Mixing and Mashing Website Themes

Linda Di Geronimo, Alfonso Murolo(✉), Michael Nebeling, and Moira C. Norrie

Department of Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland
{lindad,amurolo,nebeling,norrie}@inf.ethz.ch

Abstract. WordPress offers users a wide choice of themes defining the structure, functionality, layout and presentation of a website together with its content types. These themes are shared by the WordPress community, enabling users to benefit from the skills of others. However, it is not possible to mix themes, so users often have to choose from a set of themes that only partially meet their requirements. We have developed a theme editor that allows users to combine both static and dynamic elements of existing themes using simple drag-and-drop operations. These elements are adapted to reflect the content and structure of the website under construction so that there is no distinction between design-time and run-time. We discuss in detail technical challenges along with our solutions for developing such an editor and integrating it into the WordPress platform. Further, we describe how the solutions could be generalised to other modern content management systems.

Keywords: Website theme · Theme generator · Web development tool · Content management system

1 Introduction

WordPress[1] has gone well beyond its origins as an open source blogging platform to become the most widely used content management system (CMS) with over 60% of the market share and estimates that it is used in nearly a quarter of the top 10 million websites[2]. Many of these sites manage large amounts of data and offer rich functionality including the integration of third party services.

Each WordPress site is an instance of a theme which defines the structure, functionality, layout and presentation of the website as well as the types of content to be managed and published. One of the main reasons for the initial popularity and widespread adoption of the platform was its support for end-user development of websites. Users could set up their website in a matter of minutes by selecting a theme, specifying a few customisation options through an administrative dashboard and adding content. In this way, they could not only create a website without any coding or deployment effort but also benefit from the design skills of other users who shared their themes.

[1] http://www.wordpress.org
[2] http://www.w3techs.com, 23.7% on 8 April 2015.

P. Cimiano et al. (Eds.): ICWE 2015, LNCS 9114, pp. 34–51, 2015.
DOI: 10.1007/978-3-319-19890-3_4

Over time, not only has the platform been extended to provide richer functionality and improved support for both end-users and developers, but the WordPress community has itself developed and shared vast numbers of themes. Furthermore, the WordPress platform provides a simple means for developers to extend the functionality of a theme through its plugin mechanism and the community has also developed and shared thousands of plugins. While some themes and plugins have been developed by professional agencies, many of them are freely shared within the community. For example, over 37'114 plugins are available at `wordpress.org`[3]. WordPress can therefore be considered as one of the most successful and influential examples of the power of crowdsourcing.

However, one limitation of the theme concept is that it only supports all-or-nothing reuse since it is not possible to mix elements of different themes. Consequently, users are forced to choose from a set of candidate themes each of which may only partially meet their requirements. To address this issue, we have developed a visual theme editor that allows users to compose their websites by mixing and mashing elements of existing sites. Both static and dynamic elements can be selected and reused via simple drag-and-drop operations. Developers can choose whether to retain the styling of the copied elements, apply the styling associated with the theme under creation or modify the styling via normal editing operations. When a dynamic element is reused, it is immediately adapted to reflect the content and structure of the website under construction. This means that developers can already see their website in operation at design-time. In this way, there is no distinction between design-time and run-time and, from the user point of view, no distinction between a website and a theme. To achieve this, it was necessary to integrate the theme editor into the WordPress platform and structure themes in terms of reusable components.

An overview of our approach of providing an editor that generates themes constructed from reusable components was presented previously in a short paper at ICWE2014 [1]. In this paper, we take the work further and present the main technical challenges and solutions of being able to dynamically mix and mash themes within the WordPress platform. Further, we discuss how the approach could be generalised to other popular modern CMS such as Drupal[4] and Joomla[5] which do not have the same theme concept as WordPress.

We start in Sect. 2 by reviewing existing WordPress theme generators as well as previous work addressing the technical challenges of reusing elements of websites. An overview of our approach is then presented in Sect. 3 before going on to provide details of how themes are structured in terms of reusable components in Sect. 4. Details of the steps involved in generating a theme and its components as well as dynamically reusing them in different content contexts are given in Sect. 5, with a review of the technical challenges and solutions in Sect. 6. We follow this in Sect. 7 with a discussion of how the approach could be generalised to other CMS. Concluding remarks are given in Sect. 8.

[3] http://www.wordpress.org, 8 April 2015.
[4] http://www.drupal.org
[5] http://www.joomla.org

2 Background

WordPress has evolved into a flexible and powerful platform capable of supporting a wide variety of websites. If a developer can find a theme that fully meets their requirements, the process of developing a website can be done through the dashboard where customisation parameters can be set, pages created, navigation menus defined and sidebars configured. The functionality of the theme can also be extended through the dashboard by selecting and adding plugins. However, as soon as a developer is faced with the task of adapting or extending a theme, they have to start working at the level of the HTML, CSS, JavaScript and PHP files as well as learning about the core WordPress model and system operation.

Developers often work on a need-to-know basis, learning only enough to solve the particular task at hand. Since WordPress offers developers a very loose framework in which to work, many different approaches are used to achieve the same look and functionality. Consequently, the documentation and tutorials vary a lot in terms of guidelines and solutions offered and it is clear from reading tutorial-style books on theme development, e.g. [2,3] as well as online forums[6], that many developers simply copy and paste bits of code with the hope that it will achieve the desired effects. However, often these attempts to reuse code fail because they are inconsistent with how other parts of the site have been developed.

A number of WordPress theme generators are available to support end-user development of themes. These focus on creating new themes from scratch but many of them have serious limitations. For example, Templatr[7] only allows users to select from a fixed set of layouts, while Lubith[8] enables users to customise layout via drag-and-drop operations, but does not support the customisation of functionality. Further, many generators are not integrated into the WordPress platform, so it is not possible to perform content-related tasks at design-time and it can lead to compatibility problems across versions. It was therefore our goal to develop our theme editor on top of, and fully integrated into, WordPress.

Other tools and frameworks, for example Themify[9], have been developed to facilitate the customisation of themes. The budget limitations of a customer usually determine the amount of customisation that can take place and hence tools that make it easier for developers as well as end-users to create or customise themes can have a major impact on the quality of websites produced. However, often the required customisations could be achieved by simply mixing elements of different themes but this is currently not supported. In a recent survey of 110 WordPress developers [4], 75% indicated that they would like to be able to mix the functionality of different themes, while 56% answered that they would like to be able to mix layout elements.

Approaches that allow end-users to design their websites by selecting and combining parts of existing websites have been explored by researchers in the

[6] for example, http://www.wpbeginner.com
[7] http://templatr.cc
[8] http://www.lubith.com
[9] http://themify.me

HCI community [5,6]. Their studies demonstrated the benefits of the design-by-example paradigm, but their solutions only addressed the reuse of elements of website design in terms of layout and presentation and not the dynamic aspects dealing with functionality and content. Modern websites tend to make heavy use of JavaScript and jQuery[10] and, rather than being static, pages are often dynamically generated. This is particularly true in the case of CMS in general, and WordPress in particular, where "the Loop" is used to define the content to be displayed in an element of a web page in terms of a database query and templates to extract data from the query result.

Extracting components from an existing web page involves identifying and extracting all the necessary HTML, CSS, JavaScript and resources. Various techniques for this have been proposed in the mashup research community. For example, Ghiani et al. [7] allow users to select mashup components from arbitrary websites through direct manipulation of the GUI. Note that to extract and reuse elements of WordPress themes, it is necessary to not only extract components of a web page, but also the PHP functionality of the theme defining the dynamic parts of the page, i.e. the code that generates these elements: This raises many new challenges that have not been addressed previously.

To support reuse, component models for web development have been proposed to ensure that pages are constructed from reusable components. WebComposition [8] was an early effort in this direction where they proposed an object-oriented support system for building web applications through hierarchical compositions of reusable application components. MashArt [9] is a system developed in the mashup community that enables advanced users to create their own applications through the composition of user interface, application and data components. More recently, an extension to WordPress was proposed that allows websites to be developed from a component model that supports composition at the data, application and interface levels [10]. The approach requires developers to model the different aspects of a website and specify the composition logic. In contrast, our goal is to support end-user development by allowing themes to be created using a visual editor where users can simply drag and drop elements of existing themes that encapsulate presentation, content and functionality.

As detailed in the next sections, our approach combines many features of the related work described above. To support reuse, we first defined a metamodel for themes that can be used to structure them in terms of reusable components. Second, we developed a theme editor that enables users to mix and mash elements of web pages at the GUI level, thereby hiding the details of the component model but ensuring that the created themes conform to the model. In an earlier paper [1], we described how this approach could be used to support the design-by-example paradigm advocated by the HCI community in contrast to the model-driven approaches proposed by the web engineering community [11–13]. Here we take this work further by detailing the technical challenges underlying the approach and the solutions that we developed. Specifically, we describe the implementation of our theme editor and how it was integrated into the WordPress platform.

[10] http://jquery.com

3 Approach

Our overall goal is to allow end-users and developers to create themes by reusing elements of existing themes that can be searched and browsed in an online gallery. We illustrate this in Fig. 1 by showing elements of two web pages that have been selected and copied into a web page under construction.

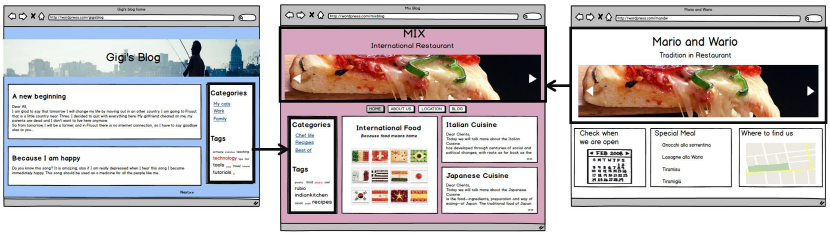


Fig. 1. Components from left and right web pages mixed at GUI level in middle page

At first sight, this appears very similar to the previous research within the HCI community where users can design a website by selecting elements from galleries of examples. However, there are important and far-reaching differences. First, as mentioned before, they only handle static web pages and have no support for the reuse of client-side functionality. Second, they only deal with web pages rather than with themes defining how web pages are generated from database content. We illustrate the different layers involved in extracting and reusing a component of a theme in Fig. 2.

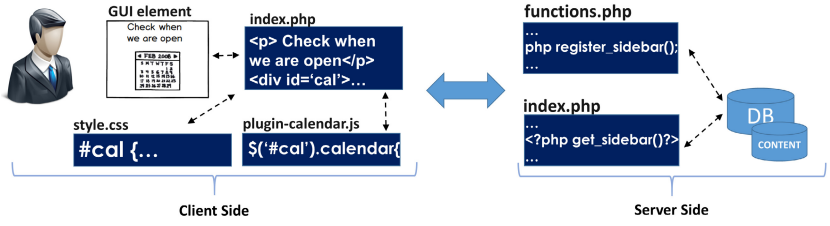


Fig. 2. Layers involved in extracting and reusing a theme component

On the left of Fig. 2, we show a rendered component of a web page and how it is defined in the underlying theme. A user should be able to extract and reuse this component by simply selecting the corresponding element within the rendered web page and copying it into their new page via drag-and-drop. They should also be able to choose whether to keep the source styling or adopt the target styling and be able to use basic editing operations to change its size, position or style. For more advanced users, there should be an option to switch to a mode where they can edit CSS code directly.

To achieve this, the corresponding DOM elements need to be identified and the required CSS and JavaScript code extracted along with the HTML. The first step of identifying the DOM elements is common to all projects dealing with the extraction and reuse of components of a web page. However, to import a selected element into a new theme, we also need to identify and extract the parts of the underlying theme that were responsible for generating them. This means that we need to extract code from the source theme's PHP templates stored on the server-side. We achieve this by exposing parts of the server-side code shown on the right of Fig. 2. Access is read-only and limited to the particular theme rather than the entire WordPress installation so that other software and users do not gain access to credentials and therefore control over the database and private data. It is further important to note that when the selected element is imported into the theme under construction, there is an immediate switch from executing queries against the WordPress database for the source theme to the database associated with the target theme.

In addition, as shown on the right of Fig. 2, some properties of a WordPress site such as content shown in a sidebar or a header are defined on the server-side and customised through the WordPress administrative interface and so we also need to handle these correctly when extracting and reusing elements.

To support our requirements, we have developed a visual theme editor with both design and reuse capabilities. This means that it can be used to design new themes from scratch or to compose new themes by mixing and mashing components of existing themes accessed in a gallery. This is important not only because it offers users full flexibility in how users create and customise their themes, but it also provides the initial motivation for users to participate that is essential to any crowdsourcing model [14]. By providing a visual editor with full capabilities for creating, positioning and styling both static and dynamic elements of a theme, including creating nested structures of arbitrary complexity, the functionality of the editor is comparable to that of the most powerful theme generators. The tool therefore has value to users and developers even without the ability to reuse components of existing themes. The themes generated by the tool are referred to as X-Themes since they are structured according to our metamodel and represented as a set of reusable components. As soon as an X-Theme is generated, it is added to the interactive gallery of existing themes accessible to the X-Themes editor and its components are immediately available for reuse. In this way, we can avoid the cold start problem and motivate users to participate.

The X-Themes editor is realised as a WordPress plugin and, once installed and activated, can be accessed via the main menu of the administrative interface. An advantage of making the editor available as a plugin is that it provides an easy means of deploying the tool to the vast developer community, while achieving our goal of integrating it into the WordPress platform.

4 Metamodel

A theme can be considered as a skeleton for a website that defines the essential form and function of the site with the dynamic content missing. It therefore defines the types of content, the structure and navigation of the site, the functionality, the presentation styles and any static content including images.

In the case of WordPress, a theme mainly consists of a set of PHP templates, CSS stylesheets and images. The templates are structured in a hierarchy to represent not only the home page and structural elements of pages such as header, footer and sidebar, but also templates for displaying different kinds of content. Since the WordPress platform was originally developed for blogging sites, the basic content types are posts and pages. While a default template should be provided for pages that display posts, it is possible through a naming scheme to construct a whole hierarchy of page templates ranging from customised pages for specific posts and categories of posts to a generic post page. Details of the WordPress model including the full template hierarchy are given in [15].

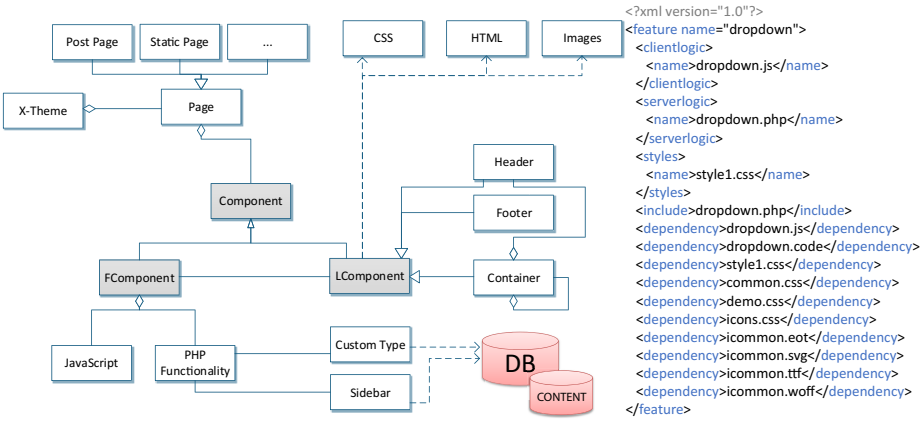
The model underlying the WordPress system is not as well-defined or documented as research systems with clearly defined concepts and a metamodel. Also, there is a lot of flexibility in terms of how and where different aspects of a theme are defined. Most parts of themes are tightly coupled and often not kept separate, making it difficult to identify, extract and reuse them. Until now, separation was up to the developer who, if following good principles of design, could manually separate code components to support future reuse. But it is important to remember that many WordPress developers are part-designer/part-developer with limited training in principles of software engineering [4].

We therefore defined a metamodel for themes consistent with the WordPress model, but introducing a notion of Components. Fig. 3a shows the core elements of the metamodel that define the structure of a theme and references to each component's resources so that they can be further accessed and reused. Specifically, we have defined Component as the reusable super-type of the model. A component can be an *LComponent* or an *FComponent*. An LComponent specifies the layout structure of a part of the page, and it is defined by CSS and HTML markup. An FComponent embeds functionality which can be defined through PHP or JavaScript logic, the former on the server-side and the latter on the client-side.

In order to give complete freedom in design, LComponents can contain other LComponents, allowing an arbitrarily nested structure, as shown in Fig. 3a. However, although LComponents can be placed with freedom within the page, FComponents instead need to be linked to an LComponent, in which they are loaded and displayed.

The code defined in the FComponents can either be integrated directly into pages via inclusion, or through widgets which provide easy access to plugins. Plugins are a means of extending the functionality of a theme and may either be integrated into a theme or added later by a user.

Elements such as headers, footers and sidebars are part of the core WordPress model, and custom content types can be added to support application-specific



(a) X-Themes metamodel (b) XML example

Fig. 3. Metamodel and XML representation

data. For example, custom types could be defined in an e-commerce system to manage product data. The actual database and content are defined by an instance of a website and not the theme and therefore are not targeted by our reuse mechanism.

The generated X-Themes are structured so that all the resources and code required for each component are stored in a separate directory. The model distinguishes between server-side resources such as PHP templates, resources which enhance the client-side experience such as JavaScript and other required files such as images. For each component, we store a representation of this information in XML which is then used during theme generation to include all the required resources. The XML representation of a component is also used when a user drags-and-drops that component into a theme to identify which resources need to be loaded into the editor and displayed.

An example of the XML representation of a component is shown in Fig. 3b. The elements *serverlogic*, *clientlogic* and *styles* are used to identify the primary resources needed for a component to work. Moreover, a general *dependency* element is used to specify additional resources required, such as images and non-standard fonts. An *include* element specifies the file responsible for starting the execution of component.

All elements of the metamodel also have a DOM-based implementation which annotates the design created within the editor by exploiting the HTML5 *dataset* API. These annotations are then used in the generation and the reuse steps detailed in Section. 5. For example, during theme design, the root element of the component for the navigation menu represented in Fig.3b will be annotated with the *data-clientlogic* attribute specifying a DOM reference to enable fast access from the JavaScript modules of our editor.

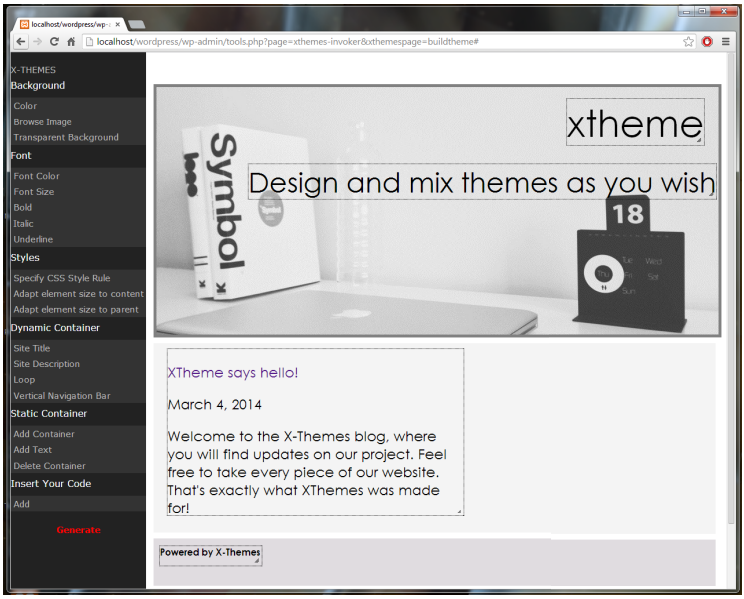


Fig. 4. X-Themes editor

We note that while the metamodel was designed around the original WordPress model, we were careful to design it in a such a way that it could be generalised to other CMS as described later in Sect. 7.

5 Implementation

The X-Themes editor is accessible through the dashboard of WordPress, and its interface is shown in Fig. 4. There is a menu on the left and the main three elements of a web page—header, body and footer—in the main editing area. Users can create an arbitrary nesting of containers within the main elements. The user can perform basic style customisations such as changing the font and background through menu options, while more expert users can also add and edit CSS rules directly.

Containers can be associated with functionality by creating or copying components into containers. To create a new FComponent, the user can import the necessary HTML, PHP, CSS and JavaScript files. When this is done, the editor performs two operations. First, it executes the FComponent directly and shows it running in the design being edited. Second, it creates a package, which is a zip archive, containing not only the source files specified by the user, but also the metamodel information as specified in Section 4. In this way, the tool creates a new FComponent which can be reused either via a drag-and-drop from the generated X-Theme accessed in an online gallery or through the zip archive. When the user is finished editing, they click on the *generate* button and their

X-Theme will be generated and immediately available for use via the WordPress dashboard, and for reuse via the X-Themes gallery.

We will now describe the generation process in detail referring to Fig. 5. We will first present the steps of the generation process shown on the left and then the reuse process on the right.

1. Design. When the editing of a theme is complete, the editor generates a set of templates for that theme together with the files defining its components and associated metadata based on the X-Themes metamodel. We will assume a simple example of a design with only two LComponents in the header and one in the body as shown in Fig. 6 to explain the steps.

1.a) **Design theme** Since our editor is a web-based tool, the design will be a subtree of the DOM structure of the page. It is important to know that the theme header, body and footer are handled as LComponents individually since this is how they are handled in WordPress. The DOM structure will contain the elements of the example together with the contents of the user's WordPress installation in the markup. As mentioned in Section 4, the root elements of the component subtrees have annotations based on the HTML5 *dataset* API to represent metadata for our model.

1.b) **Metamodel Generation**. A browser node recursively builds metamodel strings for each of the components present in the design. This involves recreating the XML structure, shown in Fig. 3b, by reading the HTML5 dataset *attributes*, reversing the mapping between the XML-based implementation of our metamodel and the DOM-based one, as explained in Section 4. In the case of Fig. 6, it builds an XML string for LC1, LC2 and FC as well as for the three default LComponents. Within the same recursive traversal, it also builds strings with

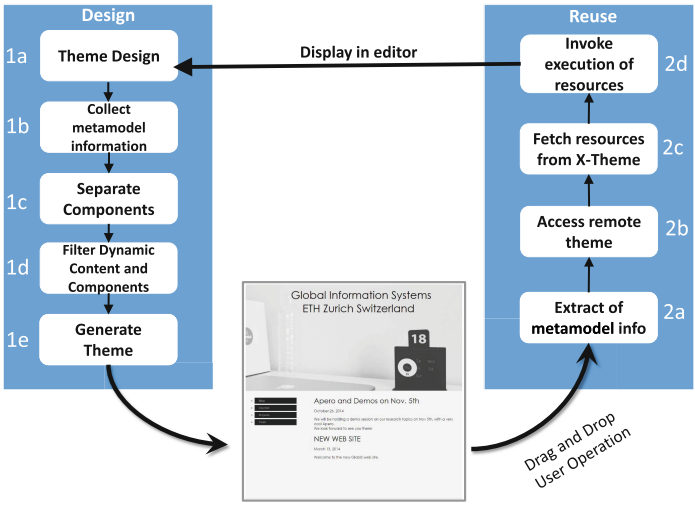


Fig. 5. X-Themes' process of generation and reuse

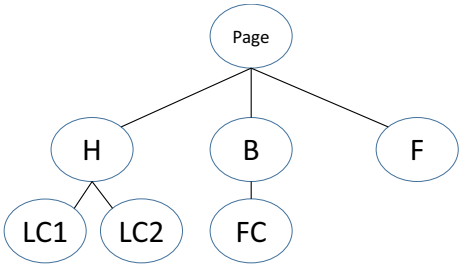


Fig. 6. Simple example design structure of an X-Theme

CSS style rules for each component so that they can be bound to the corresponding metamodel element, which from now on are referred to as *meta-elements*.
1.c) **Separate Components**. The first use we make of our XML specification is to distinguish the individual meta-elements and allocate a directory for each of them. For each meta-element, all files which have to be executed or loaded with it are saved in its directory, along with a copy of the PHP code (in a different format) that will allow future reuse. Also during this step, the CSS style information, which up until this point is represented as a string in inline rules, is separated out into CSS files. The theme's header, footer and body get processed in the same way. For the example in Fig. 6, this means it would create a separate self-contained structure for LC1, LC2 and FC nodes in the tree as well as for H, B and F. It is important to note that up to this point, the content originally shown to the user for each LComponent is still part of the markup saved to the PHP files.
1.d) **Filter Components**. The LComponents are analysed and code for any dynamic elements generated, including specific WordPress template functions, namely *template tags*. For example, assume that the node H in Fig. 6 contains two containers—one with a logo and one with the site's name. Our implementation operates over the theme DOM tree which in this case would have the parent node and its two child nodes. The site's name is a dynamic element since this information is contained in a site parameter and the corresponding PHP code `bloginfo('name')` would therefore be inserted into the container in place of its original output. Now that every meta-element is in the form of a self-contained package, we can replace their occurrence in the markup with an *include()* call to the PHP file responsible for executing it. Until this step, the theme's header, body and footer are processed in exactly the same way as any other LComponent. They now need special handling since their inclusions have to be in specific locations according to the structure of a WordPress theme, and this requires additional inclusions to be placed in some files for the theme to work.
1.e) **Generate**. The editor now creates an XML representation of the theme that defines all the components. This is useful for the gallery as it allows the complete theme structure to be analysed from a single source.

2. Reuse. On the right of Fig. 5, we show the main processing steps to extract an FComponent of an existing X-Theme for inclusion in the theme being edited. We name this approach **Clone-and-execute** since it performs local clones of the FComponent and executes it through an AJAX call accessing the WordPress API. It is important to note that we chose to develop a special approach for the reuse of simple LComponents and will explain this in our review of the implementation given in Section 6.
2.a) **Metamodel extraction**. When the user selects and drags an FComponent, such as a navigation menu implemented in PHP/JavaScript and CSS, the editor accesses the metamodel information of that FComponent and acquires references to the theme's location in the source web server. Moreover, for each file of a meta-element, each metamodel description contains references which are local to the source theme. These sub-references are also retrieved and act as an input for the next step.
2.b) **Theme Access**. The theme HTTP location and the local sub-references are now chained together to obtain an absolute HTTP-based location for each individual dependency of the meta-element itself. In this way, the editor propagates the selection and is ready to get the FComponent. Note that resources stored in remote WordPress installations first have to be cloned in order that the FComponent can be executed in the editor.
2.c) **Fetch resources**. The selected resources (JavaScript, CSS, PHP, images, etc.) are then fetched in order to clone them in the user's WordPress installation. The access is done via the HTTP protocol directly on the source theme server and the necessary connections made to download the required files. Note that only the resources in the component's directory get copied during this process. Remote resources that are referenced but not part of the WordPress installation do not get copied, for example a picture from a different website linked via a remote URL. Moreover, as remote access to PHP is not possible for security reasons, our implementation instead uses a copy of the FComponent's PHP code created when the X-Theme was generated.
2.d) **Execution**. After every resource has been selected, accessed and copied locally, the FComponent can be executed in the context of the X-Themes editor running on the user's own WordPress installation and accessing their database transparently. We perform this through an AJAX request to an endpoint meant to evaluate meta-elements coming from reused components and requiring access to the WordPress API. This endpoint is registered through some WordPress specific functions, namely *hooks*, which allow us to execute our code taking advantage of the WordPress API. Unfortunately, this approach comes with some disadvantages which will be explained in more detail in Section 6. Since our meta-element is sent to this AJAX endpoint within WordPress, all PHP and the WordPress-related queries can be executed and the result is used as output to the user. Once the AJAX response has been received, the editor will dynamically load any other CSS or JavaScript code that is required. As a result, the FComponent can directly display the actual contents of the target database rather than the source one. For example, any imported navigation menus, would imme-

diately reflect the structure and labels defined by the user in the administrative dashboard for the theme under creation rather than those that appear in the source theme.

Allowing users to design their own themes by letting them reuse any PHP or JavaScript code raises potential security issues. For example, it would be possible to have a malicious FComponent containing JavaScript code which accesses other components through the DOM and somehow modifies them. While out of scope at this stage, the topic of detecting and preventing such side-effects is one direction for possible future work.

6 Review of Technical Challenges and Solutions

As is often the case when a system is extended to support goals for which it was not originally designed, realising the theme editor as a WordPress plugin brought many challenges. Specifically, there were many issues that had to be addressed to enable run-time components to be dragged-and-dropped from a web page running in a browser and deal with coherence of the components, the performance/efficiency of reuse and the re-execution of such components in a totally new environment. Moreover, while realising the editor as part of the WordPress dashboard offered potential advantages in terms of deployment and acceptance in the community, it meant that some compromises were necessary to display the editor within the WordPress administration pages. We will first discuss the main choices and compromises that we had to make to coexist with the WordPress dashboard, and then detail the trade-offs related to efficiency and performance.

As shown in Section 5, our tool makes heavy use of AJAX requests. Our X-Themes editor is reachable from a page registered to the WordPress dashboard through hooks and we also had to declare AJAX endpoints for our plugin in the same way, namely as a page in the WordPress dashboard. This is how many CMS achieve extensibility in contrast to the plugins developed for desktop applications such as MS Office and integrated development environments. The overall advantage of this approach is that the entire definition of the WordPress API is already complete before the flow of execution reaches the hooking point where the developer's plugin is executed, and therefore all of the required dependencies have been correctly prepared for the plugin to work. WordPress offers many hooks, some of them before an HTML component is generated and some of them after. Unfortunately, this means that the dashboard is prepended or appended, depending on the position in the flow of our target hook, to every AJAX request started by the editor, introducing noise that has to be filtered within the HTTP responses.

There were also challenges faced while developing the reuse process detailed in Section 5. When the user performs a drag-and-drop reuse operation, we could choose between two reuse techniques planned for our editor which act as a trade-off between reliability of the reused component and performance:

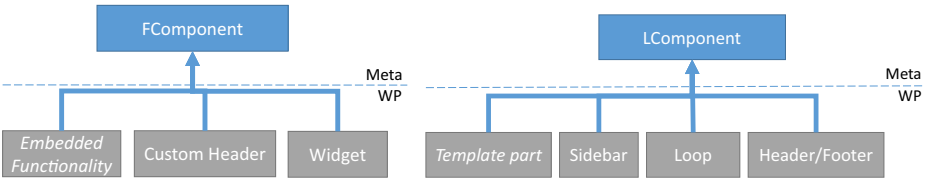
- *Clone-and-execute*: This is the approach presented in the reuse process in Section 5. It creates separate HTTP connections to retrieve each of the files. With slow source web servers, this can perform quite poorly.
- *Copy-and-filter*: CSS rules are identified and inlined to the markup. Then, we filter the output of WordPress template tags (which are PHP functions) from markup and replace it with the resulting output of the same functions in the target installation. This approach can be performed almost instantly, but it is not applicable when using components which have JavaScript or other PHP dependencies. Moreover, this approach can also cause issues when trying to infer which CSS rules are applied to the selected components.

During our implementation, we tested the tool on both fast and slow source servers and decided to opt for an approach that makes the choice of technique applied dependent on the type of component to be migrated. If the component is an LComponent, the editor filters out WordPress template tags and replaces their output from the resulting markup with a Copy-and-filter approach in the target installation. When it comes to bigger and more complex FComponents involving JavaScript and PHP dependencies, the editor applies a Clone-and-execute approach and triggers the correct execution of all the required resources for each component. The reason is that the Copy-and-filter approach heavily relies on the capabilities of matching calls to WordPress functions and replacing them with the correct output from the target installation. Unfortunately, this cannot always be guaranteed and, generally, it may be an unsafe approach: Some elements might be missed and therefore the data of the target website might not be incorporated into the new design. This makes the Clone-and-execute approach more reliable than the Copy-and-filter, but it can be much slower.

Another potential issue of the Copy-and-filter approach also has performance implications. Dragging an LComponent to the X-Themes editor requires knowledge of which CSS rules must be applied so they can be inlined. This could be implemented in two ways. The first method is to ask the browser about the computed style of each element, which is, however, not cross-browser compatible and may have serious performance issues for complex LComponents. Therefore, we chose the option of implementing our own algorithm which visits every CSS rule specified in the loaded stylesheets and only includes it if it influences the display of the corresponding elements. While this works well for most CSS rules that explicitly target DOM elements by ID or class, it can still raise performance issues in the case of many page-wide or deeply cascaded CSS rules.

7 Generalisation of Approach for Other CMS

The approach implemented for X-Themes was conceived with the goal of being general enough to be applied to other CMS. The metamodel itself avoids platform-specific concepts. This could be thought of as implying that the metamodel provides limited support for specific platforms, however we argue that our metamodel can support platform-specific concepts through taxonomy-like



(a) WordPress FComponents (b) WordPress LComponents

Fig. 7. WordPress theme concepts implemented as elements of our metamodel

extensions of LComponents and FComponents which in turn can be powered by platform-specific implementations of theme generators.

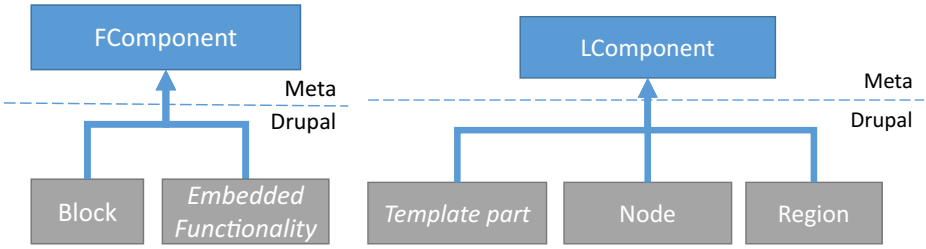
We will begin by presenting which concepts of WordPress we have been able to represent in our metamodel and then map these to similar ones in other widely used CMS, providing details of how these concepts work in each target CMS and how they can be implemented as an instance of our metamodel. According to W3Techs[11], WordPress is the most widely used CMS with over 60% of the market share. The main competitors sharing the podium are Drupal and Joomla, with 7.3% and 5.1% of the market share, respectively. We will therefore use these to explain how the approach can be generalised by analysing what the equivalent concept of a theme is in each of these CMS, and providing an insight into the differences and similarities between each of the platforms and WordPress. We will then assess how well the concepts of each platform can be handled in our proposed approach.

In all three platforms, themes consist of template files used to generate output for different parts of the theme. The level of granularity of the template hierarchy is specific to each CMS and varies a lot. However, all of these platforms provide the opportunity to code specific layout details or functionality within the templates.

WordPress provides a lot of freedom in terms of the way in which a theme can be developed. Some developers choose to embed functionality within the theme templates, while others try to create a more decoupled structure making use of more advanced concepts such as *sidebars* and *widgets*. The former are areas of the template which can be configured through the administrative dashboard to show specific *widgets* providing functionality such as showing some dynamic content or overviews of other content areas of the website. Widgets can also be registered from plugins providing functionality that will be accessed through widgets. Additionally, developers often exploit functionality provided by the WordPress platform itself such as *Customisable headers*.

As shown in Fig. 7, sidebars have been implemented in X-Themes as an LComponent, since they shape where and how widgets can be displayed in the page. Widgets, on the other hand, often introduce functionality, for example a search box, and therefore are implemented as FComponents. Moreover, we have an interest in reproducing the functionality that the widgets provide and not

[11] http://www.w3techs.com/on8April2015



(a) Drupal FComponents (b) Drupal LComponents

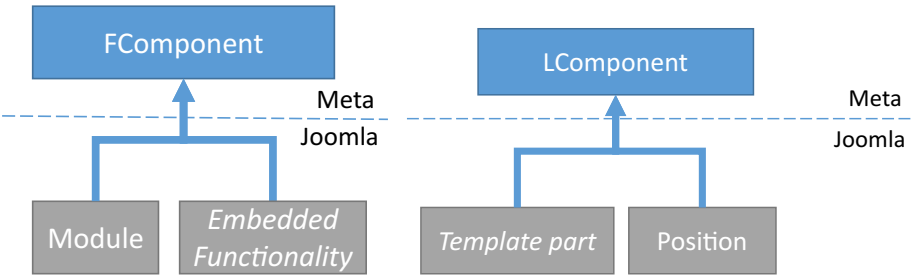
Fig. 8. Drupal theme concepts implemented as elements of our metamodel

only the way the content is shown, therefore a *clone-and-execute* approach is more suitable for widgets. *Customisable headers* allow users to specify either a single or set of header images to be shown randomly and so have also been implemented as FComponents through the re-execution of content.

Drupal provides theme developers with similar tools for dynamically configuring a theme but uses the concepts of *regions* and *blocks*. *Regions* are areas of the layout designed to host atomic *blocks* of content, which have been defined by users or through so-called *modules* which act as plugins. Differently from WordPress, regions and blocks have default templates and specific templates. However, similarly to WordPress, we can distinguish components that specify layout from those that may involve functionality. Consequently, individual regions can be implemented as LComponents and blocks as FComponents as shown in Fig. 8. Further, the sidebars of WordPress can be mapped to Drupal's regions, and widgets to blocks. Drupal uses the concept of nodes to represent individual units of data, which are generally shown iteratively, and therefore is equivalent to the Loop in WordPress. Therefore, we are able to implement Node through an LComponent.

Joomla also distinguishes between components that define layout and those that define functionality as shown in Fig. 9. In the case of Joomla, positions are declared which have special placeholder code in the theme template, and these are detected and processed by the Joomla template engine, which replaces them with the template generated for the so-called *modules*. Positions and modules can be realised as LComponents and FComponents, respectively. It is clear that Joomla's positions can be mapped to Drupal's regions. We can do the same with Joomla's modules, which can be mapped to Drupal's blocks. The automatic query of the current content being viewed, handled by the Loop in WordPress and the execution of Nodes in Drupal, is handled by a specific Joomla module, internally called *component*.

Although the three CMS use different terminology and the details of the concepts and features offered vary, they all distinguish between components that deal only with layout and those that offer functionality. In addition, they all provide some means of querying and displaying the content that is equivalent to the Loop in WordPress and also support extensibility through some kind



(a) Joomla FComponents (b) Joomla LComponents

Fig. 9. Joomla theme concepts implemented as elements of our metamodel

of plugin mechanism. As discussed, it is therefore possible to map the main concepts of all of these CMS to our metamodel, extending the LComponent and FComponent hierarchies where necessary to deal with specialisations.

8 Conclusion

We have shown how the arbitrary reuse and mixing of both layout and functionality of WordPress themes can be supported. Compared to previous work, users are able to select, reuse and combine parts of existing themes, transparently propagating the reuse to dynamic resources that define functionality on both the client and server sides. Since a theme must be based on the metamodel in order for it to be an X-Theme and accessible to the visual editor, we have defined a manual procedure for converting existing themes to an X-Theme, and are currently investigating semi-automated approaches.

There are also other research questions that we plan to address in the future. The first of these concerns data-intensive web sites which require the integration of custom post types to manage data. In previous work within our group, a tool was developed that generates a WordPress plugin with custom post types based on an entity-relationship data model defined by a developer [16]. We have now started to investigate an alternative approach that lets the user annotate sample data content from mockups or other similar websites and then automatically generates a data schema which is implemented as custom post types in the WordPress platform [17].

Acknowledgments. We acknowledge the support of the Swiss National Science Foundation who financially supported part of this research under project FZFSP0_147257.

References

1. Norrie, M.C., Nebeling, M., Di Geronimo, L., Murolo, A.: X-Themes: supporting design-by-example. In: Casteleyn, S., Rossi, G., Winckler, M. (eds.) ICWE 2014. LNCS, vol. 8541, pp. 480–489. Springer, Heidelberg (2014)
2. McCollin, R., Blakeley-Silver, T.: WordPress Theme Development. Packt Publishing (2013)

3. Casabona, J.: Building WordPress Themes from Scratch. Rockable Press (2012)
4. Norrie, M.C., Di Geronimo, L., Murolo, A., Nebeling, M.: The forgotten many? A survey of modern web development practices. In: Casteleyn, S., Rossi, G., Winckler, M. (eds.) ICWE 2014. LNCS, vol. 8541, pp. 290–307. Springer, Heidelberg (2014)
5. Hartmann, B., Wu, L., Collins, K., Klemmer, S.R.: Programming by a sample: rapidly creating web applications with d.mix. In: Proc. of the 20th ACM Symp. on User Interface Software and Technology (UIST). ACM (2007)
6. Lee, B., Srivastava, S., Kumar, R., Brafman, R., Klemmer, S.: Designing with interactive example galleries. In: Proc. of the 28th Conf. on Human Factors in Computings Systems (CHI). ACM (2010)
7. Ghiani, G., Paternò, F., Spano, L.D.: Creating mashups by direct manipulation of existing web applications. In: Piccinno, A. (ed.) IS-EUD 2011. LNCS, vol. 6654, pp. 42–52. Springer, Heidelberg (2011)
8. Gellersen, H., Wicke, R., Gaedke, M.: WebComposition: An Object-Oriented Support System for the Web Engineering Lifecycle. Computer Networks **29**(8) (1997)
9. Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Florian, D., Matera, M.: A Framework for rapid integration of presentation components. In: Proc. of the 16th Intl. Conf. on the World Wide Web (WWW). ACM (2007)
10. Leone, S., de Spindler, A., Norrie, M.C., McLeod, D.: Integrating component-based web engineering into content management systems. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 37–51. Springer, Heidelberg (2013)
11. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann (2002)
12. Houben, G., Barna, P., Frasincar, F., Vdovjak, R.: Hera: development of semantic web information systems. In: Cueva Lovelle, J.M., Rodríguez, B.M.G., Gayo, J.E.L., Ruiz, M.P.P., Aguilar, L.J. (eds.) ICWE 2003. LNCS, vol. 2722, pp. 529–538. Springer, Heidelberg (2003)
13. Knapp, A., Koch, N., Zhang, G.: Modeling the structure of web applications with ArgoUWE. In: Koch, N., Fraternali, P., Wirsing, M. (eds.) ICWE 2004. LNCS, vol. 3140, pp. 615–616. Springer, Heidelberg (2004)
14. Quinn, A.J., Bederson, B.B.: Human computation: a survey and taxonomy of a growing field. In: Proc. of the 29th Intl. Conf. on Human-Computer Interaction (CHI). ACM (2011)
15. Williams, B., Damstra, D., Stern, H.: Professional WordPress Design and Development. Wiley (2013)
16. Leone, S., de Spindler, A., Norrie, M.C.: A meta-plugin for bespoke data management in wordpress. In: Wang, X.S., Cruz, I., Delis, A., Huang, G. (eds.) WISE 2012. LNCS, vol. 7651, pp. 580–593. Springer, Heidelberg (2012)
17. Murolo, A., Norrie, M.: Deriving custom post types from digital mockups. In: Proc. of the 15th Intl. Conf. on Web Engineering (ICWE). Springer (2015)