A Meta-plugin for Bespoke Data Management in WordPress

Stefania Leone, Alexandre de Spindler, and Moira C. Norrie

Institute for Information Systems, ETH Zurich
CH-8092 Zurich, Switzerland
{leone,despindler,norrie}@inf.ethz.ch

Abstract. WordPress is a powerful and extensible platform for web-based information publishing and management. While the WordPress core is targeted to the publication of chronologically ordered textual articles typical of blogs, users have developed plugins as well as themes to support the data management requirements of specific domains such as e-commerce or e-learning. However, the creation of such plugins requires development skills and effort. We present a meta-plugin that automatically generates bespoke plugins for data management based on user-defined ER models. We illustrate the approach using an example of creating a WordPress site for managing information about courses.

Keywords: Wordpress, Meta-Plugin, Data Management Platform

1 Introduction

WordPress is a powerful information management and publishing platform that allows end-users to set up their web sites by selecting and adapting shared themes. Through the administrator interface, users can customise their theme of choice as well as authoring content, uploading media and integrating a wide variety of plugins. The success of the approach is reflected by the large number of over 50 million online web sites[1] based on WordPress.

Although WordPress is commonly associated with blogging sites, nowadays it is widely used as a general information management and publishing system. Examples include museums[2], corporate websites[3] and more domain-specific applications, such as online stores[4] and e-learning systems[5]. However, while WordPress provides a powerful infrastructure for web publishing, the core data model is very limited since it is based on a simple pages and posts paradigm for the publication of semi-structured text and embedded media.

The WordPress core model and functionality can be extended through plugins. Thousands of plugins have been developed by the community with examples

[1] http://en.wordpress.com/stats
[2] http://wordpress.org/showcase/the-toledo-museum-of-art
[3] http://wordpress.org/showcase/atlantic-southeast-airlines
[4] http://kartellstorela.com
[5] http://testdatei.schatzverlag.ch

X.S. Wang et al. (Eds.): WISE 2012, LNCS 7651, pp. 580–593, 2012.

including an e-commerece plugin[6] and the Buddypress[7] plugin for the design of social networking sites. Plugins can easily be shared among the user community, but users with very specific data management requirements may have to develop their own plugins which requires knowledge of PHP as well as a detailed understanding of the WordPress platform and its inner workings.

To simplify the task for end-users, we show how the concept of a meta-plugin can be used to generate bespoke plugins for data management. The meta-plugin allows end-users to specify an ER model in the WordPress administrator interface and automatically generates a plugin based on this model. This enables users to profit from the infrastructure provided by WordPress, while being able to tailor the underlying model to their needs rather than having to work around the limitations of a core model originally designed for blogging sites.

Section 2 discusses support for end-user development of data-intensive web sites. The core model of WordPress is presented in Sect. 3 and an extension to support ER models in Sect. 4. In Sect. 5, we detail the concept of meta-plugins and demonstrate their use in Sect. 6. Concluding remarks are given in Sect. 7.

2 Background

Nowadays, many professional as well as private web sites are developed by single users, either as end-users or developers with a small amount of technical knowledge combined with some design skills. In line with research on end-user development, it is therefore important to consider how to make web information systems not only easy to use, but also easy to develop [1].

Within the web engineering research community, model-driven approaches to web site development such as [2–4] have been advocated strongly. These methodologies offer systematic approaches based on models defining the structural, navigational and presentation aspects of a web information system. In the case of WebML [2], a developer designs an ER data model, followed by navigation and presentation models. The associated tool Web Ratio [8] can then generate and deploy the application. While model-driven approaches are powerful and can generate complex web sites with little or no programming by the developer, they are not targeted at end-user development since they still require detailed knowledge of the models and how the web functions. Rather, they were aimed at supporting teams of developers where there should be a clear separation of concerns between database developers, web architects, programmers and designers.

End-users typically use a platform that offers document-based content publishing and allows users to design their web sites by configuring the content and structure of the site in terms of general publishing units and presentation styles. Popular platforms include WordPress and Drupal[9].

[6] `http://wordpress.org/extend/plugins/wp-e-commerce`
[7] `http://buddypress.org`
[8] `http://www.webratio.com`
[9] `http://drupal.org`

In the case of WordPress, the core model was targeted at blogging sites and the content is organised in terms of the two basic textual publishing units posts and pages that can be enriched with media. Further, WordPress users can employ a design-by-example approach [5] by selecting one of many web site themes developed and shared by the user community. This combination of a simple model and design-by-example enables users to set up a blogging site and start producing content within minutes. A major benefit of such a platform is that both the configuration and content of a web site can be updated dynamically. While Drupal is less blogging-specific, so called distributions provide pre-configured installations with similar support for setting up web sites.

WordPress features a plugin mechanism with which the original blogging model can be extended in terms of additional entity types, management operations and user interface widgets. A similar mechanism is present in Drupal, allowing developers to bundle reusable application components into modules. As a result, the development of web applications not only consists of extending and configuring a basic initial site with existing plugins or modules, but also optionally includes the development of such extensions.

Web information systems usually have a data management component and some researchers have addressed the problem of end-user development of data-intensive web sites. WYSIWYG application editors [6, 7] have been proposed to allow end-users to specify custom data. For example, the editor presented in [6] supports a top-down approach where a user specifies the presentation layer by creating forms representing domain entities. Based on these forms, an ER graph is extracted and the corresponding database schema is created automatically along with the presentation views. Visual mashup editors such as MashMaker [8] and Mash-o-matic [9] have been designed to create web information systems by integrating existing data sources. However, they do not provide the basic infrastructure to facilitate the design of new web information systems.

Drupal offers a module that supports the definition of custom data types, together with the generation of user interfaces to manage data. Similar support is provided in WordPress through plugins[10]. However, in both cases, custom data types are not explicitly represented in the database back-end. WordPress plugins typically make use of a single key-value table per data type instance to attach attributes to individual entities in a semi-structured manner. In Drupal, attribute declarations and values are represented on a meta level using two tables, one containing all declarations and another containing all values with references to the entities containing these values. Note that the WordPress plugin referenced does not support the association of entities. Drupal entities may reference each other by means of dedicated attributes containing entity identifiers. This effectively realises a single generic relationship construct rather than enabling custom relationships with constraints over source and target entities. Furthermore, since generic relationships are not explicitly represented in the database, referential integrity must be maintained as part of the application code.

[10] e.g. Ultimate Post Type Manager:
`http://wordpress.org/extend/plugins/ultimate-post-type-manager`

These approaches make it much more difficult to write application-specific queries than in the usual representation of ER models and may lead to poor performance. Moreover, it is difficult to reuse and extend custom types as the application evolves and to integrate custom data with external systems. While plugins and modules offer general extension mechanisms and developers could customise how they represent application data, such extensions require programming skills and a deep understanding of the platform-specific programming model. Consequently, their development is not a suitable option for end-users.

We decided to investigate ways in which platforms such as WordPress and Drupal could be extended to support the development of web information systems with application-specific data management requirements. In web information system development, data requirements are typically modelled using a structured data model such as ER. The general idea is similar to the work in [10], where they introduce a domain-specific language to support the generation of corporate web sites on top of popular wiki software. The approach that we adopted was to build on the powerful concept of plugins in WordPress and produce a meta-plugin that can generate data management plugins based on user-defined ER models.

3 WordPress Data Model

In this section, we will have a detailed look at the concepts supported by WordPress and its core data model. Note that, although the main concepts and terminology are introduced in a document describing the WordPress Semantics[11], the details of the core data model can only be established by examining the underlying database as well as extracting bits and pieces from various articles in the WordPress documentation [11]. Figure 1 gives a conceptual overview of the WordPress core concepts and how they relate to each other.

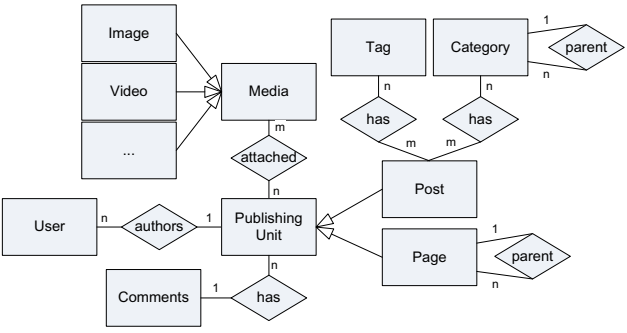

Fig. 1. ER model of WordPress concepts

WordPress distinguishes between static and dynamic content using two types of publishing unit—pages for static content and posts for dynamic content. When

[11] `http://codex.wordpress.org/WordPress_Semantics`

designing a web site, pages are typically used for content that has a fixed location within a web site and changes rarely. In the case of a blogging site, this could be information about the author. A page has a title, content and date. Content may be pure text, which is often enriched with HTML for structuring, or may also embed media, such as images, videos and audio files. Pages are usually accessed over a navigation menu and can exhibit nested structures.

In contrast, posts are used to publish new content. Although the content of an individual post is usually static, the collection of posts is dynamic and often presented in reverse chronological order with only the latest posts being visible in prominent positions such as the home page. This means that the location of a post within a web site changes over time and it becomes less and less visible to the users. Similar to pages, a post has a title, content and date. Both pages and posts may have comments, which can be configured by the developer.

To structure and organise posts, users can tag them or assign them to specific categories. Both tags and categories are user-defined. While tags represent a flat, user-defined taxonomy, categories typically have a hierarchical structure. The categories may be used to provide further navigational structures within a web site by creating corresponding menu items and showing only posts belonging to that category on the associated web page. Details of the structure and layout is controlled by the selected theme.

While Fig. 1 shows the view of the WordPress model presented to end-users, the internal metamodel that would be used by the developers of plugins is somewhat different as indicated in Fig. 2. We have based this developer model on the OMG Meta Object Facility (MOF) [12], where a model is represented by means of metamodel concepts, model concepts and data.

The first thing to note is that while the end-user model distinguishes the concepts of posts and pages, internally these are both instances of a general `PostType` in the M-2 metamodel. For the sake of clarity, we have therefore labelled the corresponding classes for these publishing units as `BlogPost` and `Page` in the developer model. A `PostType` defines a name and a list of attributes where `Attribute` is defined by a name and a type.

The data model on level M-1 represents the WordPress core data model. Here, we focus on the various post types since this is the extension point for data management plugins. We have therefore shaded out the parts of the core model dealing with other concepts such as taxonomies and users on the left of the figure and will not deal with them in detail.

WordPress actually offers five default post types for publishing content, which include attachments (any media file), revisions and navigation menus as well as pages and blog posts [11]. Furthermore, developers are free to create customised post types for publishing content that is structured differently. The design of customised post types is done programmatically and WordPress offers a number of methods in their API for the creation and registration of such custom types. Technically, the design of custom types is encapsulated and realised as plugins, which define the data types as well as the associated behaviour and presentation. Plugins will be discussed in more detail in Sect. 5.

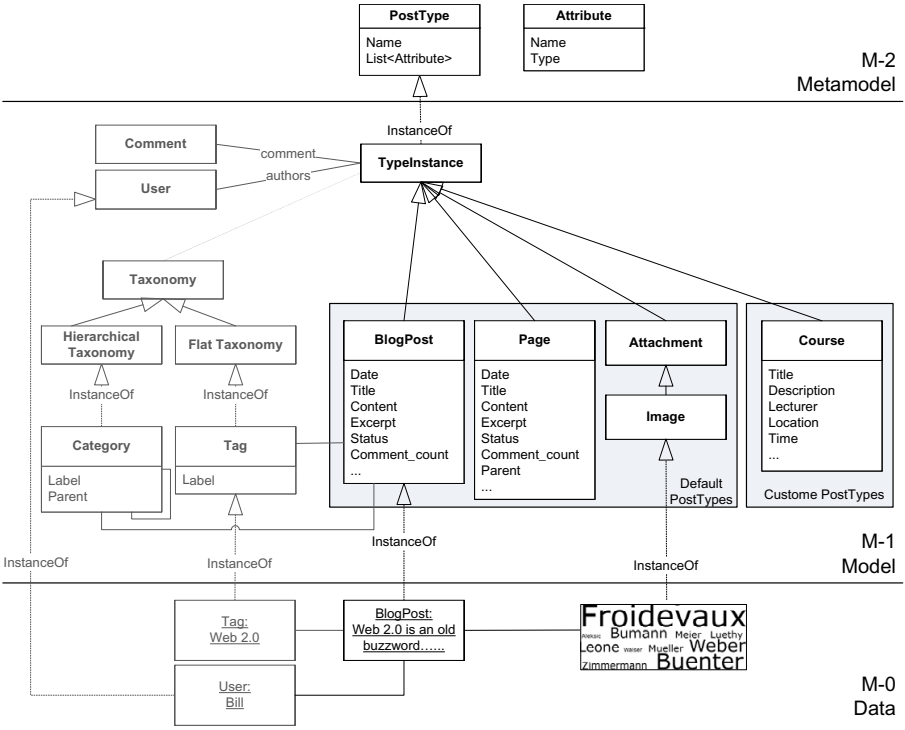

Fig. 2. WordPress developer model

In the centre of Fig 2, we show three of the five default post types, namely `BlogPost`, `Page` and `Attachment`. There are some interesting differences between pages and blog posts that can also be seen in the model. Note that only blog posts can be associated with tags and categories while only pages can be nested.

On the M-0 level, we indicate actual data instances such as a blog post about Web 2.0 with an image of a tag cloud as an attachment. The article has been written by a user named Bill and tagged with the term 'Web 2.0'.

Now consider the case where a developer might want to publish some structured data on a web site. For example, they might have the task of creating a WordPress site to publish and manage information about courses offered by a university. Rather than trying to manage the data about courses as text contained within pages or posts, a developer could create a plugin defining a custom post type `Course` as shown in Fig 2. This post type is specifically targeted at publishing information about university courses, specifies attributes such as title, description, time and location and is involved in a relationship associating courses with their lecturers.

Our goal was to enable end-users to create such plugins in the administrator interface without requiring detailed knowledge of the internal WordPress model or programming effort. We did this by extending the metamodel and then

creating a meta-plugin that could generate plugins automatically based on an ER model defined by the end-user through a form-style interface. We will first detail the extension to the metamodel before describing the meta-plugin.

4 WordPress Core Extension for Supporting ER Models

Having introduced the WordPress core data model, we will now show how the model can be extended based on user-defined ER models. We will present the extensions using the example of the course management system. Assume the ER model for the system is as shown in Figure 3.

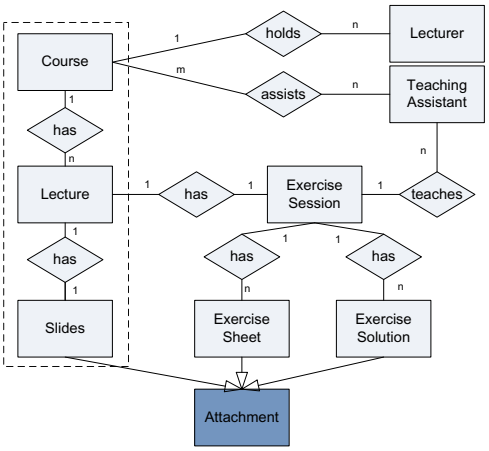

Fig. 3. ER model of university course management

Each course has one or more lecturers and one or more teaching assistants. A course has a number of lectures with associated slides. Each lecture may have an exercise session and, for each exercise session, there are exercise sheets and solutions. Also, an exercise session is taught by one or more teaching assistants.

Figure 3 also indicates how certain concepts of the ER model relate to concepts in the WordPress core model. Slides, exercise sheets and solutions are sub-entities of `Attachment` which is shaded to indicate that this is a concept of the core WordPress model shown in Fig. 2.

Figure 4 shows the extended WordPress model using MOF. On the metamodel level M-2, we have introduced a number of new concepts which complement the core concepts post type and attribute with other concepts of the ER metamodel. The `PostType` corresponds to an entity type and defines a number of attributes. An `EntitySet` is used to manage entities of a specific `PostType`. Entity sets can be related to other entity sets via n-ary relationships. A `Relationship` defines a name and list of attributes as well as cardinalities. While we named the cardinalities `source` and `target`, it is simply a naming convention since relationships are not directed. These concepts are used to instantiate an ER model on the M-1 level.

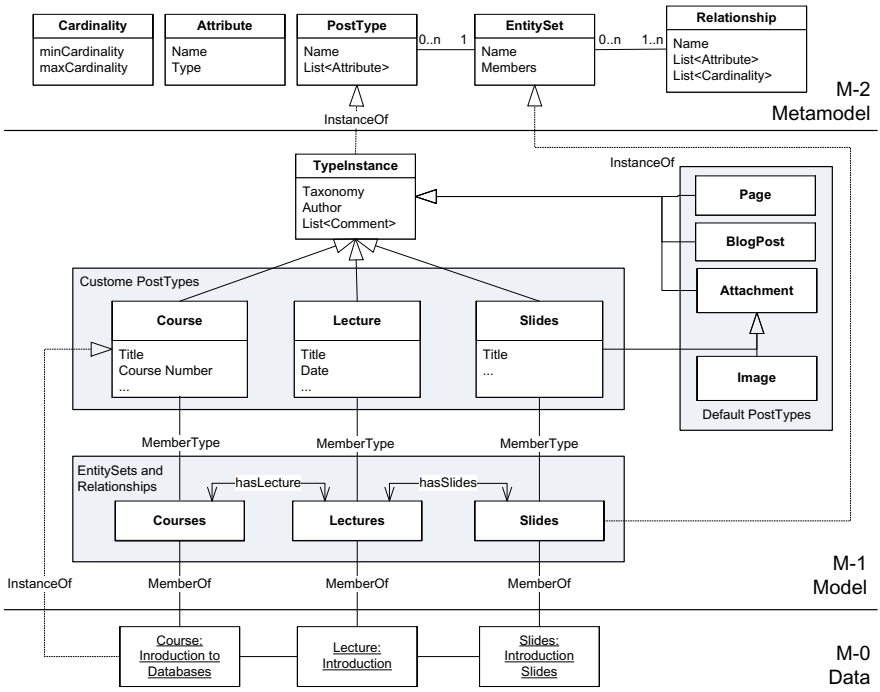

Fig. 4. WordPress model with course extension

The WordPress data model for our course management system is shown on the M-1 level. For the sake of space, we only show the additional concepts corresponding to the part of the ER model contained with the dashed line in Fig. 3, i.e. courses, lectures and slides. We also omit other parts of the core model such as the users and taxonomies.

Three new post types `Course`, `Lecture` and `Slides` have been integrated into the WordPress model, with the type `Slides` as a subtype of `Attachment`. Instances of these types are managed in the corresponding entity sets, namely `Courses`, `Lectures` and `Slides`, as indicated by the `MemberType` association between post types and entity sets. These entity sets are all instances of the `EntitySet` on level M-0. The relationships defined in the ER model can be represented using regular relationships between the entity sets. Note, however, that ER relationships that define attributes may be represented as entities in their own right that associate two entity sets.

On the M-0 level, data objects are shown. On the left, there is a course instance with the title 'Introduction to Databases'. The course has an associated lecture object which represents the introductory lecture of the course and is associated to the introduction slides on the right. These objects are instances of the newly defined post types and members of the corresponding entity sets, as indicated by the `InstanceOf` and `MemberOf` associations.

5 Meta-plugin

We now introduce in detail the notion of a WordPress meta-plugin, which is a plugin that generates new plugins. The idea is similar to that of template-based programming, for example in C++ [13] or XSLT [14].

The meta-plugin introduces a new data design process into the WordPress core. Instead of only being able to create pages and posts, a user can use the meta-plugin to define an ER model specifying application data entities and associations. From the specified ER model, the meta-plugin generates a bespoke plugin that the user simply has to install to create a data-backend for their web site. The generated plugin will allow application data to be created and manipulated based on the defined structure. Of course, the plugin can be used in combination with the powerful plug-n-play infrastructure provided by WordPress. This means that the structure provided by the generated plugin can be extended with additional pages and posts to refine the design, either using standard WordPress functionality or by installing additional plugins. Since all data entities are realised as post type instances, they can also be classified by means of categories and tags without additional development effort. For the look-and-feel, we rely on WordPress themes, which can be used and adapted by the user.

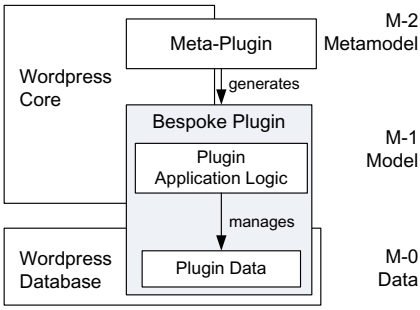

Fig. 5. Meta-plugin, plugin and data

Figure 5 gives an overview of our extension. The WordPress metamodel extension on level M-2 of Fig. 4 has been realised as a meta-plugin that extends the WordPress core with ER modelling capabilities and allows users to define ER models through a graphical user interface. Based on these user-defined ER models, the meta-plugin automatically generates bespoke plugins that realise the user-defined ER models on the M-1 level. Again, the generated plugin extends the WordPress core model, in this case, however, with a bespoke data model. In our example, that would be support for course management. The bespoke plugin consists of application logic and also an extension to the WordPress database to manage the data of the generated plugin. In our example, this would be data about courses, exercises, lecturers, assistants etc. The generated application logic offers functionality to create, manipulate and also view this data.

The meta-plugin as well as the generated plugins are realised as regular WordPress plugins that can be installed through the WordPress administrator interface, i.e. the dashboard. Once installed, the meta-plugin extends the WordPress dashboard with functionality for ER modelling. More concretely, it creates a menu item 'ER Modelling', with sub-menus to view and create entity types, entity sets and relationships between them, as shown in the menu bars on the left of Figs. 6 (a) and (b). Using these menus, the user can create their ER model: Fig. 6 (a) shows the interface for creating a new entity type and Fig. 6 (b) for creating a relationship. In the current example, a number of entity sets have already been created and the user can select the source and target entity sets of the relationship from the drop-down menus. Once the user has defined an ER model, they can trigger the generation process of the bespoke plugin using the 'Generate Plugin' menu. The user has to provide a plugin name and a description. The generation of a bespoke plugin is then triggered.

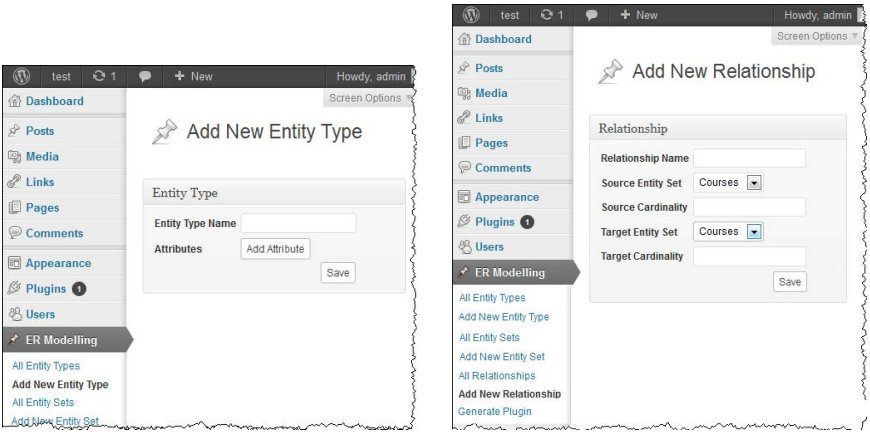

Fig. 6. Meta-plugin screenshots

6 Bespoke Plugin Generation

As part of its plugin mechanism, WordPress offers a number of hooks, which allow users to inject additional functionality, data structures and presentation into execution environment of the WordPress core. Hooks are plugin lifecycle events such as their installation or uninstallation, as well as administrative or end-user activities including the creation, manipulation, retrieval, selection, display and deletion of posts, pages or plugin-specific data. Typically, plugin code includes functions for creating and deleting database tables, for inserting and selecting table data and the assignment of these functions to particular hooks.

To install a plugin, the files containing the plugin code need to be uploaded into the target WordPress platform through the dashboard. The availability

of the plugin is then displayed to the user and can be activated. As a result of the activation, the additional functionality, data structures and presentation facilities become part of WordPress and are available for immediate use.

The meta-plugin presented previously is a plugin capable of generating files which constitute bespoke plugins. The code contained in these files is created using parametrised code templates instantiated with information from the ER models. The generated plugin encapsulates functions for the creation of custom types and database relations corresponding to the ER model, the extension of the dashboard with functionality to create data according to the model and functionality to present this data. We will now describe how these functions are bound to the WordPress hooks.

On plugin installation, custom types and database relations need to be created, and the dashboard extended with data creation and management functionality. First, a function is generated, which uses the custom type registration facility in order to register each entity type. As a result of such a registration process, the dashboard is automatically extended with the functionality to manage entity type instances. Then, for each entity type, entity set and relationship, functions containing `CREATE TABLE` and `DROP TABLE` statements are generated.

```
...
$plugin.=
"<?php
$postTypeRegistration;
register_activation_hook(_FILE_,
   '".$n."_activate');
...
function ".$n."_activate(){
   global \$wpdb;
   \$wpdb->query(
  'CREATE TABLE ".$tablename."(
      ID INT(6) PRIMARY KEY
        NOT NULL AUTO_INCREMENT,
      post_id BIGINT(20),
      ".$attributes.");'
  );
  ...
}?>"
...
```

Fig. 7. Parameterised PHP Template

```
<?php
register_post_type('Course',$args);
...
register_activation_hook(_FILE_,
   'coursemgt_activate');
...
function coursemgt_activate(){
  global $wpdb;
  $wpdb->query(
   'CREATE TABLE course(
      ID INT(6) PRIMARY KEY
        NOT NULL AUTO_INCREMENT
      post_id BIGINT(20),
      title VARCHAR(45) not null,
      description VARCHAR(256),
      location VARCHAR(45);');
}
...
?>
```

Fig. 8. Generated PHP File

Figure 7 shows an excerpt of the PHP template used to generate these functions. Upon template instantiation, variable `$postTypeRegistration` is replaced by a code snippet that registers all the entity types defined in the ER model as custom types. Then, a parametrised function invocation of the activation function registration is executed, where the activation function name is passed as a string, followed by a parametrised `CREATE TABLE` statement.

In Fig. 8, we show an excerpt of the resulting code for our course management plugin example. First, the course custom type is registered, followed by the registration of the activate function with the activation hook. On plugin activation, the function `coursemgt_activate()` is invoked and, as a first step, a database table for the course entity is created.

Below, there is an excerpt of the relational model in the underlying WordPress database after the activation. The first relation `WP_posttype` is the WordPress relation used to store all post type instances, be it pages, posts or custom post type instances. The following relations `Course`, `Session` and `HasSession` have been created upon plugin activation. Whenever a new entity, e.g. a course, is created, WordPress automatically generates a new tuple for the `WP_posttype` relation. In addition, our plugin creates a tuple in the `Course` relation containing all the course data values, with a foreign key to the `WP_posttype` tuple.

WP_posttype (postID, title, content, date, postType_id, ...)
...
Course (courseID, postID, description,...)
Lecture (sessionID, postID, title, date, ...)
HasLecture (id, courseID, sessionID)
...

The relation `Lecture` is used to manage lecture tuples and, analogously to the `Course` relation, it defines the `postID` as a foreign key. The `HasLecture` relation represents the relationship between courses and lectures. Note that, in the current implementation, cardinality constraints are handled in the application logic and, therefore, every ER relationship is realised as an M:N relationship. Moreover, entity sets are currently represented by all the tuples of a relation and, hence, a course entity is automatically a member of the `Courses` entity set.

The `coursemgt_activate` function also contains code that extends the dashboard with a new menu item and sub-menus that offer support for the creation and manipulation of entities, as well as the functionality to associate entities from one entity set to entities from another entity set as defined in the ER model. In the screenshot in Fig. 9, the administrator interface of the generated course management plugin is shown.

The screenshot shows the interface to create a new course, by specifying course title, course description and location. In addition, the interface offers the possibility to relate the current entity to other entities according to the data model. In the current example, the generated interface offers the possibility to associate courses to lecturers and assistants directly. Each time a new entity is created, functions that are registered to the `save_post` hook are invoked. In our case, a function `save_entity_info` is registered, which checks for the instance's post type, and creates an entry in the corresponding database table.

For each post type, code is generated that specifies how the post type is displayed. In order to display data, the plugin registers a function to the `the_post` hook, which is invoked when data is loaded. The registered function checks for the post type of the instance that is to be displayed. Based on that, the appropriate data is retrieved from the corresponding database table and displayed

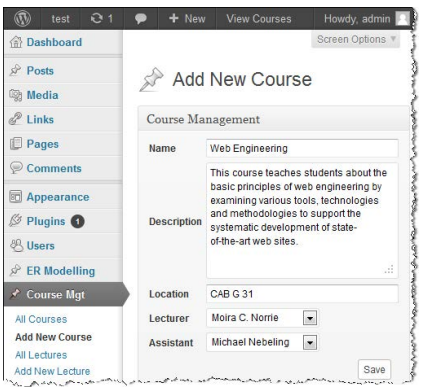

Fig. 9. Course management admin

Fig. 10. Course management example

according to the presentation defined for that specific custom post type. Entity sets are currently represented as pages containing a list of entities. Users can click on any entity to see a detailed view of the entity with all the attribute values. Also, for relationships defined between entities, we provide links in the entity detail view to navigate from one entity to an associated one. For the layout of the application, the user can use the regular WordPress themes to define the look and feel of their application. Given the use of the `the_post` hook, templates are kept completely independent of the post types.

The screenshot in Fig. 10 shows the generated course management web site, in this case for the courses of our research group. Links have been created from the selected course to the lecturer and assistant.

Table 1 provides a complete overview of all WordPress hooks and meta- as well as bespoke plugin functions assigned to them.

Table 1. WordPress hooks and high-level plugin functionality descriptions

Hook	**Functionality**
Meta-Plugin	
Display administration interface (`admin_menu`)	Add support for the definition of entity types and relationships
Bespoke Plugin	
Installation (`register_activation_hook`)	Create database relations realising the ER model
Display administration interface (`admin_menu`)	Add support for the management of entities and relationships
Create post (`save_post`)	Insert attribute values and relationship tuples into corresponding database relations
Display post (`the_post`)	Retrieve and display data from corresponding database relations
Uninstallation (`register_deactivation_hook`)	Drop relations created during installation

7 Conclusion

We have presented an approach that complements the rich web information publishing infrastructure provided by WordPress with support for application-specific data management requirements. We plan to further investigate how the design process can be facilitated and enriched, for example by introducing the notion of component themes customised to a specific post type that can be generated and composed to an overall theme. Having tightened the notion of relationships within WordPress, we are also beginning to distinguish kinds of relationships such as generalisation and inclusion relationships. Furthermore, we plan to exploit the fact that entity types, entity sets and relationships could be tagged and categorised in WordPress. In this way, it would not only be possible to enhance WordPress with ER modelling capabilities, but also to enhance the ER model with WordPress concepts.

References

1. Lieberman, H., Paterno, F., Wulf, V. (eds.): End User Development (Human-Computer Interaction Series). Springer (2006)
2. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): A Modeling Language For Designing Web Sites. Computer Networks 33(1-6) (2000)
3. Hennicker, R., Koch, N.: A UML-Based Methodology for Hypermedia Design. In: Evans, A., Caskurlu, B., Selic, B. (eds.) UML 2000. LNCS, vol. 1939, pp. 410–424. Springer, Heidelberg (2000)
4. Vdovják, R., Frăsincar, F., Houben, G.J., Barna, P.: Engineering Semantic Web Information Systems in Hera. Journal of Web Engineering 1(1-2) (2003)
5. Lee, B., Srivastava, S., Kumar, R., Brafman, R., Klemmer, S.R.: Designing with Interactive Example Galleries. In: Proc. ACM Intl. Conf. on Human-Computer Interaction, CHI 2010 (2010)
6. Yang, F., Gupta, N., Botev, C., Churchill, E.F., Levchenko, G., Shanmugasundaram, J.: WYSIWYG Development of Data Driven Web Applications. Proc. VLDB Endow. 1(1) (2008)
7. Karger, D.R., Ostler, S., Lee, R.: The Web Page as a WYSIWYG End-User Customizable Database-backed Information Management Application. In: Proc. ACM Symposium on User Interface Software and Technology, UIST 2009 (2009)
8. Ennals, R., Brewer, E., Garofalakis, M., Shadle, M., Gandhi, P.: Intel Mash Maker: join the web. SIGMOD 36(4) (2007)
9. Murthy, S., Maier, D., Delcambre, L.: Mash-o-Matic. In: Proc. DocEng. (2006)
10. Díaz, O., Puente, G.: A DSL for Corporate Wiki Initialization. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 237–251. Springer, Heidelberg (2011)
11. Wordpress.org: Wordpress Documentation (2012), `http://codex.wordpress.org`
12. OMG: Meta Object Facility (MOF) Core Specification Version 2.0 (2006), `http://www.omg.org/cgi-bin/doc?formal/2006-01-01`
13. Prata, S.: C++ Primer Plus, 5th edn. SAMS (2005)
14. Grossniklaus, M., Norrie, M.C., Büchler, P.: Metatemplate Driven Multi-Channel Presentation. In: Proc. Workshop on Multi-channel and Mobile Information Systems, WISEW 2003 (2003)