A Genereic DSL for Content Type Development; CT-DSL

Louis Fleron, Jonas Norlin, Gunn Weihe Reinert, Nataliya Vlizko, and Xia Liu

IT University of Copenhagen,
Rued Langgaards Vej 7, DK-2300 Copenhagen S
`{jnor,lfle,gwre,???,???}@itu.dk`
`http://www.itu.dk`

Abstract. Content Management Systems (CMS) control user-created content in relation to websites, the content is defined by content-types. CMSs come with predefined content-types, but system specifications may require for customized content-type to be developed. The development of customized content-types is a time-consuming process that involves several manual steps. In this work we analyze three open source CMSs and present a Model-Driven Development approach for automating the steps required in the development of these customized content-types. The approach focuses on a generic domain-specific language (DSL) called CT-DSL, that enables domain experts to describe content-types at a higher abstraction level.
We present two kinds of evaluations of CT-DSL, an informal software experiment and an online community questionnaire. We find that the CT-DSL has indeed reduced the time needed for development, but we also find from the questionnaire that the communities do not require the CT-DSL.

1 Introduction

Content Management Systems[20] provide the basic framework for modern web solutions and let users publish, edit and modify content from a central interface.

With the advent of the internet, the requirements of CMSs have been constantly increasing, resulting in very complex systems. The increase in complexity has caused domain experts, notably web designers, to spend an increased amount of time and resources on adapting code bases to meet customer demands e.g. the development of customized content-types. Content-types are the heart of a CMS as they define the properties of the content. For example, a picture must contain a title, description and an image file.

Many CMS's come with predefined content-types, but these are rarely enough to meet the demands of customers. To meet the demands, customized content-types must be developed and added to the CMS. The development of customized content-types is a time-consuming process which requires several manual steps. The steps typically involve writing a description, an editor, and a HTML representation of the content-type. Finally, the content-type must be registered in the

CMS. The following example from the CMS Jease illustrates the problem. The example shows one of the manual steps required for creating a simple Person content-type that contains first name, last name, birthday and an address. We will be using this example throughout the paper.

Person content-type

```
...
public class Person extends Content {
public String firstname;
public String lastname;
public Date birthday;
public String address;
public String getFirstname() {
return firstname; }
public void setFirstname(String firstname) {
this.firstname = firstname; }
public String getLastname() {
return lastname; }
public void setLastname(String lastname) {
this.lastname = lastname; }
public Date getBirthday() {
return birthday; }
public void setBirthday(Date birthday) {
this.birthday = birthday; }
public String getAddress() {
return address; }
public void setAddress(String address) {
this.address = address; }
...
```

(A code-snippet from the Java file required for a Person content-type.)

There are three more steps required to complete the Person content-type in Jease, two of the steps require implementation of Java classes that are of the same length and complexity as the example. The last step involves registering the Person content-type within Jease.

Writing the Java implementations leads to introduction of errors, the process of implementation and rectification of errors is a cumbersome process. In this paper we present a Model-Driven Development approach to the automation of the steps in the development of customized content-types. Our focus is on providing a generic DSL, thus enabling the domain experts to describe the content-type at a higher abstraction level. Our goal is to improve the productivity of the developers by automating the implementation process. The automation will reduce the time needed for development of customized content-types. It will also prevent manually introduced errors.

We evaluate our DSL by conducting two different experiments, an informal software experiment and an online user-group questionnaire. The software experiment indicates that CT-DSL improves productivity for the development of a simple content-type, while the results from the questionnaire indicate that the majority of the participants do not require a DSL. The remainder of the paper is structured in eight main sections. Section II introduces the background of CMS's and greater details of the chosen systems. Section III presents our DSL for automating the steps involved in creating customized content-types. In section IV we present two different evaluations of the DSL.

Section V introduces the threats to validity, which might undermine the results from the evaluations. Section VI introduces the related work that has been done in the area of interest. Section VII introduces a discussion of our findings. Finally section VIII presents the conclusions of our work.

2 Background

Content Management Systems help web developers accomplish mundane tasks easier by providing predefined functions and structures that can be extended and modified to meet the needs of a custom solution. A common denominator for CMS's is the concept of a "content type" which defines and holds information about the various types of content that resides in the system. Content types are created in many different ways depending on the chosen CMS; Some have graphical interfaces while others require code to be implemented manually.

W e decided on using Jease (`http://www.jease.org`) , N2 (`http://n2cms.com`) and Concrete5 (`http://www.concrete5.org`) in our work.

Jease is implemented in Java, the name Jease stands for Java with ease, and requires writing three Java classes and a XML file for a single content type. Jease uses inheritance to implement content-types, each content-type inherits from a content class.

N2 is implemented in .Net C# and uses inheritance and .Net attributes to define content types. N2 requires the creation of three classes for a single content type.

By using inheritance and attributes the development of new content-types can be done very quickly. N2 does not use XML files to install the content-type in the database. Each content-type inherits from a content-item class which is defined in the database.

Concrete5 is implemented in PHP. In Concrete5, a content type is called a page type which is basically a container for numerous attributes and fields. Blocks are pieces of content which can be placed on pages in defined regions. The page types are implemented with four PHP files that integrates the new functionality and a XML file that describe how tables should be defined in the database.

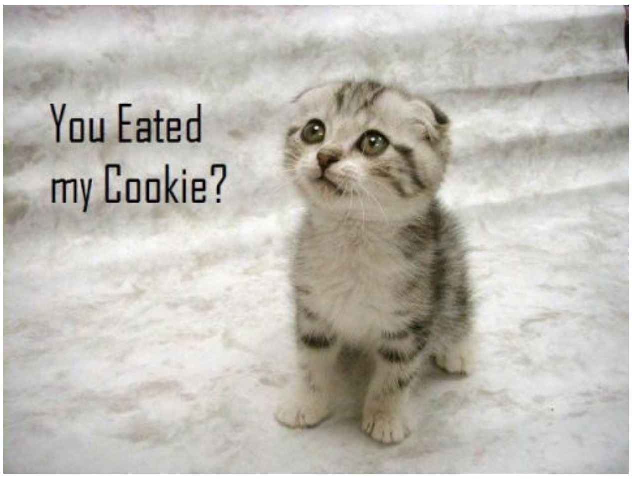

Fig. 1. The following figure shows how a content-type is usually developed in the mentioned CMSs. Each of the files must be manually written.

3 Content-Type Domain-Specific Language (CT-DSL)

We have addressed the problem concerning creation of customized content types by conducting an analysis of the three open source CMS's mentioned in the previous section.

These three CMSs use different programming languages and concepts for the development of customized content-types. Relying on heterogeneous CMSs when creating a DSL for implementation raises the likelihood that the resulting DSL is generic.

O ut of the architectures for customized content-type implementation we create a conceptual model.. The conceptual model is expanded iteratively analyzing the content-type component found in each of the selected CMS's. The primary class is a content type which contains many properties. Property is also a class . A user is also part of the model as the content-type has a creator. The user has a role within the CMS. Finally, the content-type can exist in many versions. Figure [X] illustrates the relationship between the mentioned classes.

The architecture analysis is used to create an abstract syntax of CT-DSL using the Eclipse Modelling Framework (EMF) [23].

The abstract syntax consists of three types of classes, architectural classes, helper classes and enumerators. The architectural classes come from the conceptual model. The helper classes are introduced to assist with forming the concrete syntax and to ensure genericity. The enumerators contain predefined values.

T he ContentType class represents all information about the content-type, information such as properties, user and versions. The Property class contains

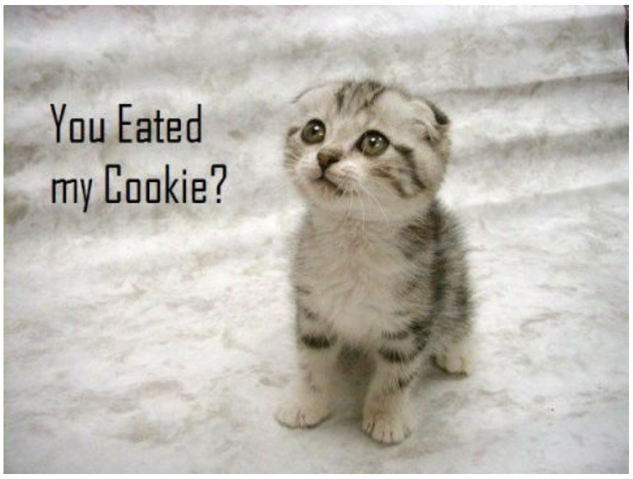

Fig. 2. The following figure shows how a content-type is usually developed in the mentioned CMSs. Each of the files must be manually written.

information about a single property of a content type. A property consists of an access modifier and data type.

The example shown in the Introduction section describes a ContentType class, Person, with four properties. The properties are first-name, last-name, birthday and address. The individual property consists of an access-modifier and a data type e.g. the address property consists of a public access-modifier and a string data type.

The User class contains all information about the user who modified the content type. Each user has a Role class that contains information about the specific role of the user. Version class defines the different versions of a content type.

ContentModel class is the root node of the abstract syntax. Each class inherits from the NamedElement class. This also allows for an easier handling of classes in the concrete syntax, as the ContentModel class only needs to contain NamedElements. The CMS class contains information about the target CMS, information such as the type of the CMS.

The CMSEnum enumerator contains the three possible CMSs from our analysis, Jease, N2 and Concrete5. The TypeEnum enumerator contains all possible data types in the DSL, example string and int. By predefining the data types, we ensure that the user selects a data type which is recognizable. AccessLevelEnum enumerator specifies if a property of the content type is public or private. By predefining these values, we ensure that the user selects a correct access modifier.

Use another font when referring to source-material, such as AccessLevelEnum. The concrete syntax is generated with xText[10]. When importing an EMF model into xText, a default concrete syntax is generated. Web developers primarily use two different notations, XML and general-purpose programming language. We have adapted the default concrete syntax by applying Guideline 14 [5] Adopt existing notations domain experts use.. This makes the syntax is closer to the existing notations used by the domain experts. This led to the creation of two different concrete syntaxes of CT-DSL, a syntax based on XML[21] and a syntax based on HUTN.[18,19].

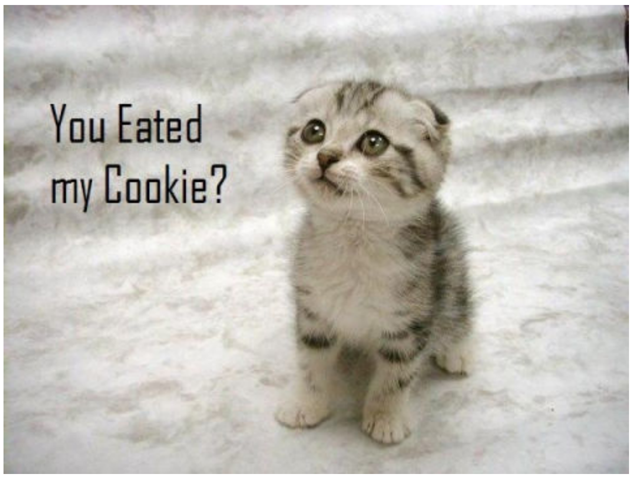

Fig. 3. The following figure shows how a content-type is usually developed in the mentioned CMSs. Each of the files must be manually written.

CT-DSL allows specifying multiple content types within the same source file, thus removing the need for multiple source files. As mentioned earlier, each CMS requires different files for implementing a content type. To obtain the required files for a content type, we apply Model-to-Text transformation in form of a code generator. The code generator is implemented with Xtend2.0.[9, 24]

CMSEnum enumerator is used in the code generator to decide the target language of the output. By specifying multiple CMSEnums in CT-DSL, it is possible for the code generator to generate content types for several CMS's.

4 Evaluation

Evaluation of CT-DSL is done in two different ways. An online questionnaire to the communities of the three CMS's mentioned in previous sections and an informal software experiment. If one of the evaluations fail, then the DSL itself must be changed or it is useless. We will only be able to claim a success if both evaluations are passed without any remarks.

Survey. As developers of CT-DSL, we should not decide how the syntax looks, It is the CMS developers who will be working with the language. We might find

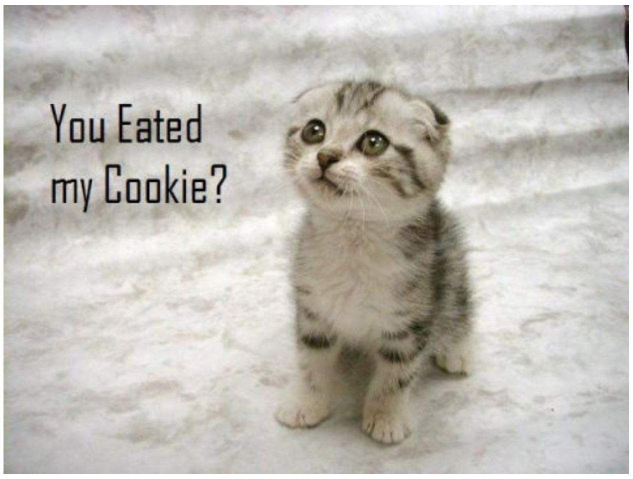

Fig. 4. The following figure shows how a content-type is usually developed in the mentioned CMSs. Each of the files must be manually written.

a specific syntax to be suitable, but if the developers do not agree, then the solution itself is useless.

The only way to determine if the current syntax is acceptable, is to let the developers evaluate it. This questionnaire will assist us in answering the fundamental questions.

- Is there a need for a DSL?
- What concrete syntax do developers prefer?
- How long do the developers spend on creating customized content-types?
- How often do the developers create customized content-types?

From the survey we will be able to determine if CT-DSL is accepted by the communities.

W e have posted a link to a questionnaire created with Google Doc in all the official forums of the three chosen CMSs [12,13,14]. There are 15 questions in total divided into 4 categories; Experience with DSL, DSL and content types and Prototype DSL for creating customized content types. The questionnaire is designed to take around 10 minutes to complete.

Survey results. The Following results is from the survey. The results answer the fundamental questions. Level of participation: 21.

24 of the participants answer that they create customized content-types within 1-3 months.

How long would you consider being sufficient for creating content-types in your primary native implementation language?.

Daily	8
Weekly	6
Monthly	9
1-3 months	12
3-6 months	3
6-12 months	5
Never	6

5 The References Section

In order to permit cross referencing within LNCS-Online, and eventually between different publishers and their online databases, LNCS will, from now on, be standardizing the format of the references. This new feature will increase the visibility of publications and facilitate academic research considerably. Please base your references on the examples below. References that don't adhere to this style will be reformatted by Springer. You should therefore check your references thoroughly when you receive the final pdf of your paper. The reference section must be complete. You may not omit references. Instructions as to where to find a fuller version of the references are not permissible.

We only accept references written using the latin alphabet. If the title of the book you are referring to is in Russian or Chinese, then please write (in Russian) or (in Chinese) at the end of the transcript or translation of the title.

The following section shows a sample reference list with entries for journal articles [1], an LNCS chapter [2], a book [3], proceedings without editors [4] and [5], as well as a URL [6]. Please note that proceedings published in LNCS are not cited with their full titles, but with their acronyms!

References

1. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. 147, 195–197 (1981)
2. May, P., Ehrlich, H.C., Steinke, T.: ZIB Structure Prediction Pipeline: Composing a Complex Biological Workflow through Web Services. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 1148–1158. Springer, Heidelberg (2006)
3. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
4. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181–184. IEEE Press, New York (2001)
5. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration. Technical report, Global Grid Forum (2002)
6. National Center for Biotechnology Information, `http://www.ncbi.nlm.nih.gov`

Appendix: Springer-Author Discount

LNCS authors are entitled to a 33.3% discount off all Springer publications. Before placing an order, the author should send an email, giving full details of his or her Springer publication, to `orders-HD-individuals@springer.com` to obtain a so-called token. This token is a number, which must be entered when placing an order via the Internet, in order to obtain the discount.

6 Checklist of Items to be Sent to Volume Editors

Here is a checklist of everything the volume editor requires from you:

☐ The final LaTeX source files

☐ A final PDF file

☐ A copyright form, signed by one author on behalf of all of the authors of the paper.

☐ A readme giving the name and email address of the corresponding author.