";
        ecm.value->forEach(ecmv:String) {
          valueList = valueList + ecmv + ","
        }
        valueList.substring(0, valueList.size() - 1);
f.println("");
f.println("      " + properties.get(DRIVER_CLASS) + ".createPostMetadata(\
    }
```

Figure 4.12: *MOFScript* snippet that outputs code for the *Blog Abstract Platform*.

are now hidden by the abstract platform. These details are encapsulated in the *BlogDriver*.

The benefits of this approach include:

- Legibility. This stems from reducing the gap between the source PIM and the (abstract) platform. The mapping focuses on how model elements become *post*/*metadata* rather than struggling with PSM specifics. *MOFScript* code is now specified in more abstract terms.

- Robustness. *MOFScript* code is shielded from the internal design of the *Blog Abstract Platform* as well as from evolution in the APIs or database schemas of *Blojsom*.

- Reuse. The very same *MOFScript* code can be reused for distinct target

PSMs. The example in Figure 4.12 can generate either file-based blogs, database blogs or hosted blogs by just changing the driver. The driver and other connection parameters (e.g. user name, password for blog connection, etc.) are provided as configuration parameters at the time the *MOFScript* script is enacted.

- Portability. New blog engines can be introduced by just realising the interfaces of the abstract platform. That is, the very same *MOFScript* code can be "ported" to new blog engines as long as appropriate drivers are provided. The only change is that now the *MOFScript* code is executed with a different value for the "driver" configuration parameter.

On the downside, introducing abstract platforms on top of existing platforms can add some cost in terms of memory and execution speed at the time the generated program is executed. This is true in a general setting but does not apply here. The overhead caused by mapping "the abstract API" to, e.g. Blojsom's APIs, only occurs at the time the blog is created, not during blog operation. Therefore, the indirection penalty has no impact on the everyday use of the so-generated blog.

The bottom line is that platform independence is not an all-or-nothing option. When pursuing platform-independence, one could strive for PIMs that are neutral with respect to all different classes of technical platforms. This frequently forces *MOFScript* mappings to bridge a wide gap between the PIM and the PSM, leading to clutter code. Rather, this chapter advocates for abstract platforms as an hybrid way by moving platform specificities from the mapping code to the drivers.

4.9 Discussion

MDE 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 break-even point between manual coding vs. automation (i.e., code generation).

4.9.1 On the way to abstraction

Broadly, model-driven application development follows a top-down approach, i.e., starting from abstract PIMs to then gradually introduce more platform specifics, till, finally, the code is generated. However, from the experience of this chapter, the creation of the MDE infrastructure (i.e., metamodels and transformations) basically proceeds the other way around. This is more so if the notion of "abstract platform" is introduced as another artifact of the MDE infrastructure. Companies have already a set of blog engines they work with. This collection of blog engines sets the initial PSM space from which the CB metamodels are abstracted. This explains why these metamodels are not general-purpose models for Web applications, but models that abstract from *existing* blog engines. The options are then limited to those available to the initial set of 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. Our experience then confirms the insights of Markus Volter: *"...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"* [Völ08].

On the way to abstraction, models and abstract platforms as introduced in this chapter, provide two complementary mechanisms. The difference stems from both the time and means to concretise the abstraction into a target PSM. Models resort to transformations as "the concretisation mechanism" which is used at generation time. By contrast, abstract platforms are executable environments where software patterns can achieve the variability needed to accommodate distinct target platforms (e.g. the *Factory* pattern which is used in the *Blog Abstract Platform*). Unlike models, abstract platforms imply indirection, and hence incur in an additional cost at execution time. This cost should be balanced against the benefits reported by more maintainable, reusable and portable *MOFScript* code. Aspects to be considered include the heterogeneity of target PSMs, the likelihood of evolution for target PSMs, or the probability of enlarging the number of target PSMs. Additionally, the scarcity of *MOFScript* programmers can also support the option of moving the burden of handling PSM peculiarities from model mapping (using *MOFScript*) to an abstract platform (e.g. implemented in Java).

4.9.2 On model transformations as a reuse mechanism

MDE achieves reuse through abstractions (models) and model transformations. Models capture the specificities of the application at hand whereas transformations account for reuse. Hence, code programming is substituted by modeling and transforming. This subsection attempts to measure the reuse gains in terms of productivity by comparing manual coding vs. code generation using MDE techniques.

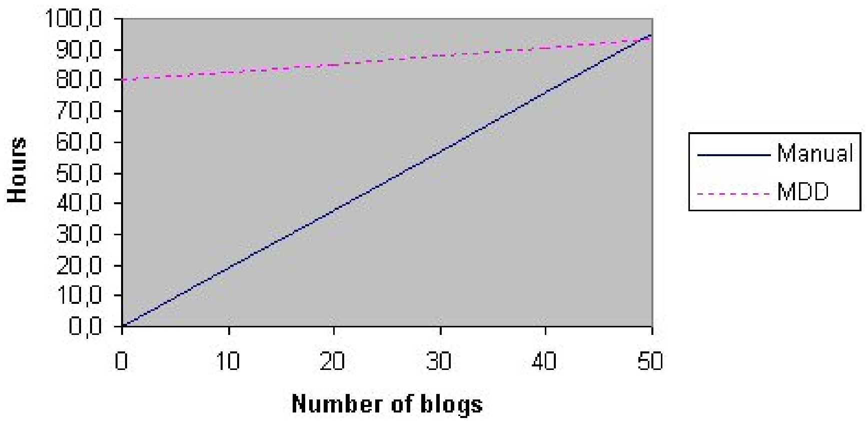


Figure 4.13: Breakeven in terms of 50-product blogs: 48,91 *catablog* projects are needed.

An empirical study was conducted comparing labour hours involved in obtaining a standalone file-based blog through either manual coding or code generation. Figure 4.13 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* is not included since this is an asset already available. 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 from granted. Therefore, the labour cost is restricted to adding the virtual community properties through the *Style model*. Total costs: 0,25 hours.

- Upfront investment. It includes both the cost of building the MDE infrastructure (i.e., meta-models and transformations), and the learning of the MDE tooling (in our case *ATL*, *AMW* and *MOFScript*). For an experienced and motivated developer, this accounts for 80 hours. Companies incur in this base cost no matter the number of *catablogs* being generated.

The gains 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 almost no impact on the MDE alternative where data is directly obtained from the OCF file. As an example, consider a sample catalogue including 25 categories and 50

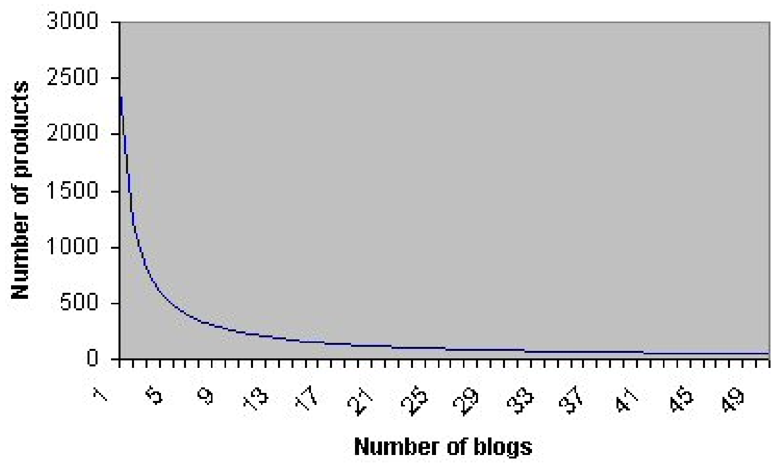


Figure 4.14: Break-even as a function of two parameters: number of blogs & number of products.

products with 7.2 properties per product on average. It took 1,9 hours to obtain the blog through manual coding. By contrast, MDE generates the very same blog but involving only 0,25 hours of labour work (mainly, the elaboration of the *Style model* and the *Annotation model*).

Figure 4.13 extrapolates these results by keeping constant the catalog size, and obtaining the breakeven as the number of *catablog* projects needed to payoff the upfront investment. Notice that the slope of the dotted line (i.e., the MDE approach) is less sharp than the manual-approach line. This stems from model annotation being easier than coding directly into *Blojsom* all the products' content. The figure highlights the role of MDE as a reuse technique where benefits are gradually obtained along distinct projects.

Figure 4.14 extrapolates those results, and assesses the breakeven as a function of two parameters, namely, the number of blogs and the number of products. For instance, scenarios where the breakeven is met include *(2 blog projects of 1500 products)* or *(25 blog projects of less than 100 products).*

4.10 Related work

This work contributes to two main areas: e-catalogue support and model-driven development. Next paragraphs provide related work in each of these areas.

Catalogue support. To the best of our knowledge, we are not aware of any project that addresses automatic generation of blogs. Catalogue browsing through small screens is addressed in [GT07]. This could be contemplated as a different technological platform from blog engines that, quite likely, would also require extension at the PIM level. Future work could contemplate transforming *catablog* to platforms other than blogs (e.g. PDA). This would permit to capitalise on existing PIMs, and focus on the transformation to PDAs, and, in so doing, proving the advantages that the MDE architecture brings to cope with new PSM.

In the area of library catalogues, [VR06] focuses on the extraction and description of library cards using XML technologies. Although the main challenge rests on harmonising the different types of catalogue cards, they also present a way to render such library cards in HTML using XSLT technology. A step forward would be to use blogs rather than static HTML pages to display library cards. Library members could then leave *comments* and engage in discussions that support reader communities. This could be easily achieved by just specifying library cards as OCF descriptions, and then, generate "library blogs" using the approach described in this chapter.

Model-Driven Engineering. The Web Engineering community is actively supporting MDE practices. There exists a series of workshops on this topic (MDWE[7]), and different experiences have been reported (refer to [RPSO07] for an overview). Most of the approaches focus on data-intensive Web applications (see [EK04] for a review on Web design methods). These approaches are design-intensive, and involve important upfront investment. By contrast, our approach has cost-effectiveness as a main requirement. From this perspective, blogs offer a good balance between price and functionality where companies can checkout the benefit of *catablogs*, and if successful, move to more sophisticated and costly solutions.

On a different front, few studies report on the experiences in adopting MDE. The adoption of this approach in a large IT consultancy organisation is described in [KR06]. Specially interesting is the quote *"small-to-medium sized projects found the model-driven approach too heavyweight and restrictive. Shorter project durations didn't allow for investments in learning/training. Need to deliver quickly made the teams intolerant to tooling irritants. The perceived loss of control of the development process and artifacts was the principal reason for these projects not taking to model-driven approach."* [KR06]. Our project can be considered as medium sized, and we indeed experienced some of the mentioned drawbacks. However, these limitations were overcome by supportive programmers that saw the project as an opportunity to delve into Software Engineering advance practices. In-

[7] http://mdwe2009.pst.ifi.lmu.de/

deed, we regards *catablog* as a pilot project from which to obtain vivid experiences on how MDE automates recurrent tasks, allowing the developer to focus on higher value labors on the hope that this experience will ultimately ease the transition to MDE in companies.

A set of variables to weight the MDE effort is proposed in [MBVM06]. These variables include tools maturity (measured by the time that the user was idle due to bugs in the tool), learning curve (measured by a survey of attendees of each course), resistance to change how willing people are to start using MDE (measured by a survey of attitudes), and, finally, perceived value of using MDE (also measured by a survey of attitudes). The paper does not compare manual vs. generate approaches. Neither do we address the impact that, let's say, tools maturity has on the final break-even point. We estimate this break-even point for a set scenario, and measure the labour cost in this scenario. Hence, we do not claim that our results can be directly extrapolated to other companies since the maturity of MDE tooling (improving as time goes by) or the resistance to change to MDE (decreasing as MDE becomes mainstream) are moving targets. However, the results show out that even with the current state of affairs at the time of this writing, MDE is moving from promises to practice.

4.11 Conclusions

This chapter addresses how blogs can be generated from product catalogues in a cost-effective way. The main challenges come from the heterogeneity and instability of blog engines. The chapter presents an MDE architecture where metamodels and transformations are introduced that gradually detach the models from their final realisation through blog engines. The use of an "abstract platform" also helps to make the final solution more robust. Some figures shown the cost-effectiveness of the MDE solution in comparison with manual coding.

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. We hope to report on these issues as further insights are gained by developing more *catablog* projects.

Chapter 5

Blog as Peers

"When I began blogging, I imagined that someday there might be hundreds of Weblogs, with tens of thousands of readers. Instead of dozens of Weblogs with a million readers, there are now well over four million Weblogs worldwide - most with only a few dozen readers."

– Rebecca Blood.

5.1 Introduction

The content of blog *posts* can range from personal experiences to professional-oriented content. Along with this broader spectrum comes the need to move from isolated, personal blogs to **blog rings** where blogs conform chains of "like blogs". The difference between an isolated blog and a blog pertaining to a ring rests on offering basically three additional links: *"Join"* | *"Next"* | *"Prev"* that permit participants to join the ring, go to the next blog in the ring, or go to the blog that is sequenced previous to the one you are on (for a list of blog rings refer to [alt10]).

So, basically, a blog ring is a way to connect blogs. This connection is restricted to visit related blogs sequentially. However, some scenarios can benefit from a tighter integration. So far, the blog subject is the main criterion to aggregate blogs into a ring (e.g. *"Canals and Waterway"* to mention a true ring [Rin08]). However, other criteria are possible. For instance, rings can also be defined on a personal basis: blogs of actors participating in the same movie, blogs of colleagues collaborating on the same project, blogs of lecturers teaching the same subject, etc. As an example, consider the latter case. Blogs are proposed as a conduit for lecturers to communicate with students. Blog *posts* can stand for course's themes or assignments. Then, students can pose their questions as blog *comments*, visible to other students. A blog ring can be defined among blogs which sustain the very same course, although taught by distinct lecturers. In this way, students can peer at different blogs of the same subject.

Blog integration through rings is commonly limited to reading. Students can peer to blogs other than the one of their own lecturer. However, writing *posts* and *comments* all along the ring is normally not supported. The community created by the ring is limited to consumers (reading *posts*) rather than providers (writers of *comments*). *Post* writing requires appropriate permissions. Even from reading, the blog ring does not account for a seamless integration: consumers are aware of skipping from one blog to the following one (e.g. changes on blog *themes*). As reported in [OTH$^+$04], blogs fall short as communication platforms. But, after all, blogs were not thought for peer collaboration but asymmetric relationships between the blog owner and the blog participants. If symmetric collaboration is the target, other platforms such as *Facebook* [Mar10] can be the answer. In *Facebook*, users collaboration starts through a username-based request. Once the request is accepted, *posts* and *comments* of each other are shown in the user wall (i.e., a page that aggregates friends' *posts* of a user). *Posts* can be private, white/black-listed, only visible for friends, or friends of friends. However, *Facebook* lacks the autonomy that blogs enjoy. Bloggers can choose not only the type of platform (e.g. where to keep their *posts*: online vs. locally) but also they retain the control over what is the subject to talk about (i.e., what are the blog *posts*). Compared with the symmetry of *Facebook* (i.e., no difference exists between lecturers and students), the asymmetry of blogs better mirrors some real scenarios (e.g. unlike students, lecturers can post *entries* in the blog). The question is how to find a *federated approach* where blog owners can freely join collaboration spaces but retain the control over their own blogs.

This chapter describes ***BlogUnion***, i.e., a federated approach to blog integration. *BlogUnion* provides a contract-based distributed and heterogeneous cross-blogging framework. Using a contract, two blogs agree on how information flows from one blog to another. For instance, *lecturerA's blog* and *lecturerB's blog* set a contract to indicate how *posts* on a certain subject or *comments* on a certain *posts* can flow between their blogs. In this way, students do not have to skip from one blog to the next, but *posts/comments/trackbacks* can seamlessly flow between blogs according to the contract specification.

Previous scenario is specified through a **contract model** which is enacted by a **contract engine**. The operational semantics of contracts is described in terms of event-condition-action rules. Rules regulate how information flows between blogs. The approach is evolutive in the sense that existing mechanisms in blog engines are sufficient to support the contract engine: contract negotiation is inspired in *trackbacks* while contract representation follows the content syndication format. The so-developed contract engine, named ***BlogUnion***, is delivered as an extension for *Blojsom* [Dav]. From the point of view of the participants (e.g. the students) there are no difference with traditional blogs.

The rest of the chapter is organised as follows. Section 5.2 introduces current approaches to integrate blogs through links and *linkbacks*. Our approach aims at providing tighter forms of integration through contracts. This is motivated through two running scenarios in Section 5.3. The design and implementation of *BlogUnion* are addressed in Sections 5.4 and 5.5, respectively. Finally, Section 5.6 discusses the approach, and conclusions (Section 5.7) end the chapter.

5.2 On blog integration through linking

Both *posts* and *comments* are URL-addressable. Hyperlinks (or links) are then a common mechanism to refer to blog content. From this perspective, *"... a 'community' is a set of blogs linking back and forth to one another's postings while discussing common topics... Members of such an informal community might list one another's blogs in a 'blogroll' (...) and might read, link to, and respond to content in other community member's blog"* [KNRT04]. Next sections delve into how hyperlinks are used to create blog communities.

5.2.1 Linking at the blog level

Blogrolls and *webrings* are two mechanisms for blog linking. Grouped blogs tend to have something in common. This commonality can rest on the blog subject (e.g. blogs about cooking) or some external matter (e.g. blogs from player pertaining to the same team) [BHH02]. Differences stem from (1) how membership is established, and (2) how related blogs are visited.

A **blogroll** is a group of blogs compiled by the blog owner, and exposed to the blog participants as a blog sidebar index [DDJ$^+$02][Wor10a]. Once on a blog, you can move to any other blog in the roll.

A **webring** is a list of blogs compiled by distinct blog owners, and exposed as a navigation ring (i.e., arrows forward and backward are available to move along the ring) [BHH02]. Once on a blog, you can only visit either the previous or the following blog in the ring. By permitting blog owners to add their blog to the ring, this structure is more dynamic and participative than its *blogroll* counterpart. Any blogger can create a *webring*, or ask for his blog to be joined to an existing *webring*. This implies filling out a form on a site, or contacting the *webring* creator directly. The *webring* creator decides which petitions to include. After approval, the requester blogger needs to add an HTML or JavaScript snippet to link the previous and next blogs of the *webring*. This permits blogs of the *webring* to receive traffic from related blogs. The linking between blogs is provided by a central site to prevent the *webring* from breaking when a blog goes offline. Examples of *webring* providers include [alt10], [Wor10d] and [Web10].

5.2.2 Linking at the post level

The term *linkback*[1] refers to a method for Web authors to obtain notifications when other authors link to one of their documents. This enables authors to keep track of who is linking/referring to their articles. To this end, three approaches exist: *Trackback*, *Pingback* and *Refback*. Next paragraphs delve into the differences.

A **trackback** enables websites to communicate via "pings", where each ping informs the blog that the sending site has made a reference to a *post* on this blog [Six04]. A *trackback ping* is sent to a *trackback URL* (i.e., a unique address associated with a *post*) through an HTTP POST request. This request needs four parameters: *title*, *excerpt*, *url*, and *blog_name*. Only the *url* parameter is required. The rest of the parameters contextualise the *trackback*.

As an example, consider two bloggers, *A* and *B*, who own *blogA* and *blogB*, respectively. *BloggerA* writes a *postA* in the *blogA*, and decides to inform the *blogB* that the *postA* references a *postB* of the *blogB*. To this end, a *trackback ping* is sent to the *trackback URL* of the *postB*. The following parameters are sent in the HTTP request: **title** of the *postA*; **excerpt** of the *postA*; **url** for the *postA* (a.k.a. its *permalink*); **blog_name** of the *blogA*.

The *trackback* is moderated by *bloggerB*. If accepted, a link from *postB* to *postA* is published on *postB*. Thus, the participants of both blogs can be involved in a conversation. Since *trackbacks* only send an excerpt of *postA*, *blogB's* partici-

[1] *http://en.wikipedia.org/wiki/Linkback*

pants are encouraged to read and comment the rest of *postA's* content [Wor10a].

Trackbacks can be seen as notifications of related or interesting content (e.g. *postA* is related to or interesting for *postB*), or as an external *comment* (e.g. a *comment* from *postA* to *postB*). Similar to *comments*, a list of *trackbacks* is attached to each *post*.

A **pingback** is a method to automatically notify when a source *post* links[2] to a target *post* [Stu02]. When a *post* is published, its content is scrutinised for HTML links. For each detected link, the source blog retrieves the target *post*, through an HTTP request, to autodiscover the URL of the *pingback server* on the target blog. This URL can be included in the "X-Pingback" HTTP header of the HTTP response, or as part of the resultant HTML page. The latter implies parsing the HTML page to search the following tag:

<link rel="pingback" href="pingback server URL" />

Next, a *pingback* is sent to this URL through an XML-RPC request[3]. This request needs two parameters: the source *post's* URL and the target *post's* URL. To prevent spam, the target blog verifies that the content of the source *post* contains a link to the target *post*. In addition, it retrieves some information from the source *post* (e.g. the page title, an extract of the source content surrounding the target link, etc). The target blog links the source *post* to the information so obtained.

Finally, a **refback** is a method to automatically discover source *post* URL through the HTTP header [Uch08]. When the participants click on a link to access a *post*, the URL of the previous *post* is sent in the "referrer" HTTP header. Then, the linked blog can access this URL, and retrieve some information, such as *pingback* does. Finally, the target blog links the source *post* with this information. Unlike *trackbacks* and *pingbacks*, *refback* only requires the linked blog to be *refback* enabled.

5.3 Requirements

Both blog and *post* linkings rest on "similarity", i.e., the condition of having something in common. This similarity can be established at the blog level: if two blogs address similar matters then, they are candidates to pertain to the same *blogroll*

[2]This differs from *trackbacks* where an HTML link is not necessary. However, *trackbacks* could be implemented to check the existence of this link as *pingback* does.

[3]This is similar to a *trackback* ping.

or *webring*. Alternatively, if two *posts* address a similar issue then, one *post* can *trackback* upon the other. Regardless of whether similarity is established at either the blog level or the *post* level, users are in charge of supporting what this similarity implies (e.g. adding a new blog to the ring, a *trackback* to a *post*, etc.). ***BlogUnion*** extension aims at allowing blog owners to set this similarity and state the consequences in a declarative way.

As a running example, consider two lecturers teaching the same course (e.g. *Databases*). Both lecturers use blogs for course support (e.g. *blogA* and *blogB*). ***Blogs are autonomous***: (1) each lecturer classifies content along his own *categories* (e.g. meetings, subjects, etc.), (2) course subjects can be categorised differently, and (3) each blog accounts for a different student community. This autonomy also implies that all, i.e., *posts*, *comments* and *trackbacks*, are within the blog silo. However, "opening the silos" (under regulated conditions) can account for cross-fertilisation among both communities. Two possible scenarios follow:

1. if *postA* is similar to *postB* then, commenting on *postB* can be propagated as a *comment* on *postA*. Both lectures need to agree on *post* similarity: one of them proposes this similarity while the other accepts or rejects it,

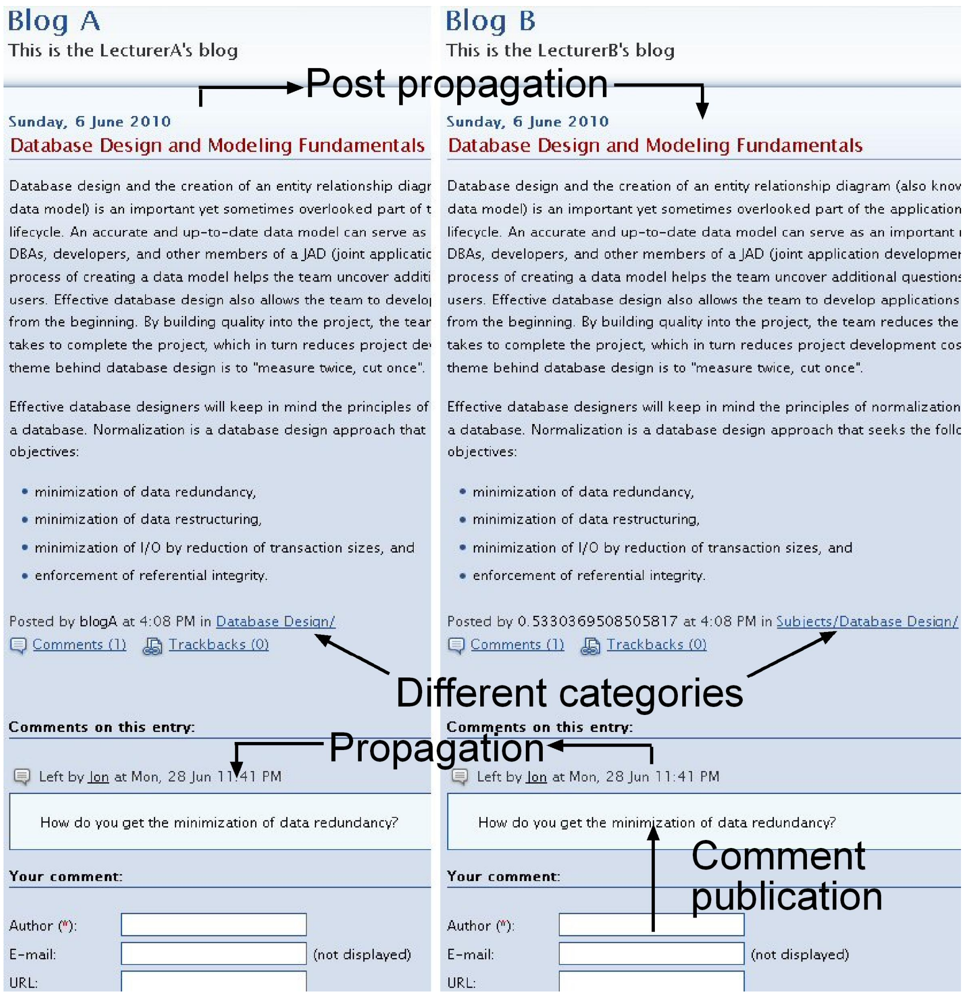
2. if *categoryA* (e.g. *Database Design)* is set as similar to *categoryB* (e.g. *Normalisation* or under a different supercategory *e.g.* $Subject \rightarrow DatabaseDesign$) then, *posts* on *categoryA* can be forwarded to *categoryB*. This can be extended to *comments* so that students of no matter the community are aware of all *comments* no matter the blog. Figure 5.1 depicts this situation.

These basic scenarios permit to identify main requirements for *BlogUnion*:

- **functionality**. Blog owners should be able to (1) agree on *post* pursuing similar aims, or addressing similar topics, and (2) indicate the consequences of this similarity as operations on blog resources,

- **usability**. *BlogUnion* is targeted to end users. It should be designed to be extremely lightweight to use, and intuitive to grasp. Blog owners should be able to define contracts at any time,

- **interoperability**. *BlogUnion* is all about opening blog silos and allowing blog data to flow between blogs. Implementation wise, *BlogUnion* should cause a minimum footprint on existing blog engines.



Blog A
This is the LecturerA's blog

Sunday, 6 June 2010
Database Design and Modeling Fundamentals

Database design and the creation of an entity relationship diagr
data model) is an important yet sometimes overlooked part of t
lifecycle. An accurate and up-to-date data model can serve as
DBAs, developers, and other members of a JAD (joint applicatic
process of creating a data model helps the team uncover additi
users. Effective database design also allows the team to develo
from the beginning. By building quality into the project, the tear
takes to complete the project, which in turn reduces project de
theme behind database design is to "measure twice, cut once".

Effective database designers will keep in mind the principles of
a database. Normalization is a database design approach that
objectives:

- minimization of data redundancy,
- minimization of data restructuring,
- minimization of I/O by reduction of transaction sizes, and
- enforcement of referential integrity.

Posted by blogA at 4:08 PM in Database Design/
Comments (1) Trackbacks (0)

Comments on this entry:

Left by Jon at Mon, 28 Jun 11:41 PM

How do you get the minimization of data redundancy?

Your comment:

Author (*):
E-mail: (not displayed)
URL:

Blog B
This is the LecturerB's blog

Sunday, 6 June 2010
Database Design and Modeling Fundamentals

Database design and the creation of an entity relationship diagram (also know
data model) is an important yet sometimes overlooked part of the application
lifecycle. An accurate and up-to-date data model can serve as an important
DBAs, developers, and other members of a JAD (joint application developmer
process of creating a data model helps the team uncover additional question
users. Effective database design also allows the team to develop applications
from the beginning. By building quality into the project, the team reduces the
takes to complete the project, which in turn reduces project development cos
theme behind database design is to "measure twice, cut once".

Effective database designers will keep in mind the principles of normalization
a database. Normalization is a database design approach that seeks the follc
objectives:

- minimization of data redundancy,
- minimization of data restructuring,
- minimization of I/O by reduction of transaction sizes, and
- enforcement of referential integrity.

Posted by 0.5330369508505817 at 4:08 PM in Subjects/Database Design/
Comments (1) Trackbacks (0)

Comments on this entry:

Left by Jon at Mon, 28 Jun 11:41 PM

How do you get the minimization of data redundancy?

Your comment:

Author (*):
E-mail: (not displayed)
URL:

Figure 5.1: *BlogA* (left handside) and *BlogB* (right handside) become a *BlogUnion*.

5.4 BlogUnion design

BlogUnion advocates a federated approach to blog integration. By federated is meant that blogs keep their autonomy, and freely decide to exchange blog data after some deliberations. Decisions are specified in terms of contracts while deliberations take the form of contract negotiations. This section introduces a model for both contract specification and contract negotiation.

5.4.1 Contract specification

A contract comprises a set of terms or clauses [MJSSW03][GHM00]. These clauses stipulate how the signing parties are expected to behave, i.e., they list the rights and

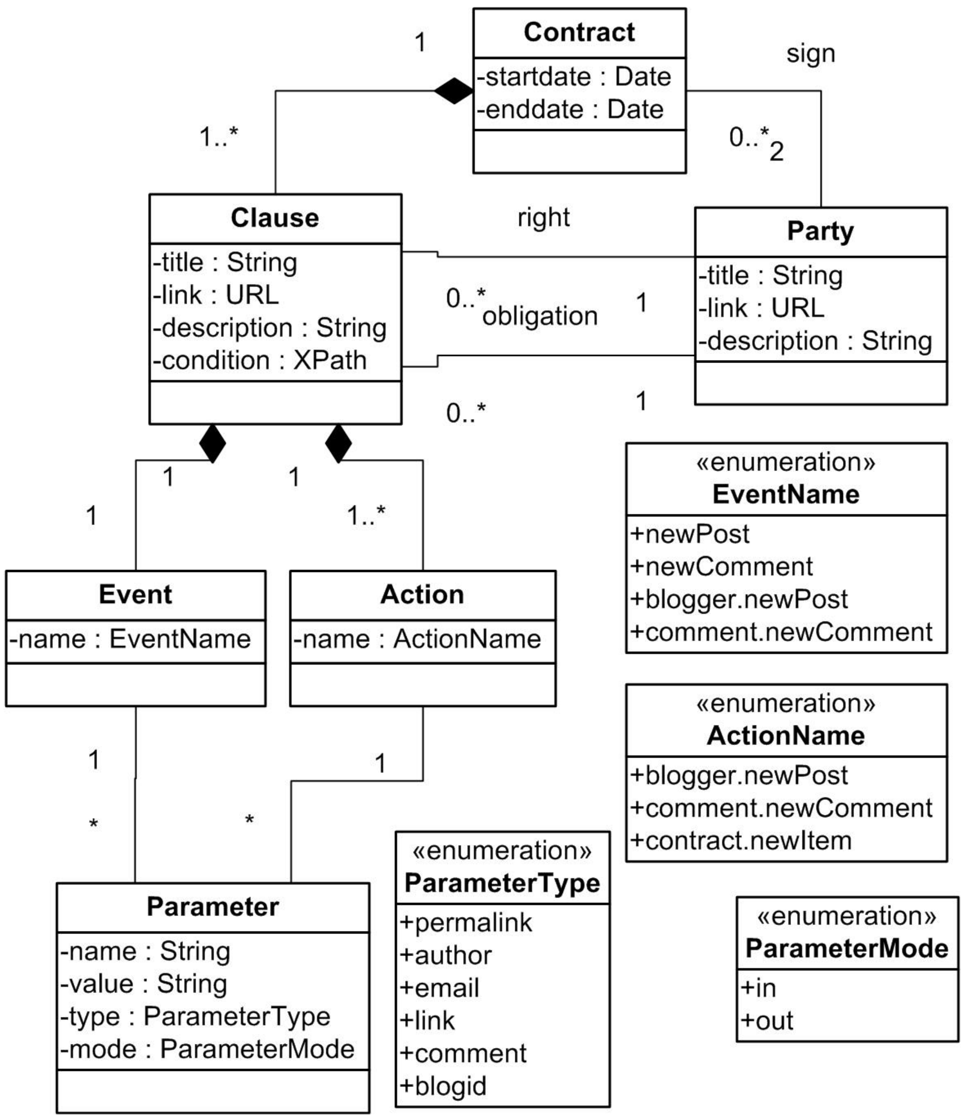


Figure 5.2: The *BlogUnion* contract model.

obligations of each signing party [MJSSW03]. [GHM00] defines what elements are needed to create a contract and how these elements appear in the clauses as *description of parties involved*, *definition and interpretation of terms*, *jurisdiction*[4], *duration and territory*, *nature of consideration* (e.g. fees, services rendered, goods exchanged, rights granted, etc.) and *obligations*.

These terms need to be re-written for the blog domain. Figure 5.2 depicts the

[4]Because of the distributed nature of the Web, neither *jurisdiction* nor *territory* is considered in our development.

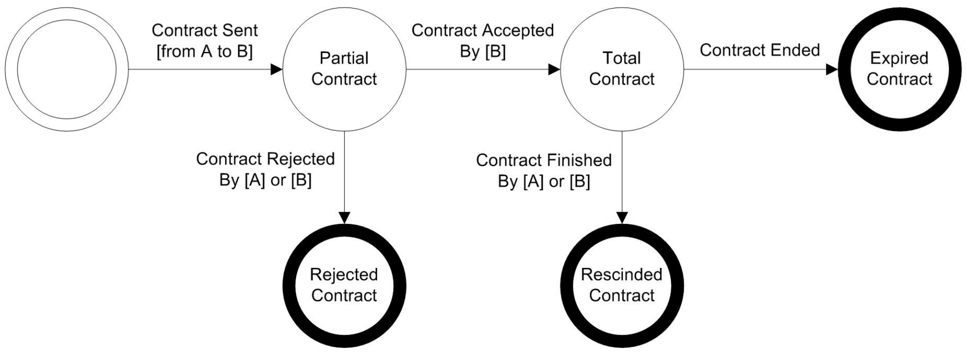


Figure 5.3: *Contract negotiation* life cycle.

model for the *BlogUnion* contract. A **Contract** is an aggregate of **Clauses**, signed between two **Parties**. The contract validity is defined by *startdate* and *enddate*. Parties can be involved in different clauses and contracts. A party is defined by a title, a link and a description. A clause grants a right to one party (e.g. to create a *post*, a *comment*, etc.) and sets an obligation for the other party. Such rights and obligations are described in two ways. The human-readable version of the rule is given by title, link, and description properties. The prescriptive version is specified in terms of event-condition-action rules (ECA rules) to be consumed by the *BlogUnion* engine.

5.4.2 Contract negotiation

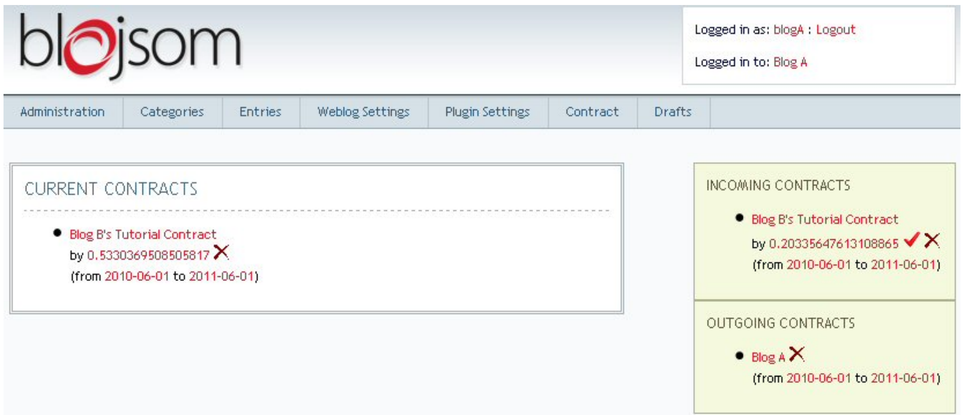
A contract negotiation is established between two parties: the *proponent* and the *receptor*. The *proponent* starts the negotiation by proposing a contract. The *receptor* either accepts or rejects it. If accepted, the negotiation goes along the following stages: *contract establishment*, *contract enactment*, and *contract ending*.

Contract establishment. This phase mimics the *trackback* communication method [Six04][PP04], so it can be easily enacted by blog owners. Figure 5.3 depicts the process (the description blends both *what* and *how* each step is supported):

1. The *proponent* lecturer searches for a *receptor* blog, and gets the *receptor's blog URL*[5] from the browser.

2. A *partial contract* is created by the *proponent*. So far, the contract only has information about him and his intentions.

[5]The blog's main page URL.



Logged in as: blogA : Logout

Logged in to: Blog A

CURRENT CONTRACTS

- Blog B's Tutorial Contract
  by 0.5330369508505817 ✗
  (from 2010-06-01 to 2011-06-01)

INCOMING CONTRACTS

- Blog B's Tutorial Contract
  by 0.20335647613108865 ✓ ✗
  (from 2010-06-01 to 2011-06-01)

OUTGOING CONTRACTS

- Blog A ✗
  (from 2010-06-01 to 2011-06-01)

Figure 5.4: Contract management GUI.

3. The *partial contract* is sent to the *blog URL*, through an *ad hoc* XML-RPC programmatic interface. This interface is anonymous to allow any blog to send a contract to another blog. The use of anonymous interfaces is broadly accepted in blog systems to create *comments* [The03] and *trackbacks* [Six04]. However, they require of moderation systems to avoid undesired information. In our case, the blog administration interface has been extended to moderate contracts as depicted in Figure 5.4. The *outgoing contract box* permits *proponent* to manage pending contracts.

4. The *receptor* is notified about arrival contracts through the *incoming contract box* (see Figure 5.4). The *receptor* could approve or reject the contract proposal[6]. If rejected, the negotiation ends.

5. If the *receptor* approves the contract then, the contract is completed with the receptor information, and becomes a *total contract* (hereafter referred to as *contract*). Both, the *proponent* and the *receptor* are notified through the *current contract box* (see Figure 5.4). Previous notifications of this contract at the *incoming/outgoing boxes* disappear. At this point, *contract enactment* is established, and each blog maintains a local copy of the contract.

[6]In the graphical-user interface, the approval or the rejection of a contract is done by clicking on either the "V" or "X" icon, respectively.

ELEMENT	RSS	CONTRACT
contract	-	duration
channel	1 *channel*, blog description	2 *channels*, party description
item	*post* content	rights and obligations

Table 5.1: RSS vs. Contract.

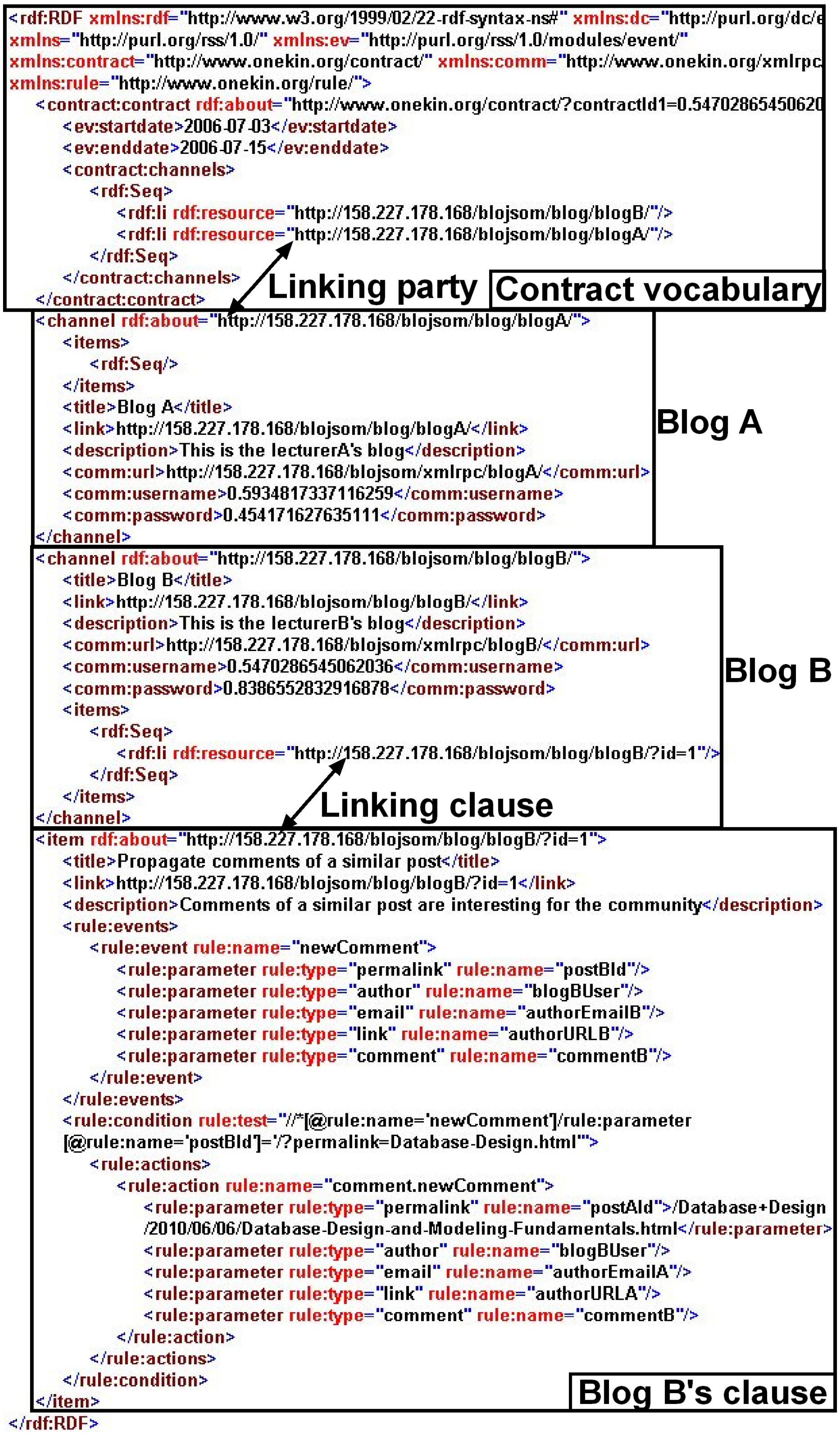
Contract enactment. A contract comprises a set of clauses. Expressiveness wise, a clause takes the form of an ECA rule. Hence, clause enactment requires first to monitor blog changes as event realisations. Next, the condition is checked, and if it is met, an action on the blog's receptor is enacted. Some of these operations require authentication (e.g. *post* creation, modification or deletion).

Contract ending. Finally, the contract could expire or be rescinded. The contract expiration is obtained automatically when an ending condition is satisfied (e.g. the last day of the contract validity passes). By contrast, the contract could be rescinded manually at any time from the *current contracts box* (see Figure 5.4) by either the *proponent* or the *receptor*. Manual cancellation is useful when one of the parties detects a breach of the contract.

5.5 BlogUnion implementation

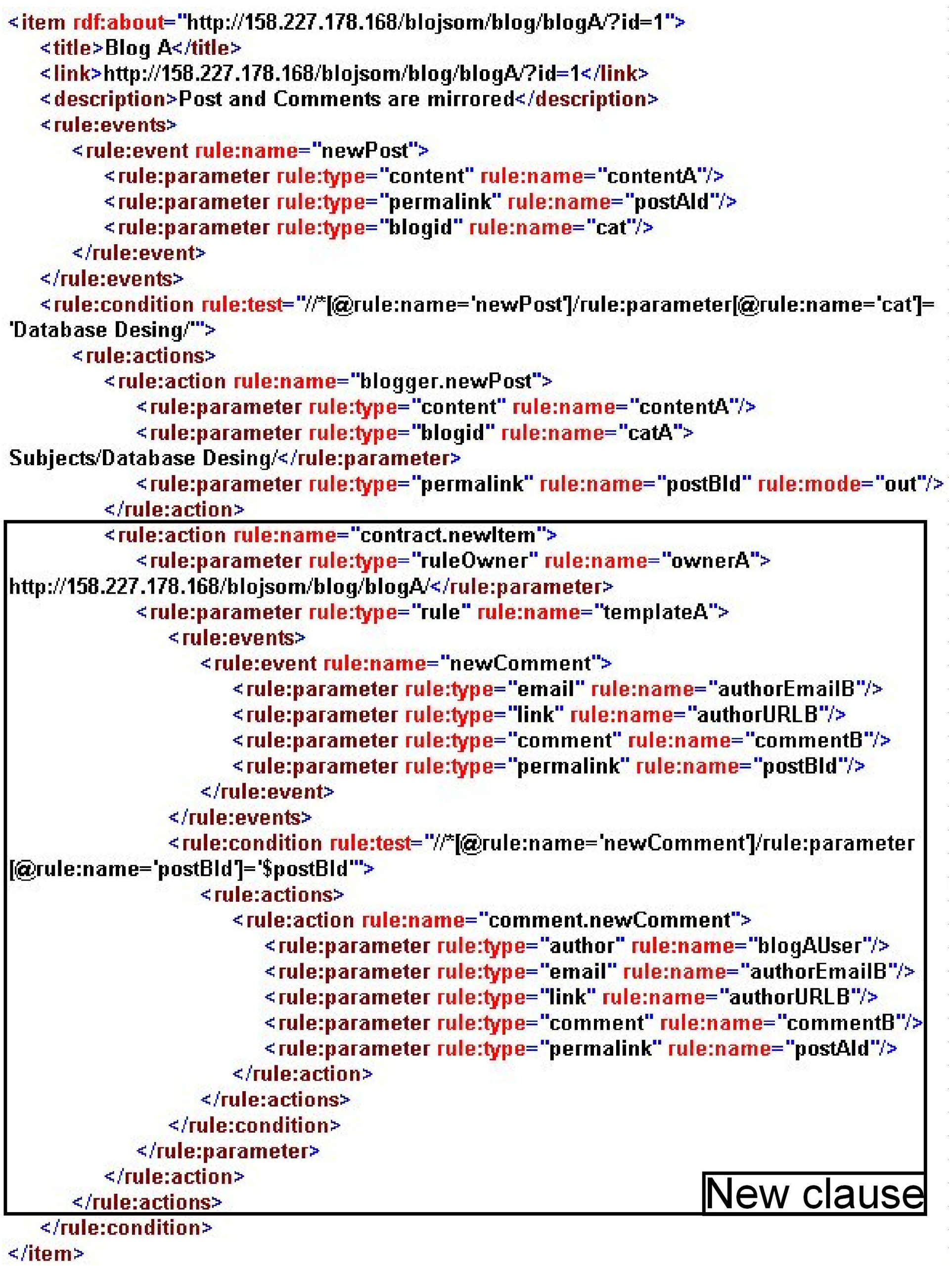
Interoperability imposes to use mechanisms that are widely available no matter the blog engine. Specifically, the main implementation decisions are: (1) contract representation follows the content syndication format, and (2), contract negotiation is inspired in *trackbacks*.

A contract is realised as a syndication between parties. Content syndication is a well-known technology used by blog developers. They develop RSS templates to be consumed by RSS aggregators. Therefore, RSS 1.0 [Web02] (i.e., an RDF implementation of RSS format) is idoneous as a base format over which contract implementation could be developed. Although a contract is a syndication between parties, RSS, just as it is, does not model a contract. The main elements of RSS 1.0 are **one** *channel* (i.e., the blog description), and a set of *items* (i.e., a list of *posts*). In our case, the contract has **two** *channels* (i.e., *description of parties*), and a set of *items* (i.e., *definition and interpretation of terms*, and *nature of consideration*). Each item is linked with one party, defining its *rights* and *obligations*. Furthermore, a new element (i.e., <*contract:contract*> tag) has been created to locate general information (e.g. *duration*). Table 5.1 compares both formats.



```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dc="http://purl.org/dc/e
xmlns="http://purl.org/rss/1.0/" xmlns:ev="http://purl.org/rss/1.0/modules/event/"
xmlns:contract="http://www.onekin.org/contract/" xmlns:comm="http://www.onekin.org/xmlrpc
xmlns:rule="http://www.onekin.org/rule/">
  <contract:contract rdf:about="http://www.onekin.org/contract/?contractId1=0.54702865450620
    <ev:startdate>2006-07-03</ev:startdate>
    <ev:enddate>2006-07-15</ev:enddate>
    <contract:channels>
      <rdf:Seq>
        <rdf:li rdf:resource="http://158.227.178.168/blojsom/blog/blogB/"/>
        <rdf:li rdf:resource="http://158.227.178.168/blojsom/blog/blogA/"/>
      </rdf:Seq>
    </contract:channels>
  </contract:contract>
  <channel rdf:about="http://158.227.178.168/blojsom/blog/blogA/">
    <items>
      <rdf:Seq/>
    </items>
    <title>Blog A</title>
    <link>http://158.227.178.168/blojsom/blog/blogA/</link>
    <description>This is the lecturerA's blog</description>
    <comm:url>http://158.227.178.168/blojsom/xmlrpc/blogA/</comm:url>
    <comm:username>0.5934817337116259</comm:username>
    <comm:password>0.454171627635111</comm:password>
  </channel>
  <channel rdf:about="http://158.227.178.168/blojsom/blog/blogB/">
    <title>Blog B</title>
    <link>http://158.227.178.168/blojsom/blog/blogB/</link>
    <description>This is the lecturerB's blog</description>
    <comm:url>http://158.227.178.168/blojsom/xmlrpc/blogB/</comm:url>
    <comm:username>0.5470286545062036</comm:username>
    <comm:password>0.8386552832916878</comm:password>
    <items>
      <rdf:Seq>
        <rdf:li rdf:resource="http://158.227.178.168/blojsom/blog/blogB/?id=1"/>
      </rdf:Seq>
    </items>
  </channel>
  <item rdf:about="http://158.227.178.168/blojsom/blog/blogB/?id=1">
    <title>Propagate comments of a similar post</title>
    <link>http://158.227.178.168/blojsom/blog/blogB/?id=1</link>
    <description>Comments of a similar post are interesting for the community</description>
    <rule:events>
      <rule:event rule:name="newComment">
        <rule:parameter rule:type="permalink" rule:name="postBId"/>
        <rule:parameter rule:type="author" rule:name="blogBUser"/>
        <rule:parameter rule:type="email" rule:name="authorEmailB"/>
        <rule:parameter rule:type="link" rule:name="authorURLB"/>
        <rule:parameter rule:type="comment" rule:name="commentB"/>
      </rule:event>
    </rule:events>
    <rule:condition rule:test="//*[@rule:name='newComment']/rule:parameter
    [@rule:name='postBId']='/?permalink=Database-Design.html'">
      <rule:actions>
        <rule:action rule:name="comment.newComment">
          <rule:parameter rule:type="permalink" rule:name="postAId">/Database+Design
          /2010/06/06/Database-Design-and-Modeling-Fundamentals.html</rule:parameter>
          <rule:parameter rule:type="author" rule:name="blogBUser"/>
          <rule:parameter rule:type="email" rule:name="authorEmailA"/>
          <rule:parameter rule:type="link" rule:name="authorURLA"/>
          <rule:parameter rule:type="comment" rule:name="commentB"/>
        </rule:action>
      </rule:actions>
    </rule:condition>
  </item>
</rdf:RDF>
```

Figure 5.5: Scenario 1: propagating *comments* of a similar *post* from *blogB* to *blogA*.



```
<item rdf:about="http://158.227.178.168/blojsom/blog/blogA/?id=1">
    <title>Blog A</title>
    <link>http://158.227.178.168/blojsom/blog/blogA/?id=1</link>
    <description>Post and Comments are mirrored</description>
    <rule:events>
        <rule:event rule:name="newPost">
            <rule:parameter rule:type="content" rule:name="contentA"/>
            <rule:parameter rule:type="permalink" rule:name="postAId"/>
            <rule:parameter rule:type="blogid" rule:name="cat"/>
        </rule:event>
    </rule:events>
    <rule:condition rule:test="//*[@rule:name='newPost']/rule:parameter[@rule:name='cat']=
'Database Desing/'">
        <rule:actions>
            <rule:action rule:name="blogger.newPost">
                <rule:parameter rule:type="content" rule:name="contentA"/>
                <rule:parameter rule:type="blogid" rule:name="catA">
Subjects/Database Desing/</rule:parameter>
                <rule:parameter rule:type="permalink" rule:name="postBId" rule:mode="out"/>
            </rule:action>
            <rule:action rule:name="contract.newItem">
                <rule:parameter rule:type="ruleOwner" rule:name="ownerA">
http://158.227.178.168/blojsom/blog/blogA/</rule:parameter>
                <rule:parameter rule:type="rule" rule:name="templateA">
                    <rule:events>
                        <rule:event rule:name="newComment">
                            <rule:parameter rule:type="email" rule:name="authorEmailB"/>
                            <rule:parameter rule:type="link" rule:name="authorURLB"/>
                            <rule:parameter rule:type="comment" rule:name="commentB"/>
                            <rule:parameter rule:type="permalink" rule:name="postBId"/>
                        </rule:event>
                    </rule:events>
                    <rule:condition rule:test="//*[@rule:name='newComment']/rule:parameter
[@rule:name='postBId']='$postBId'">
                        <rule:actions>
                            <rule:action rule:name="comment.newComment">
                                <rule:parameter rule:type="author" rule:name="blogAUser"/>
                                <rule:parameter rule:type="email" rule:name="authorEmailB"/>
                                <rule:parameter rule:type="link" rule:name="authorURLB"/>
                                <rule:parameter rule:type="comment" rule:name="commentB"/>
                                <rule:parameter rule:type="permalink" rule:name="postAId"/>
                            </rule:action>
                        </rule:actions>
                    </rule:condition>
                </rule:parameter>
            </rule:action>
        </rule:actions>
    </rule:condition>
</item>
```

Figure 5.6: Scenario 2 (partial view): propagating *posts* published in a *category* from *blogA* to *blogB*, and sending back the *comments* from *blogB* to *blogA*.

5.5.1 The Contract Vocabulary

Contracts are defined using RSS 1.0. Figure 5.5 and Figure 5.6 describe the contract for implementing the scenario 1 and 2, respectively. The contract starts defining the XML Namespaces [W3C09] to accommodate additional tags (i.e., *modules*). Next, <*contact:contract*>[7] tag defines information with a global effect in the contract: *duration* and *parties* involved. The contract *duration* is described by the module *mod_event*[8] [Sor01], which uses <*ev:startdate*> and <*ev:enddate*> tags for this purpose. The <*contract:channels*> tag lists the *parties* involved through links to each *party* definition.

5.5.2 Party definition

Each *party* has an unique identifier (i.e., the *blog URL*), which is linked from the <*contract:contract*> tag, and is described through the RSS <*channel*> tag. There, information about <*title*>, <*link*>, and <*description*> tags is provided by the blog configuration properties. This gives human-readable information for describing a *party* (compare the header of the left handside blog in Figure 5.1 with the *Blog A*'s <*channel*> tag of Figure 5.5). RSS <*items*> tag lists the *obligations*[9] applied to this *party* through links to each *clause* definition. This differs from a pure RSS format where <*items*> refers to news articles.

The <*channel*> tag also contains non-RSS tags, such as <*comm:url*>, <*comm: username*>, and <*comm:password*>. These tags extend <*channel*> to provide information about the communication process (see Section 5.4.2). While the contract is *partial*, the *proponent* fulfil the communication information with the URL to receive feedback about contract negotiation. Although the *username* and the *password* are given, they are an autogenerated proposal only valid while the contract is in force. At this point, the *proponent* only knows the *receptor's blog URL*. Additional information about the receptor will only be available once the receptor accepts the contract.

5.5.3 Clauses as ECA rules

A *clause* defines an *obligation* of a *party*, making reference to the *nature of consideration* (e.g. goods exchanged, rights granted, etc.) of the other party's objects. The *nature of consideration* is based on the *definition and interpretation of terms*. These *terms* are described by the remote programmatic interfaces' methods involved in the *contract enactment* phase (e.g. create, modify, delete, etc.), and acts

[7]Each contract has an identifier, provided by the *proponent* blog, which permits identify univocally a contract in the *contract-realm*.

[8]An RSS 1.0 module proposed for being standard.

[9]Notice that, in Figure 5.5, *LecturerA* has no obligations while *LecturerB* has only one.

on blog objects (e.g. *posts*, *comments*, *trackbacks*, etc.). ECA rules define how to interpret these terms.

The proposal of ECA rules for contract definition is not new. In [KK08], it is stated that *"With Web services support, organizations can also send primitive events (when required) among business partners at different sites to generate the requisite composite events. Such events can then cause an e-contract system to evaluate an ECA rule's condition for fulfilling the contract activities. If the condition is evaluated to be true, the system can trigger the action for execution through a target Web service. A unified Web services infrastructure thus lets us streamline distributed events and ECA rule processing"*. We follow this approach adapted to blog-way of working.

Each *clause* has a *blog URL*-based unique identifier, which is linked from the *party*'s <*items*> tag, and is described through the RSS <*item*> tag. There, the *proponent* provides human-readable information for describing a *clause (*i.e., the <*title*>, <*link*>, and <*description*> tags). Additionally, we need a machine-oriented counterpart that permits rule enactment by the *BlogUnion* engine. This is done by defining ECA rules at <*item*> level with non-RSS tags (e.g. <*rule:events*>).

Events are happenings of interest for the domain at hand. For blogs, events stand for changes in the blog data (i.e., *posts*, *comments*, *trackbacks*). Such changes can be performed graphically (i.e., the event arises as a result of interacting with the blog's GUI) or programmatically (i.e., the event accounts for sending a message along the blog's API). Notice that both can achieve the very same results (e.g. creating a *post*, introducing a *comment*). However, their origins are different: graphical events are sourced by end users while programmatic events are originated by programs. These different sources advice to name them differently. When the blog is manipulated through its GUI, graphical events are risen: *"newPost"*, *"newComment"*, etc. By contrast, when it is manipulated through remote programmatic interfaces, the event is qualified by the name of the API: *"blogger.newPost"*, *"comment.newComment"*, etc. These APIs are being subject to standardisation. Finally, no matter the origin, events have parameters (i.e., the event payload). Figure 5.5 shows the case for the event *"newComment"*. Event parameters (i.e., *permalink*, *author*, *email*, *link*, and *comment*) are kept as rule variables (e.g. *"postBId", "blogBUser", "authorEmailB", "authorURLB"* and *"commentB"*). These variables are now available for the rule's condition and actions.

Condition. The <*rule:condition*> tag holds a *"test"* property which keeps an XPath expression over the rule's variables [W3C99]. For instance, the following expression checks whether the current event stands for commenting on the *post* "*/?permalink=Database-Design.html*", i.e., the *post's permalink* (see Figure 5.5):

rule:test="//[@rule:name='newComment']/rule:parameter [@rule:name='postBId']='/?permalink=Database-Design.html"'*

Actions can create programmatic events to change a blog remotely. Just like events, actions have parameters with their respective types and names. However, action parameters can also have two modes: input[10] and output. Input parameters get their values from event parameters or from output parameters returned by previously executed actions. By contrast, an output parameter creates a new parameter, or modifies an existing one as a result of the action execution. Additionally, parameters can be assigned constants (e.g. *"/Database+Design/2010/06/06/Database-Design-and-Modeling-Fundamentals.html"* for the *"postAId"* parameter in Figure 5.5). If parameters are optional, the empty string is assigned as default value.

Rules can then cause the creation of blog data on the partner's blog. Additionally, it is sometimes required to create rules on the partners' blog. This is so when the contract requires the flow of data to go both ways. This is illustrated by the second scenario. This scenario permits *posts* from *blogA* to flow to *blogB*, but once a so-created *post* is available in *blogB*, *comments* on this *post* at *blogB* should go the other way around to *blogA* so that *comments* are synchronised no matter where they are written.

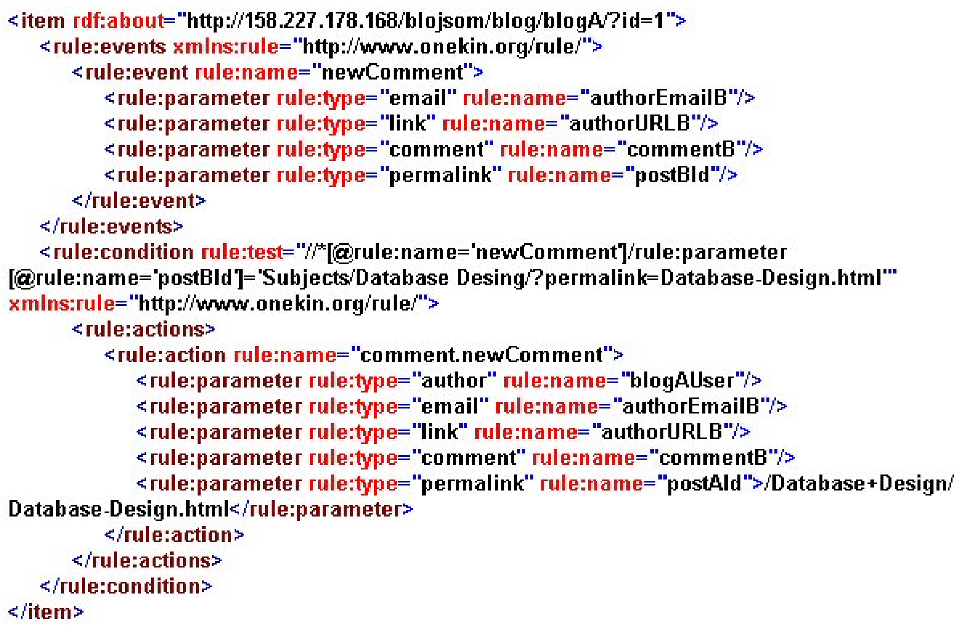
Creating a rule involves two parameters (see Figure 5.6):

- *ruleOwner* defines which party the rule will be applied to (see *rdf:about* attribute in the <*item*> element of Figure 5.7).

- *rule* parameter defines a rule *template*. When a new rule is created, the parameters of the *template* are filled with their values as a constant (see Figure 5.7). In this case, it can be necessary the use of *$variables*[11] to discriminate between value and constant value in the condition. For instance, in the condition of Figure 5.6, *"postBId"* is a constant value in *"@rule:name='postBId"'* while *"$postBId"* is the parameter value in *"='$postBId"'*. The condition of the new rule (see Figure 5.7) replaces *"$postBId"* with *"'Subjects/Database Design/?permalink=Database-Design-html"'*.

[10]This is the default value if the attribute mode does not appear.

[11]*$variableName* represents the value of the parameter called *variableName* taken from the rule context.



```
<item rdf:about="http://158.227.178.168/blojsom/blog/blogA/?id=1">
    <rule:events xmlns:rule="http://www.onekin.org/rule/">
        <rule:event rule:name="newComment">
            <rule:parameter rule:type="email" rule:name="authorEmailB"/>
            <rule:parameter rule:type="link" rule:name="authorURLB"/>
            <rule:parameter rule:type="comment" rule:name="commentB"/>
            <rule:parameter rule:type="permalink" rule:name="postBId"/>
        </rule:event>
    </rule:events>
    <rule:condition rule:test="//*[@rule:name='newComment']/rule:parameter
[@rule:name='postBId']='Subjects/Database Desing/?permalink=Database-Design.html'"
xmlns:rule="http://www.onekin.org/rule/">
        <rule:actions>
            <rule:action rule:name="comment.newComment">
                <rule:parameter rule:type="author" rule:name="blogAUser"/>
                <rule:parameter rule:type="email" rule:name="authorEmailB"/>
                <rule:parameter rule:type="link" rule:name="authorURLB"/>
                <rule:parameter rule:type="comment" rule:name="commentB"/>
                <rule:parameter rule:type="permalink" rule:name="postAId">/Database+Design/
Database-Design.html</rule:parameter>
            </rule:action>
        </rule:actions>
    </rule:condition>
</item>
```

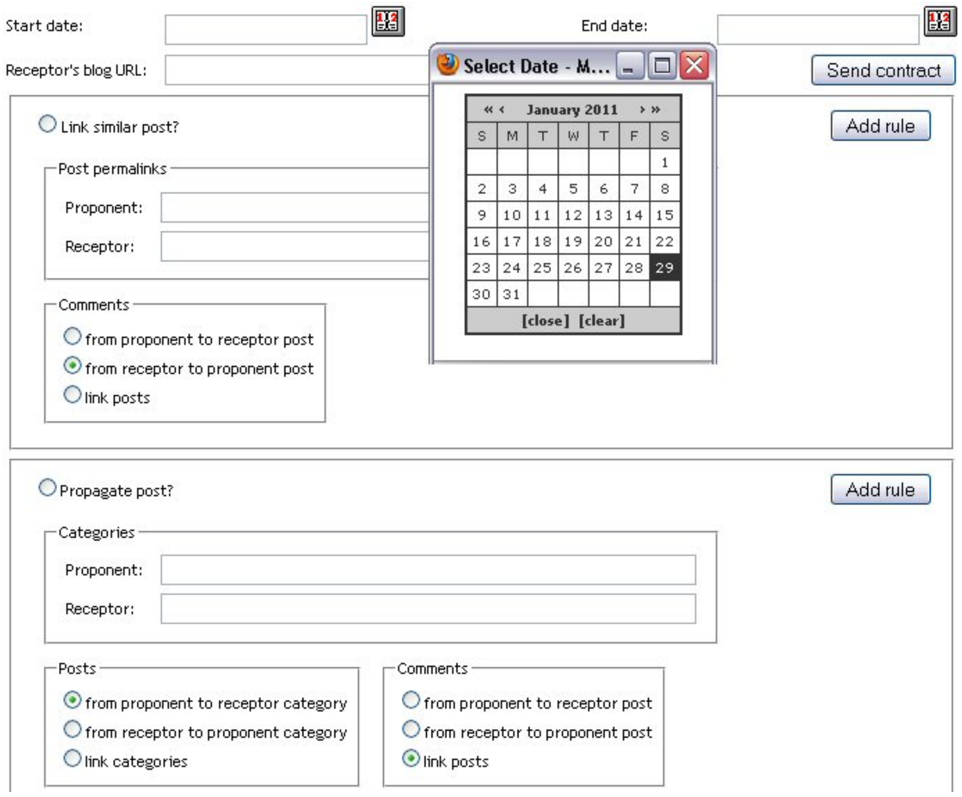
Figure 5.7: New rule for scenario 2.

5.5.4 An interface for contract definition

The use of XML for contract specification can certainly put users off, more to the point if blogs are targeted to non technicians. A GUI is required that permits the generation of XML contracts. This section presents a GUI provided as part of the *BlogUnion* extension (see Figure 5.8).

Firstly, the *blog owner* fills out the *receptor's blog URL* to send the contract, and defines its duration through the start and end date boxes. The duration fields are optional. The absence of the start date indicates the contract is in force on acceptance. In the case of the end date, the contract never ends so the contract can be cancelled only manually (see Figure 5.4). Next paragraphs describe the use of this interface to define the running scenarios.

Scenario 1. This scenario requires the *permalinks* of the *posts* being identified as similar, and the direction flow for *comment* propagation. The *permalinks* are obtained in a copy&paste manner (e.i., copying the *permalinks* from the blogs and, next, paste them into the interface). The *comment* propagation is configured by compulsorily selecting one of the following three options: (1) from the *receptor's post* to the *proponent's post*; (2) from the *proponent's post* to the *receptor's post*; (3) from one *post* to the other, and vice versa.



Start date:

End date:

Receptor's blog URL:

Send contract

Select Date - M...

« ‹			January 2011			› »
S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

[close] [clear]

Link similar post?

Add rule

Post permalinks

Proponent:

Receptor:

Comments

from proponent to receptor post [✓ from receptor to proponent post] link posts

Propagate post?

Add rule

Categories

Proponent:

Receptor:

Posts

[✓ from proponent to receptor category] from receptor to proponent category link categories

Comments

from proponent to receptor post from receptor to proponent post [✓ link posts]

Figure 5.8: Contract creation interface proposal.

Scenario 2. This scenario needs to set the blog *categories* being identified as similar, and the direction flow for *post* propagation[12]. *Category* values are also obtained in a copy&paste manner. The *post* propagation is configured by compulsorily selecting one of the following three options: (1) from the *receptor's category* to the *proponent's category*; (2) from the *proponent's category* to the *receptor's category*; (3) from one *category* to the other, and vice versa.

5.6 Discussion

Although the proposed approach suggests a rather flexible mechanism for sharing information between blogs, it only works on predefined blog engines. However, blog engines are in continuous evolution. We expect the proposed functionality will be useful enough to be adopted by blog developers. This was the case for the blog communication through *trackbacks*. Initially, blogs lacked of this framework

[12]Propagation could also be extended to *comments* if so desired.

when it was develop by Six Apart [Sixb]. From then on, it has been adopted by some blog engines, such as *b2evolution* [Fra], *Blojsom* [Dav], *PivotX* [Keva], to name a few. Others, such as *NucleusCMS* [Wou] and *Blosxom* [Kevb], provide *trackback* functionality as a plugin (i.e., not included in the blog engine core).

To ease this adoption, the new functionalities have been developed using technologies close to bloggers and blog developers. From the point of view of the bloggers, they have to learn to create and manage contracts. As shown in Section 5.5.4, the process to create a *BlogUnion* contract is quite simple. No technical requirements are needed. Also, sending a contract is similar to sending a *trackback*, a well-known technology. The contract management is quite similar to the *comment* and *trackback* moderation, where the blogger decides if a *comment* or *trackback* is finally published.

However, blog developers have more work to do. Although the technologies are close to them, they have to develop the contract model and the negotiation protocol with the corresponding interfaces. Apart from these new developments, they have to integrate the new interfaces with the old ones, and extend the interactions of the blog to connect them with the *contract engine*. For instance, when calling the *"newPost"* method of the Blogger API, a new *post* is created. Nonetheless, now, it is also needed to create a *"blogger.newPost"* external event, pass the parameters of the call to the *contract engine*, and process the contracts in force of the *current contract box* with these parameters (see Figure 5.4).

Despite the extension, the Blogger API and the publication delay (i.e., the response time between sending content to be published and the publication itself) are maintained. The interface does not change. Only it changes the implementation to add the new functionality to the existing one. Thus, the applications built around this API will call it, and work as before. Moreover, the new functionality does not affect the efficiency of the publication delay, because firstly the *post* is published, and, in this way, the delay is maintained. Finally, the contracts are processed in background. The latter has some cost in terms of memory and execution speed. [Hen09] explains this issue for wrapper development. In our case, it happens when adding an extra behaviour to process contracts. This cost depends on the number of rules to process, and actions to execute.

5.7 Conclusions

This chapter introduces a federated approach to blog integration. Blogs are kept autonomous, and eventually decide to engage in data flows between blog peers. Such data flows are defined through contracts and realised as event-condition-action rules. Data flow affects *posts*, *comments* and *trackbacks*. This vision is realised through *BlogUnion*, an extension for *Blojsom* that permits blogs to become parties in contracts, and, hence, participate in the exchange of blog data among their peers. The use of a GUI for contract definition permits end-users to easily define contracts. Next steps include a thorough evaluation of *BlogUnion* through real blogs. We plan to do this for educational blogs.

Chapter 6

Conclusions

"Each painting has its own way of evolving... When the painting is finished, the subject reaveals itself."

– *William Baziotes.*

6.1 Introduction

Throughout this dissertation, the use of blogs as diaries, as virtual community platforms, and as peers has been described. Because of posts do not always find their source in the personal experiences (i.e., contents from user's desktop documents, companies' catalogues, or another blogs are amenable to be blogged), each chapter exposed a very different problem focused on a specific data source. To solve these problems, three new publication techniques were used: annotation, model transformation, and crossblogging. These techniques gave rise to the implementation of *Blogouse*, *Catablog*, and *BlogUnion*, respectively. As a result, these tools strove to improve the publication process for the data sources.

The rest of this chapter summarises the main contributions of this dissertation. Next, the publications achieved along these years are listed. Finally, proposals for future research are described.

6.2 Main Contributions

Depending on the nature of the data source, it is up to the *bloggers* to decide what is the most convenient way of publishing in a blog. This dissertation presents some scenarios where new tools were developed to ease *bloggers* the publication. Next, the concrete contributions are detailed.

In chapter 3, *Blogouse* describes how semantic annotation can be successfully applied to improve blog publication client tools. These improvements basically consist on an extension of mouse devices functionality, which they make the mouse ontology-aware. As a result, a document with its annotations (metadata) can be discussed and shared easily with the blog *participants*. At this point, annotation and document are completely decoupled, and the blog is the link to give them a meaning.

This approach was presented in the 7th International Conference on Web Information Systems Engineering (WISE 2006): *Blogouse: Turning the Mouse into a Copy&Blog Device* [VAD06].

Chapter 4 addresses how blogs can be generated from product catalogues in a cost-effective way. The main challenges come from the heterogeneity and instability of blog engines. This chapter presents an MDE architecture where metamodels and transformations are introduced that gradually detach the models from their final realisation through blog engines. The use of an "abstract platform" also helps to make the final solution more robust. Some figures shown the cost-effectiveness of the solution in comparison with manual coding.

This approach was presented in 5th International Workshop on Model-Driven Web Engineering (MDWE 2009), co-located with the 9th International Conference of Web Engineering (ICWE 2009): *Generating blogs form Product Catalogues: A Model-Driven Approach* [DV09], and, also, in the Journal of Systems and Software (JSS 2010): *Generating blogs out of product catalogues: An MDE approach* [DV10].

Finally, chapter 5 introduces a federated approach to blog integration. Blogs are kept autonomous, and eventually decide to engage in data flows between blog peers. Such data flows are defined through contracts and realised as event-condition-action rules. Data flow affects *posts*, *comments* and *trackbacks*. This vision is realised through *BlogUnion*, an extension for *Blojsom* that permits blogs to become parties in contracts, and, hence, participate in the exchange of blog data among their peers. The use of a GUI for contract definition permits end-users to easily define contracts.

This approach was presented in the 11th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2011): *A federated approach to cross-blogging through contracts* [VD11].

6.3 Publications

Next, the publications in which the author was involved along the development of this dissertation are presented. They are grouped by type of publication, and organised chronologically inside the group.

6.3.1 Journals

- *Generating blogs out of product catalogues: An MDE approach.* Óscar Díaz and Felipe M. Villoria. Journal of Systems and Software (JSS 2010). October 2010 [DV10]. Impact Factor: 1.340[1].

6.3.2 International Conferences

- *Peer-Blog-Peer: The Swiss Knife.* Iker Azpeitia, Óscar Barrera, Felipe M. Villoria and Óscar Díaz. 3rd International Association for Development of the Information Society - Web Based Communities (IADIS 2006). San Sebastián, Spain. February 2006 [ABVD06]. Acceptance Rate: 54%.

- *Powering RSS Aggregators with Ontologies: a case for RSSOwl aggregator.* Felipe M. Villoria, Óscar Díaz and Sergio Fernández Anzuola. 8th International Conference on Enterprise Information Systems (ICEIS 2006). Paphos, Cyprus. May 2006 [VDA06]. Acceptance Rate: 59%.

- *Blogouse: Turning the Mouse into a Copy&Blog Device.* Felipe M. Villoria, Sergio Fernández Anzuola and Óscar Díaz. 7th International Conference on Web Information Systems Engineering (WISE 2006). Wuhan, China. October 2006 [VAD06]. Acceptance Rate: 29%.

- *Designing portlet aggregation through statecharts.* Óscar Díaz, Arantza Irastorza, Maider Azanza and Felipe M. Villoria. 7th International Conference on Web Information Systems Engineering (WISE 2006). Wuhan, China. October 2006 [DIAV06b]. Acceptance Rate: 20%.

[1]Journal Citations Reports © published by Thomson Reuters 2010

- *A federated approach to crossblogging through contracts.* Felipe M. Villoria and Óscar Díaz. 11th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2011). Munich, Germany. July 2011 [VD11]. Acceptance Rate: 32%.

6.3.3 International Workshops

- *Generating blogs form Product Catalogues: A Model-Driven Approach.* Óscar Díaz and Felipe M. Villoria. 5th International Workshop on Model-Driven Web Engineering (MDWE 2009), co-located with the 9th International Conference of Web Engineering (ICWE 2009). San Sebastián, Spain. June 2009 [DV09]. Acceptance Rate: 62%.

6.3.4 Spanish Conferences

- *Modelado de la agregación de portlets por medio de statecharts.* Óscar Díaz, Arantza Irastorza, Maider Azanza and Felipe M. Villoria. XI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2006). Sitges, Barcelona. October 2006 [DIAV06a]. Acceptance Rate: 33%.

6.4 Future Work

Blogouse, *Catablog*, and *BlogUnion* were implemented and proved as feasible using some sample cases. However, more research rests ahead.

Blogouse has been presented as a text based publication tool. However, documents also include figures, references (e.g. to footnotes, to figures, to tables, etc.), which should have to be considered as well. Considering the requirements of user-friendliness, editor-independence, and blog-independence, *Blogouse* should handle these elements and links, and provide the corresponding layout in the blog.

In the case of *Catablog*, future work includes enriching the *catablog* Model with additional aspects such as security or permission. Also, an option to explore is the definition of scopes (see *Style Model* in Section 4.6) based on queries over the catalog so that the style applies to those products meeting the query criteria. Another main issue is evolution. Catalogues evolve, and this evolution percolates their blog counterparts. Moreover, transforming *catablog* to platforms other than blogs (e.g. PDA) would permit to capitalise on existing PIMs, and focus on the transformation to PDAs, and, in so doing, providing the advantages that the MDE architecture brings to cope with new PSM.

The next steps for *BlogUnion* include a thorough evaluation through real blogs. We plan to do this for educational blogs. Also, exploring the use of *BlogUnion* in other domains will be beneficial for the development of the framework.

Finally, previous tools use the programmatic interfaces to communicate with blogs. However, these APIs are unstable and incomplete, therefore, API evolution is an open research issue.

Bibliography

[ABVD06] Iker Azpeitia, Oscar Barrera, Felipe M. Villoria, and Oscar Diaz. Peer-Blog-Peer: The Swiss Knife. *IADIS*, (February), 2006. 79

[ADvP04] J.P. Andrade Almeida, R.M. Dijkman, M.J. Sinderen van, and L. Ferreira Pires. Platform-independent modeling in MDA: supporting abstract platforms. In *Model Driven Architecture. European MDA Workshops, MDAFA 2004*, Sweden, June 2004. 46

[alt10] alt-webring. The alternative web ring system, 2010. http://www.alt-webring.com/. 57, 60

[Ama] amazon.com associates blog - The official blog of the Amazon Associates program. http://affiliate-blog.amazon.com/. 3

[APR$^+$06] Karl Aberer, Zhiyong Peng, Elke A. Rundensteiner, Yanchun Zhang, and Xuhui Li, editors. *Web Information Systems - WISE 2006, 7th International Conference on Web Information Systems Engineering, Wuhan, China, October 23-26, 2006, Proceedings*, volume 4255 of *Lecture Notes in Computer Science*. Springer, 2006. 85, 91

[Aud] hipcast: the audio & video podcasting service. http://www.audioblog.com/. 3

[Bar97] Jorn Barger. Robot Wisdom Weblog, 1997. http://www.robotwisdom.com/. 1

[BCFM07] M. Brambilla, S. Comai, P. Fraternali, and M. Matera. *Web Engineering: Modelling and Implementing Web Applications*, chapter Designing Web Applications with WebML and WebRatio, pages 221–260. Springer, 2007. 38

[BHH02] Paul Bausch, Matthew Haughey, and Meg Hourihan. *We Blog: Publishing Online with Blogs*. Wiley, 2002. 2, 8, 9, 59, 60

[Bloa] Blogger. http://www.blogger.com. 19, 31, 45

[Blob] BlogTalk. http://www.blogtalk.net/. 2

[Bloc] BloggerThemes.net. BloggerThemes: #1 WordPress Theme Generator. http://www.bloggerthemes.net/. 8

[Blod] BloggingThemes.com. Blogging Themes: Free Blog Templates. http://www.bloggingthemes.com/. 8

[Bloe] BlogThemes.com. BlogThemes: The Ultimate WordPress Theme. http://www.blogthemes.com/. 8

[Blof] BlogThemesPlus.com. BlogThemes Plus: ThemeForest. http://blogthemesplus.com/. 8

[Blo02] Rebecca Blood. *The Weblog Handbook: Practical Advice on Creating and Maintaining Your Blog*. Perseus Publishing, 2002. 2

[Blo05] Blogger. Blogger for Word, 2005. http://buzz.blogger.com/bloggerforword.html. 14, 18

[Blo06] BlogPulse, 2006. http://www.blogpulse.com/www2006-workshop/. 2

[Blo07] Blogged Out! List of Blog Software & Blog Generators , 2007. http://www.blogged-out.com/2007/05/26/list-of-blog-software-blog-generators/. 2

[BOB] The BOBs awards. http://www.thebobs.com/. 2

[BU] BPIR.com and Massey University. Customer market segmentation. http://www.bpir.com/ customer-market-segmentation-bpir.com/menu-id-72/expert-opinion.html. 41

[Dav] David Czarnecki. Blojsom. http://sourceforge.net/projects/blojsom/. 13, 15, 19, 31, 33, 42, 45, 59, 75

[Dav99] Dave Winer. XML-RPC Specification, 1999. http://www.xmlrpc.com/spec. 13, 19

[Dav01] Dave Winer. Weblogs.Com XML-RPC interface, 2001. http://www.xmlrpc.com/weblogsCom. 13

[Dav02] Dave Winer. RFC: Metaweblog API, 2002. http://www.xmlrpc.com/metaweblogAPI. 13, 45

[Dav03] Dave Winer. RSS 2.0 Specification, 2003. http://cyber.law.harvard.edu/rss/rss.html. 11

[DDJ+02] Cory Doctorow, Rael Dornfest, J. Scott Johnson, Shelley Powers, Benjamin Trott, and Mena G. Trott. *Essential Blogging*. O'Reilly, 2002. 13, 14, 15, 59

[dFBV06] 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. 36

[DIAV06a] Oscar Díaz, Arantza Irastorza, Maider Azanza, and Felipe M. Villoria. Modelado de la agregación de portlets por medio de statecharts. In José Cristóbal Riquelme Santos and Pere Botella, editors, *JISBD*, pages 453–462, 2006. 80

[DIAV06b] Oscar Díaz, Arantza Irastorza, Maider Azanza, and Felipe M. Villoria. Modeling portlet aggregation through statecharts. In Aberer et al. [APR+06], pages 265–276. 79

[DSMX02] Rajan Lukose Kiran Nagaraja Jim Pruyne Bruno Richard Sami Rollins Dejan S. Milojicic, Vana Kalogeraki and Zhichen Xu. Peer-to-peer computing. Technical report, HP Laboratories, 2002. 4

[DV09] Oscar Díaz and Felipe M. Villoria. Generating Blogs from Product Catalogues: A Model-Driven Approach. In *Proceedings of the 5th International Workshop on Model-Driven Web Engineering (MDWE)*, pages 61–75, June 2009. 78, 80

[DV10] Oscar Díaz and Felipe M. Villoria. Generating Blogs out of Product Catalogues: An MDE Approach. *Journal of Systems and Software*, 83(10):1970–1982, 2010. 78, 79

[Edu] edublogs: the world's most popular education blogging service. http://edublogs.org/. 3

[EK04] María José Escalona and Nora Koch. Requirements Engineering for Web Applications - A Comparative Study. *Journal of Web Engineering*, 2(3):193–212, 2004. 54

[Eva03] Evan Willians. Blogger API, 2003. http://www.blogger.com/developers/api/1_docs/. 13, 45

[Fra] François Planque. b2evolution. http://b2evolution.net/. 31, 75

[Gar05] Susannah Gardner. Time to check: Are you using the right blogging tool?, July 2005. http://www.ojr.org/ojr/stories/050714gardner/. 31

[GCP01] Jaime Gómez, Cristina Cachero, and Oscar Pastor. Conceptual modeling of device-independent Web applications. *IEEE MultiMedia*, 8(2):26–39, 2001. 38

[GHM00] Andrew Goodchild, Charles Herring, and Zoran Milosevic. Business contracts for b2b. In Heiko Ludwig, Yigal Hoffner, Christoph Bussler, and Martin Bichler, editors, *ISDO*, volume 30 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2000. 63, 64

[Goo10] Google. Google Toolbar, 2010. http://www.google.com/intl/en/toolbar/ff/index.html. 14

[Gra99] Brad L. Graham. The BradLands: Must See HTTP://, September 1999. http://www.bradlands.com/weblog/comments/september_10_1999/. 1

[GT07] Sheng Uei Guan and Yuan Sherng Tay. Interactive product catalogue with user preference tracking. *International Journal of Web and Grid Services*, 3(1):58–81, 2007. 54

[Ham03] Ben Hammersley. *Content Syndication with RSS*. O'Reilly, March 2003. ISBN: 0-596-00383-8. 12

[Hen01] James A. Hendler. Agents and the Semantic Web. *IEEE Intelligent Systems*, 16(2):30–37, March/April 2001. 27

[Hen09] Michi Henning. Api design matters. *Communications of the ACM*, 52(5):46–56, 2009. 75

[Hin07] Dion Hinchcliffe. A checkpoint on Web 2.0 in the enterprise, Part 2, August 2007. http://blogs.zdnet.com/Hinchcliffe/?p=135. 29

[Keva] Kevin Pascal and Bob den Otter. PivotX. http://pivotx.net/. 75

[Kevb] Kevin Scaldeferri and Axel Beckert. Blosxom. http://www.blosxom.com/. 75

[KK08] P. Radha Krishna and Kamalakar Karlapalem. Electronic contracts. *IEEE Internet Computing*, 10:60–68, 2008. 71

[KNRT04] Ravi Kumar, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. Structure and evolution of blogspace. *Communications of the ACM*, 47(12):35–39, 2004. 7, 59

[KQ04] David R. Karger and Dennis Quan. What Would It Mean to Blog on the Semantic Web? In *Proceedings of 3rd International Semantic Web Conference (ISWC2004)*, pages 214–228, November 2004. 20, 26

[KR06] Vinay Kulkarni and Sreedhar Reddy. Introducing MDA in a large IT consultancy organization. In *APSEC*, pages 419–426. IEEE Computer Society, 2006. 54

[LD08] Nicholas S. Lockwood and Alan R. Dennis. Exploring the corporate blogosphere: A taxonomy for research and practice. In *HICSS*, page 149. IEEE Computer Society, 2008. 29

[Let06] Letmeparty.com. Blog via SMS from your Mobile Phone!, 2006. http://www.letmeparty.com/. 14

[Li04] Charlene Li. Blogging: Bubble Or Big Deal?, November 2004. 17

[Lif] LifeType Open Source Blogging Platform. http://lifetype.net/. 31

[Lim10] Marcelo L. Limaverde. w.bloggar, 2010. http://www.wbloggar.com. 14

[Liv] LiveJournal. http://www.livejournal.com/. 31

[Mar] MartSoft, Inc. OCF - Open Catalog Format. http://www.martsoft.com/. 32

[Mar10] Mark Zuckerberg and Eduardo Saverin and Dustin Moskovitz and Chris Hughes. Facebook, 2010. http://www.facebook.com/. 58

[MBVM06] Jason Xabier Mansell, Aitor Bediaga, Régis Vogel, and Keith Mantell. A process framework for the successful adoption of model driven development. In Arend Rensink and Jos Warmer, editors, *ECMDA-FA*, volume 4066 of *Lecture Notes in Computer Science*, pages 90–100. Springer, 2006. 55

[MD05] Knud Möller and Stefan Decker. Harvesting Desktop Data for Semantic Blogging. *ISWC 2005 Workshop*, November 2005. 19, 27

[Me05] Kevin Marks and Tantek Çelik. Vote links, 2005. http://microformats.org/wiki/vote-links. 40, 41

[Meg02] Meg Hourihan. What We're Doing When We Blog, 2002. http://www.oreillynet.com/pub/a/javascript/2002/06/13/megnut.html. 11

[Mil] MILbloggin.com. http://milblogging.com/. 3

[MJSSW03] Carlos Molina-Jiménez, Santosh K. Shrivastava, Ellis Solaiman, and John P. Warne. Contract representation for run-time monitoring and enforcement. In *CEC*, pages 103–110. IEEE Computer Society, 2003. 63, 64

[Mov05] Movable Type. Movable Type API, 2005. http://www.sixapart.com/movabletype/docs/mtmanual_programmatic. 13, 45

[N06] Carr N. Lessons in corporate blogging. 2006. Business Week Online. 29

[Odd10] Sigfús R. Oddson. blogBuddy, 2010. http://blogbuddy.sourceforge.net/. 14

[OMG05] OMG. Software Process Engineering Metamodel Specification (SPEM). Adopted Specification, January 2005. 33

[OTH$^+$04] Ikki Ohmukai, Hideaki Takeda, Masahiro Hamasaki, Kosuke Numa, and Shin Adachi. Metadata-driven personal knowledge publishing. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 591–604. Springer, 2004. 58

[Pax03] Salam Pax. Salam Pax: The Clandestine Diary of an Ordinary Iraqi, 2003. 2

[Pho] Photoblog. http://www.photoblog.com/. 3

[PP04] Sébastien Paquet and Phillip Pearson. A topic sharing infrastructure for weblog networks. In *CNSR*, pages 301–304. IEEE Computer Society, 2004. 65

[PR85] Jon Postel and Joyce Reynolds. File Transfer Protocol (FTP), 1985. http://tools.ietf.org/html/rfc959. 2

[Rad] Radio UserLand. http://radio.userland.com/. 19, 45

[RHJ99] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification, 1999. http://www.w3.org/TR/html4/. 2

[Rin08] Ringsurf. World canals & inland waterways ring, 2008. http://webrings.bendall.de/waterways-ring.html. 57

[Röl03] Martin Röll. Business Weblogs - A pragmatic Approach to introducing Weblogs in medium and large Enterprises. *BlogTalk*, May 2003. 17, 19, 27

[Röl04] Martin Röll. Distributed KM - Improving Knowledge Workers' Productivity and Organisational Knowledge Sharing with Weblog-based Personal Publishing. *BlogTalk 2.0*, July 2004. 17

[Roc] RocketBoom: daily internet culture. http://www.rocketboom.com/. 3

[RPSO07] G. Rossi, O. Pastor, D. Schwabe, and L. Olsina, editors. *Web Engineering: Modelling and Implementing Web Applications*. Springer, 2007. 54

[RSD] Really Simple Discoverability 1.0. http://archipelago.phrasewise.com/rsd. 21, 22

[Sal] Salam Pax: The Baghdad Blogger. http://salampax.wordpress.com/. 2

[Sif06] David Sifry. State of the Blogosphere, 2006. http://www.sifry.com/alerts/archives/000436.html. 2

[Sif07] David Sifry. The State of the Live Web, 2007. http://www.sifry.com/alerts/archives/000493.html. 2

[Sixa] Six Apart. Movable Type. http://www.movabletype.com. 19, 31, 45

[Sixb] Six Apart. Six Apart News & Events. http://www.sixapart.com/. 75

[Six04] Six Apart. Trackback Technical Specification, 2004. http://www.sixapart.com/pronet/docs/trackback_spec. 13, 40, 60, 65, 66

[Sor01] Soren Roug and European Environment Agency. RDF Site Summary 1.0 Modules: Event, 2001. http://web.resource.org/rss/1.0/modules/event/. 70

[Stu02] Stuart Langridge and Ian Hickson. Pingback 1.0, 2002. http://www.hixie.ch/specs/pingback/pingback. 61

[SV06] Thomas Stahl and Markus Voelter. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, 1 edition, May 2006. 30, 32

[Tem] TemplatesBlogger.com. Templates Blogger: Free Blogger Templates for your Blog. http://www.templates-blogger.com/. 8

[The03] The Well Formed Web. The Comment API, 2003. http://wellformedweb.org/story/9. 12, 13, 66

[Tom03] Tom Coates. On permalinks and Paradigms..., 2003. http://www.plasticbag.org/archives/2003/06/on_permalinks_and_paradigms/. 11

[Uch08] Uche Ogbuji. Real Web 2.0: Battling Web spam, Part 2 - Use the power of community against spam, 2008. http://download.boulder.ibm.com/ibmdl/pub/software/dw/web/wa-realweb11/wa-realweb11-pdf.pdf. 61

[VAD06] Felipe M. Villoria, Sergio Fernández Anzuola, and Oscar Díaz. *logouse*: Turning the mouse into a copy&blog device. In Aberer et al. [APR$^+$06], pages 554–559. 78, 79

[VD11] Felipe M. Villoria and Oscar Díaz. A federated approach to cross-blogging through contracts. In *SAINT*, pages 91–99. IEEE Computer Society, 2011. 79, 80

[VDA06] Felipe M. Villoria, Oscar Díaz, and Sergio Fernández Anzuola. Powering rss aggregators with ontologies - a case for the rssowl aggregator. In Yannis Manolopoulos, Joaquim Filipe, Panos Constantopoulos, and José Cordeiro, editors, *ICEIS (4)*, pages 197–200, 2006. 79

[VdCdFM08] 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)*, 2008. 36

[Völ08] Markus Völter. MD* Best Practices, December 2008. http://www.voelter.de/data/articles/DSLBestPractices-Website.pdf. 51

[VR06] Jovana Vidakovic and Milos Rackovic. Generating content and display of library catalogue cards using xml technology. *Software: Practice and Experience*, 36(5):513–524, 2006. 54

[W3C] W3C. Cascading Style Sheets. http://www.w3.org/Style/CSS/. 8, 38

[W3C99] W3C. XML Path Language, 1999. http://www.w3.org/TR/xpath. 72

[W3C04] W3C. Resource Description Framework (RDF), February 2004. http://www.w3.org/RDF/. 5

[W3C09] W3C. Namespaces in XML 1.0 (Third Edition), December 2009. http://www.w3.org/TR/xml-names/. 12, 70

[Web02] Web Resource. RSS 1.0 Specification, 2002. http://web.resource.org/rss/1.0/. 12, 67

[Web10] WebRing. Creating Communities, Connecting People, 2010. http://dir.webring.com/rw. 60

[Win99] Dave Winer. EditThisPage.Com, 1999. http://www.scripting.com/davenet/1999/12/08/editthispagecom.html. 1

[Win01] Dave Winer. The history of weblogs, 2001. http://www.userland.com/theHistoryOfWeblogs. 1, 8

[Wor10a] WordPress.org. Introduction to Blogging, 2010. http://codex.wordpress.org/Introduction_to_Blogging. 8, 11, 13, 59, 61

[Wor10b] WordPress.org. Post to your blog using email, 2010. http://codex.wordpress.org/Post_to_your_blog_using_email. 14

[Wor10c] WordPress.org. Weblog Client, 2010. http://codex.wordpress.org/Weblog_Client. 2

[Wor10d] World of Webrings. World of Webrings, 2010. http://www.webringworld.org/. 60

[Wou] Wouter Demuynck and Rodrigo Moraes and Jeroen Budts. Nucleus CMS. http://nucleuscms.org/. 75

Epilogue

Making a dissertation is a travel full of uncertainties. The only certainty is the starting point, but not whether or when the travel is going to end. Behind they remain a lot of hours of dedication, which, probably, few people will appreciate. However, the end of this travel is the beginning of a new one, where the lessons learned should be applied. New uncertainties for a new amazing travel, where the only certainty is the evolution of the world. This evolution will provide us new technologies and ways of understanding the world. Let's ready for changes!

Acknowledgments

After two years working in various companies, I decided to get on a train with an uncertain destination. Firstly, I went to a well-known train station, the Faculty of Computer Engineering at the University of the Basque Country. There, I started looking for a train to achieve my goal, i.e., making a dissertation. Suddenly, two trains appeared in my way, which seemed to go towards this objective, although with different routes. I already knew some passengers in both trains, but I had to choose one of them. The train I caught was the Ekin research group, nowadays called Onekin. The beginnings were very hard, because I was alone in a carriage. However, this changed when the train arrived at the next station, the towers of Arbide, where all passengers were put together. After a three-year stay there, I went back to work for a company, from where I visited Arbide. Although, now, the train is again at the initial station (i.e., in the university), I see this travel from a new perspective. Along these years, I shared my life with people, or, better said, colleagues, whom I have too much to thank. I only hope I do not forget anyone.

First of all, I would like to express my gratitude to Óscar Díaz for giving me the chance to work with him in his research group. He taught me to open my mind, and encouraged me along this dissertation. Thanks for your dedication, and for showing me the right way. However, this would not have been possible without Iñaki Paz, who previously put me in touch with Óscar.

Along these years, I have had the chance to collaborate with Cristobal Arellano, Maider Azanza, Iker Azpeitia, Óscar Barrera, Óscar Díaz, Sergio Fernández, and Arantza Irastorza, from whom I learnt a lot. Thanks Arantza for sharing your dissertation experience, and reviewing this one, it has been very helpful.

Moreover, my gratitude to the remaining members of Onekin: Luis M. Alonso, Jokin García, Felipe Ibáñez, Jesús Ibáñez, Jon Iturrioz, Arturo Jaime, Jon Kortabitarte, Sandy Pérez, Gorka Puente, Juan J. Rodríguez, and Salvador Trujillo.

Outside of Onekin, Tomás A. Pérez advised me throughout my studies, although not always I have followed his advices. Thank you for your comments about this dissertation. Also, thanks to Javier Álvez, who was always there to give me a hand with LaTeX.

Despite the demand of this work, there was life out of the university. I would like to thank my family and friends for being always accessible when I needed them, even after long periods of time without having been in contact with some of them. They helped me to free my mind for some hours. Also, thanks to my parents, who have taught me the value of education from my childhood. They have always encouraged me to follow studying.

I specially thank Marisa for accompanying me along this hard travel. Along these years, we have moved house, got married, had and brought up a baby, and suffered this dissertation. However, what I most appreciate is your invaluable dedication, and your physical and mental endurance. Not only do you have supported me technically, but also at personal level. You have to be both father and mother of our daughter Patricia in my repeated absences. Sincerely, you have done a really fantastic job. I only hope I can compensate for your continuous efforts.

And what to say about you, my little Patricia. When we met, I was already involved in this dissertation, and you suffered some of my continuous absences. Thanks for making me get up the chair to play together, for being always so happy and giving away so much infectious smiles to me. We have a lot of time to recover.

Finally, I would like to express my gratitude to the unknown reviewers of the submitted articles for their insightful comments, and the reviewers and members of the dissertation committee for your invaluable work beforehand.

Thank God that's over!

Vita

Felipe Martín Villoria was born in San Sebastián, Spain on December 26th, 1977. He entered in the Faculty of Computer Engineering at the University of the Basque Country in 1995, where he received the degree of Engineer (Master of Science) in 2000. After two years working in various companies, he entered the Doctoral School at the University of the Basque Country in 2002, where he conducted his doctoral research until 2012. The last years of this dissertation were combined with his work as an IT consultant.