A Core Foundation: Best Practices for Content Architecture in Sitecore

By: Jamie Stump, Delphic Digital Last Revision: October 31, 2014

Sitecore began as a .NET-based Content Management System (CMS) in 2001. At its heart, any CMS is designed to allow organizations to open up the ability to manage everyday content on their websites beyond just their IT departments. Over the past decade, Sitecore has kept its CMS system as its core offering, even as it evolved into a much more robust and refined product. Repeatedly named as a leader in the Gartner Magic Quadrant for Web Content Management, Sitecore's greatest differentiator in today's market is its integrated Digital Marketing capabilities, which are delivered via the Sitecore Digital Marketing System (DMS). Many enterprise-level organizations have or are in the process of converting their public-facing websites to Sitecore because of its integrated abilities to personalize content, perform A/B and multivariate testing upon content and create automated marketing engagement plans, among many other marketing-based features. However, to fully realize the Digital Marketing functionality that Sitecore delivers out of the box, it's critical that websites built upon Sitecore adhere to best-practice architecture. This white paper will describe some approaches toward Sitecore content architecture along with the strengths and weaknesses of each approach.

Learning the Language

Before jumping into individual architecture patterns, it is important to make sure the reader is familiar with some common Sitecore terminology and their definitions. This section will cover basic Sitecore terms and explain how they are used, in addition to comparing them to some commonly known constructs from other software, such as Microsoft SharePoint.

Sitecore Desktop

The Sitecore Desktop is the most-used Admin interface for interacting with Sitecore content, and is delivered via a web browser and specific URL. The Desktop is named as such because it is designed to look like a Windows desktop screen, including having its own "Sitecore Start Button" on a taskbar. The Desktop is only accessible after a visitor has been authenticated. Content Authors will find themselves inside the Sitecore Desktop where they are able to perform their day-to-day actions of updating content for their Sitecore-powered websites. Content Authors are able to launch all of the Sitecore-associated applications and functionality, such as the Content Editor and Page Editor, from the Desktop.

Sitecore Content Editor

The Sitecore Content Editor is a web-based, visual interface that allows for Content Authors to edit their website content. The Content Editor displays a tree view of content on the left-hand column, similar to the view you'd see in Windows Explorer. The right-hand side of the screen presents information for the currently selected object from the tree, including fields that the Content Author can complete. Like the Sitecore Desktop, the Sitecore Content Editor can only be accessed by authenticated users. The Sitecore

Content Editor can be launched from the Sitecore Desktop, or it can be logged into directly from the Sitecore admin login screen.

Sitecore Page Editor
The Sitecore Page Editor is also a web-based, browser-delivered admin interface for Content Authors to manage their content and perform digital marketing tasks. The Page Editor is presented as a "What You See is What You Get" (WYSIWYG) interface, which allows Authors to edit web pages in a manner that presents the content very closely to how it will be seen by public visitors. The Sitecore Page Editor does have some exclusive functionality that is not available to Authors within the Sitecore Content Editor, such as starting and stopping A/B or Multivariate tests. As with the Content Editor and Desktop, access to the Page Editor is restricted to authenticated users only. The Page Editor can be launched from within the Sitecore Desktop or Content Editor, or it can be logged into directly. Note that the Sitecore Page Editor is expected to be rebranded as the Sitecore Experience Editor at the end of 2014 with the release of Sitecore 8.

Items
The Sitecore Glossary defines the term Item as "an addressable unit of content made up of fields that contain information." This means that Items can be referenced by an address and are a storage mechanism for information that is segmented into fields. A Sitecore Item is the most basic level of content. In fact, all content created within Sitecore is, at its base, an Item – including (but not limited to) Pages, Digital Asset content, Templates, Presentation Detail content and DMS Goals. Sitecore stores content Items in a SQL Database.

Fields
A Sitecore Field is a single container of information. A Field contains a name, a type, and a source among other characteristics. There are many out-of-the-box options for the Field type. Some examples are Single Line Text, Treelist, Checkbox, Rich Text Editor, Image and General Link. Fields store particular pieces of information for Items and, in combination, store the collective properties of an Item. Developers unfamiliar with Sitecore can reconcile the Sitecore Field as the equivalent of a database field.

Templates
A Template in Sitecore terms is a construct for storing a particular set of information. In other words, a Sitecore Template is the combination of zero to many Fields that together represent the properties of a given object. The Template itself does not have any values for these Fields. Templates can be reused across the Sitecore implementation to allow many instances of the object they represent to be created. Templates allow Fields to be grouped into sections, though this grouping is meant only for ease of use for the Content Authors and does not have any technical significance. It is important to note that a Sitecore Template has no presentation properties / values on its own. Content Authors can inherit templates from other Templates, which allows for reusability and reduces the need to create fields more than once. Developers unfamiliar with Sitecore can reconcile the Sitecore Template as the equivalent of a database table.

Item Instances
An Item Instance in Sitecore is the usage of a Sitecore Template to represent a specific object. The Instance would have the Fields of the Template populated with the appropriate values. Item Instances may be referred to as just Instances, but there is a difference between an Item Instance and a Sitecore Instance. Developers unfamiliar with Sitecore can reconcile the Sitecore Instance as the equivalent of a populated record within a database.

Presentation Details
Presentation Details are the Items within Sitecore that point to the .NET files that will serve to display the specific controls. Presentation Details can come in a number of different varieties within Sitecore, such as Layouts, Sublayouts and Renderings.

Layouts
A Sitecore Layout points to a specific .ASPX file that will present the output HTML for this specific presentation detail. A Sitecore Layout can be considered the equivalent to a standard .NET Master Page by developers.

Sublayouts / Renderings
Sublayouts / Renderings are Presentation Details that point to .ASCX user control files, XSL Renderings, or MVC Controllers or Views. A Sitecore Sublayout / Rendering can be considered the equivalent to a web part or component by developers.

Placeholders
Placeholders are Sitecore constructs that are placed via code into Presentation Details (Layouts / Sublayouts / etc.). Sitecore Placeholders allow Sublayouts / Renderings to be placed within them. Using Placeholders allows the individual presentation details to be constructed into a cohesive web page. Placeholders are similar to web zones in Microsoft SharePoint development.

Content Tree
The Content Tree for Sitecore is the overall structure and hierarchy of all of the items that are stored within Sitecore. It is visually represented within the Sitecore Content Editor similar to the file tree that is found on the left-hand side of Windows Explorer.

Understanding the Physics
Building on the definitions of our common set of terms, we can now dive into how the Sitecore DMS works. Sitecore DMS offers analytics, content personalization, and A/B or Multivariate testing at a Component level. This means that to personalize a piece of content, there must be a related component that can be adjusted. Sitecore does not allow “field level personalization” out of the box at this time, which means individual fields within an Item Instance cannot be personalized. Many Sitecore implementations have been delivered without this in mind, rendering the organizations owning the implementation unable to utilize Sitecore’s full capabilities without a potentially large shift in the existing architecture of their sites.

Content Architecture Patterns

This section describes some commonly used patterns for Content Architecture within Sitecore, as well as recommendations when to use a particular pattern.

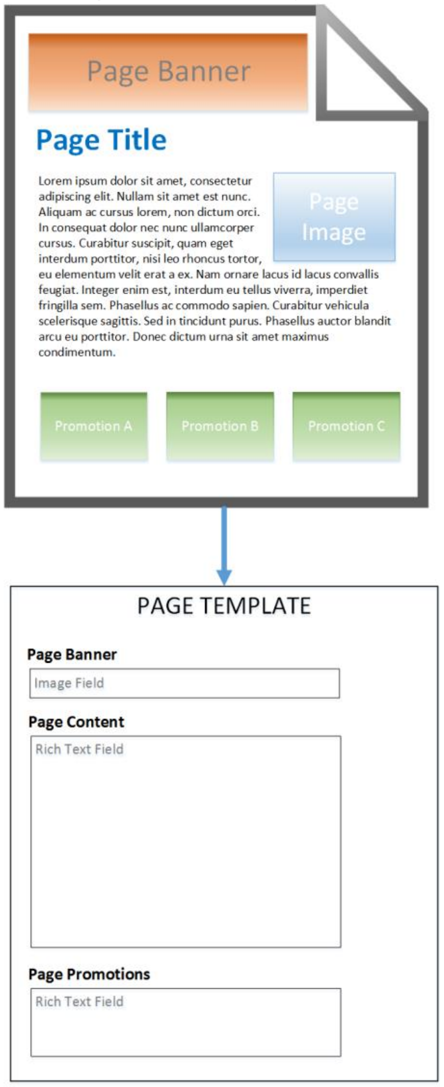
Architecture Pattern 1: Inventing the Wheel

The most basic and simple Sitecore content architecture consists of creating all necessary fields that are required to create an output web page directly into specific usage Page Templates. Any given page item instance, therefore, would have all the information it needs fully contained within its own fields. Often, this architecture pattern may result in the liberal use of the Rich Text Editor field type, which supplies Content Authors with a WYSIWYG editor, allowing for full HTML to be entered. These fields can often serve as a “blank canvas” for Authors to create any content they want or need, but at the cost of eliminating the ability to separate the content in any way. Each field that is necessary to store content is simply added to a Template. The Template is used to group all associated fields. Presentation Detail components will get all their content information directly from the Fields of the page that is being viewed on the website. This architecture is very straightforward and may be the preferred method for inexperienced developers to tackle the challenge of architecting a site. Its lone strength lies upon its simplicity, both for the developer implementing the site as well as the Content Authors responsible for filling in any content for the site. The architecture is so straightforward because all related Fields are contained within a single Template that corresponds to a single page on the website.

Sites built with this architecture pattern may be very quick to launch, but will soon suffer a host of problems. This architecture does not allow for reusability within the content – each Field or set of Fields is specific to its page, and any content that appears in multiple pages must be defined by Content Authors on each page individually. Beyond having the issue of reusability, websites architected in this pattern would also not allow for any out-of-the-box DMS functionality to be implemented without wide-spread changes to the architecture being made first. These changes can often be very costly in both time and money and also may result in content rewrites or content migration into the new architecture.

The disadvantages of this architecture outweigh its single strength by a factor that is so large that it makes this pattern impossible to recommend. Experienced Sitecore partners should never implement a site in this manner.

Inventing the Wheel Example Visualization



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam sit amet est nunc. Aliquam ac cursus lorem, non dictum orci. In consequat dolor nec nunc ullamcorper cursus. Curabitur suscipit, quam eget interdum porttitor, nisi leo rhoncus tortor, eu elementum velit erat a ex. Nam ornare lacus id lacus convallis feugiat. Integer enim est, interdum eu tellus viverra, imperdiet fringilla sem. Phasellus ac commodo sapien. Curabitur vehicula scelerisque sagittis. Sed in tincidunt purus. Phasellus auctor blandit arcu eu porttitor. Donec dictum urna sit amet maximus condimentum.

Architecture Pattern 2: Steam Powered
Going beyond the most basic architecture pattern described above involves separating Fields from pages into a concept such as a Data Template. Instead of containing all related Fields within a single Item Instance, the Fields are now separated into smaller logical units. In addition to splitting fields amongst templates, this pattern will often see a reduction in the amount of Rich Text Editor Fields used, instead breaking content into separate, more defined Fields that have less flexibility but more context. Once Fields are split amongst many Instances, the architect must also supply a method to relate the Instances that are logically associated to each other. In this architecture pattern, the method of relation

Building a Roadway System

Site Navigation within Sitecore is a topic that can be divisive amongst Sitecore architects. The content for menus, such as main menus, sidebar navigation structures, and footer menus is owned by the Marketing departments of organizations running on Sitecore but primarily maintained by IT resources. Because of this, the content such as link text, order of links in a menu, and the target URL of links should be managed within Sitecore itself, rather than being hardcoded by a developer. The hierarchical nature of Sitecore content allows for menus to be automatically generated directly off of page items. This prevents Content Authors from having to create any additional items for menus, thus eliminating some “extra” work. However, in order to meet the requirements of different menus, relying solely upon the page structure will most likely require a number of fields being added to the page Templates. For example, the ability to hide (or conversely, explicitly show) a page from a navigation structure is critical. The more navigation structures that a website has, the more fields need to be added to the page, and the more likely it may be that a field is accidentally overlooked by a Content Author. Moreover, menus that do not adhere exactly to the URL structure of the pages become a challenge with this navigation architecture. For example, if a page requires a URL that is only one level removed from the home page of a site, but the link to this page on the main menu needs to be grouped under a heading (or show up in multiple locations on the same menu structure), using the hierarchical nature of pages becomes challenging. Some Sitecore implementation partners circumvent this would be to create Fields that allow the selection of one or more related objects within the Page Item itself. Presentation Detail components now have to determine which Item they are to pull their content from based upon the Fields within the Page Item that have selected other Items. The “Steam Powered” architecture pattern eliminates the issue of reusability that the first pattern faced; by separating the Fields into smaller Instances, Content Authors are now able to relate the Items on a one-to-many basis. The separation of content can take a number of different forms. Some developers will place the separated content “beneath” the actual page Items themselves. Creating children of the separate content allows for a natural association between the Page Item itself and content that is separate but connected to the page. The downside is that, while there is not a restriction on reusing the content across pages, the logical representation can make reuse harder to grasp. Having content within the Page Items hierarchy also exposes that content to an accessible URL, even though it is not a “page” itself – this may be undesired. Other partners choose to separate all non-page content into a data repository or library section of the Content Tree, completely outside the page hierarchy. This separation allows for easy reuse and does not result in physical URLs being accessible for non-page content, but reduces the logical connection between page and data. An additional approach for storing this content is a hybrid – data that is page specific can be created as a child of a Page Item, and any data that can be on many different pages can be put in a centralized repository. This architecture is still very straight forward from a development perspective and is often the approach that is first chosen by experienced developers that are new to Sitecore.

While the reusability issue is solved in this architecture pattern, the implementation of some DMS functionality will still pose a problem. Because the Instances of Items are related via Fields, it will not be possible to use the out-of-the-box functionality to personalize or test any components that pull their content in this manner. Organizations with this architecture will still be able to get some usage out of the analytics data from Sitecore DMS such as goal tracking and campaign management.

challenge with the use of Link or Redirect pages. These are Page Items within the regular site hierarchy that are designed to simply allow an option on a menu to show while performing a 302 redirect to the “proper” URL anytime a request to the Link page is encountered.

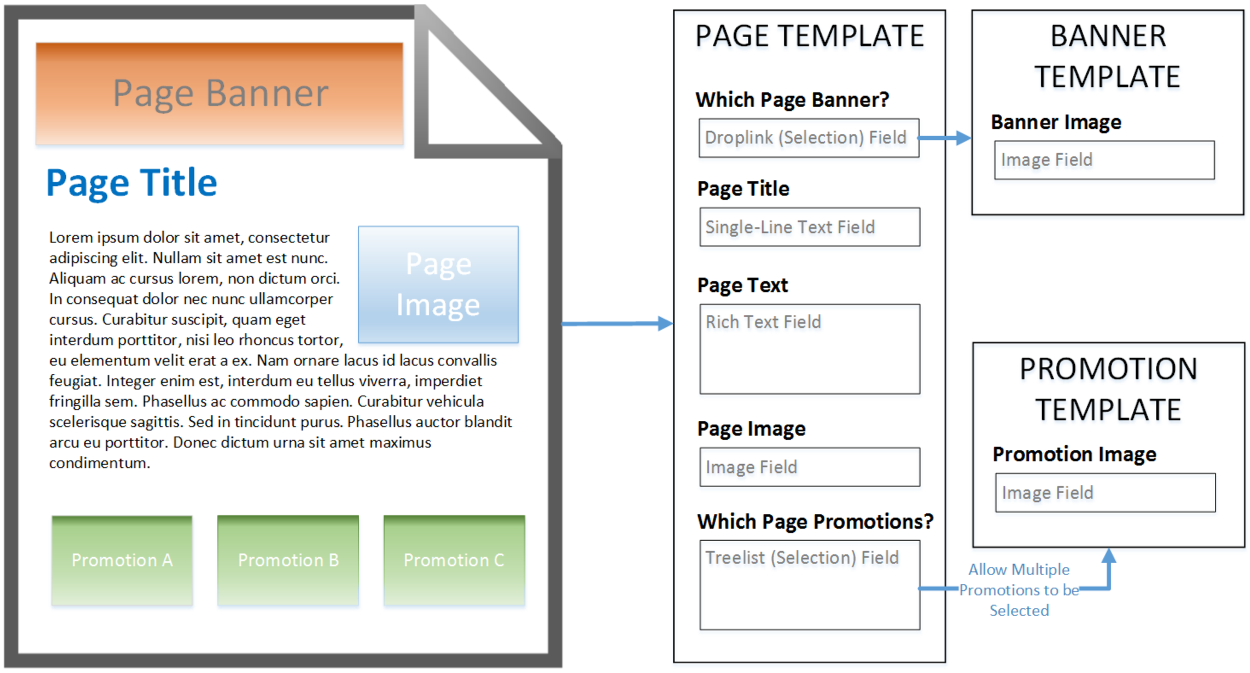
A different approach toward navigation structures within Sitecore is to split them completely from the page structure. This separation allows for each menu to be managed by Content Authors on its own, independent of the hierarchy of pages. The argument against this separation is that it requires the Content Authors extra effort to manage their navigation structures in addition to their pages. The advantages to this approach are providing a clear-cut view from the Content Editor as to the content of a menu and the ability to easily separate menu structure from page URL structure without requiring any cluttering with Link pages. Another advantage of this method is that it reduces the number of repetitive fields that must be included within Page Items, reducing the risk of Fields being overlooked by accident. Finally, separating the menu content from the pages allows for menus to be subject to Sitecore DMS functionality, although the value of applying DMS to navigation structures is debatable.

Ultimately, the architecture selected for navigation structures must fit the organization that owns the Sitecore implementation. Neither automatically building menus off the pages nor separating the content from the pages are the “right” or “wrong” ways to do things, as both can result in meeting navigational requirements. The important factor is that the client organization should be informed of the

The “Steam Powered” architectural pattern should only be considered by organizations that are 100% confident that they do not and will never want or need to implement any Sitecore DMS-related functionality. As with the “Inventing the Wheel” pattern, the cost of re-architecting a site implemented with this pattern to enable the use of DMS functionality can be significant, both in monetary outlay and timeliness.

decisions toward navigation that are being made by the implementation partner throughout the architecting process. As long as the client agrees that their organization will be able to understand and utilize the architecture being defined, then the difference in patterns is largely subjective.

Steam Powered Example Visualization


Page Banner
Page Title
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam sit amet est nunc. Aliquam ac cursus lorem, non dictum orci. In consequat dolor nec nunc ullamcorper cursus. Curabitur suscipit, quam eget interdum porttitor, nisi leo rhoncus tortor, eu elementum velit erat a ex. Nam ornare lacus id lacus convallis feugiat. Integer enim est, interdum eu tellus viverra, imperdiet fringilla sem. Phasellus ac commodo sapien. Curabitur vehicula scelerisque sagittis. Sed in tincidunt purus. Phasellus auctor blandit arcu eu porttitor. Donec dictum urna sit amet maximus condimentum.
Page Image
Promotion A
Promotion B
Promotion C
PAGE TEMPLATE
Which Page Banner?
Droplink (Selection) Field
Page Title
Single-Line Text Field
Page Text
Rich Text Field
Page Image
Image Field
Which Page Promotions?
Treelist (Selection) Field
BANNER TEMPLATE
Banner Image
Image Field
PROMOTION TEMPLATE
Promotion Image
Image Field
Allow Multiple Promotions to be Selected

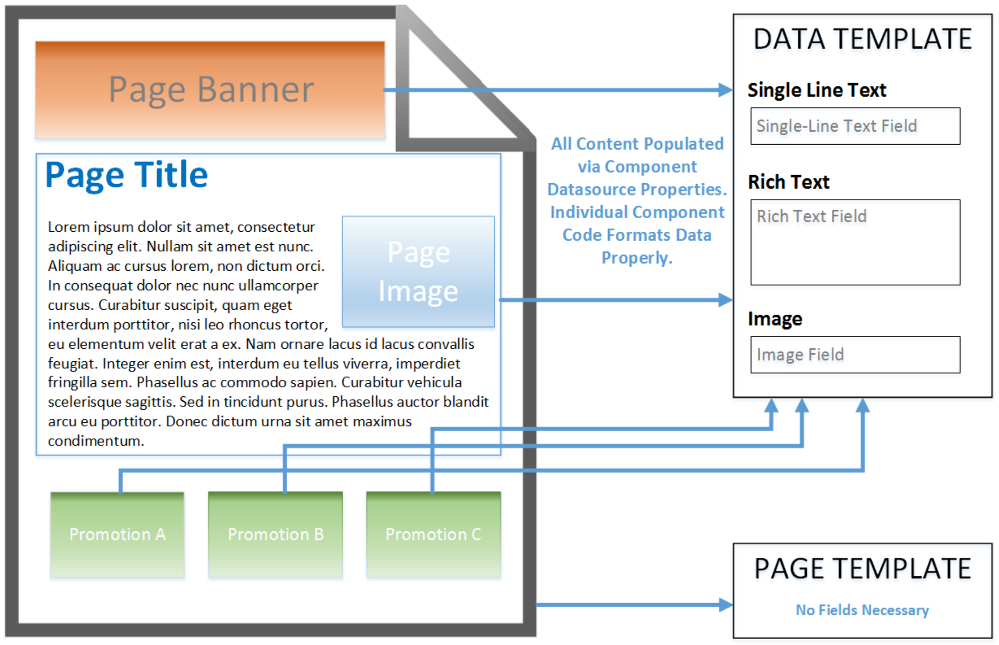
Architecture Pattern 3: Anti-Matter

While the first two architectural patterns suffer from an inability to allow DMS functionality, this third pattern is designed largely with the DMS in mind. The defining factor of this architecture pattern is the reduction of Templates, particularly Page Templates, to a stark minimum, sometimes having only a single Page Template for an entire Sitecore implementation. In fact, many partners who utilize this architectural pattern are vocal in stating their disdain for Page Templates and suggest to avoid Page Templates as much as possible. To achieve this reduction in Templates, the Templates are purposefully made generic so that they can fit many situations. Similar to the last pattern, the “Anti-Matter” architectural pattern sees the vast majority of content is split into separate data items. The difference lies in the association that is created between the separate content and the Page Items themselves. This architectural pattern associates the content to the page via the datasource property of the components that comprise a particular page. Because of utilizing only a single (or few) Template(s) for page items, Content Authors must utilize the Sitecore Page Editor interface for constructing the components that appear on each of their individual pages. The developer of an implementation following this pattern will utilize placeholders on their Sitecore Layouts (and possibly Sublayouts) to allow certain governance that can be followed by the Content Authors as they lay out the components on their page.

The major advantage of the “Anti-Matter” pattern is the aforementioned ability to fully utilize Sitecore’s DMS functionality. However, this pattern is also advantageous for organizations that plan on having their Content Authors primarily utilize the Page Editor interface. Moreover, this pattern can make implementing site search (using indexing rather than web crawling) against Sitecore content easier, because of the limited number of Templates that are utilized. While the complexity of this architecture yields many advantages, it does post somewhat of a challenge to business users. The nature of this pattern requires the content be very abstract and separate in nature. Organizations that have Content Authors and Marketing Departments that are not able to understand the abstractness of their data may struggle to maintain their sites and content. This can especially be true in the Sitecore Content Editor, where it is possible for the Content Authors to get lost or confused, and struggle with finding content when it is this abstract. Some implementing partners actually prevent this possible confusion by locking their clients out of the Content Editor and forcing them to only use the Page Editor. However, doing this is simply reducing the functionality of the Sitecore platform and may not be seen as an ideal situation for many clients. While the long-term complexity of this pattern is exposed to the business users of Sitecore, the “Anti-Matter” architecture pattern also has a higher learning curve for implementers.

Because of the abstractness and complexity, this architectural pattern should be utilized only by organizations with technically mature business users and marketers. Organizations that are already planning on using the Page Editor and who have full DMS requirements will get the most benefit from this pattern.

Anti-Matter Example Visualization

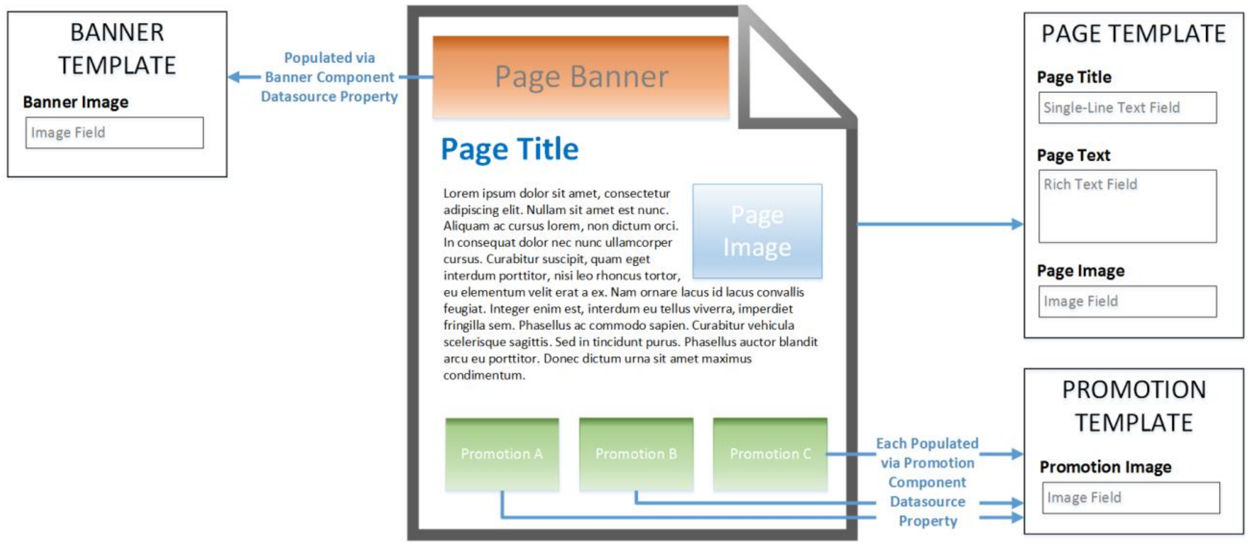

Page Banner
Page Title
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam sit amet est nunc. Aliquam ac cursus lorem, non dictum orci. In consequat dolor nec nunc ullamcorper cursus. Curabitur suscipit, quam eget interdum porttitor, nisi leo rhoncus tortor, eu elementum velit erat a ex. Nam ornare lacus id lacus convallis feugiat. Integer enim est, interdum eu tellus viverra, imperdiet fringilla sem. Phasellus ac commodo sapien. Curabitur vehicula scelerisque sagittis. Sed in tincidunt purus. Phasellus auctor blandit arcu eu porttitor. Donec dictum urna sit amet maximus condimentum.
Page Image
Promotion A
Promotion B
Promotion C
All Content Populated via Component Datasource Properties. Individual Component Code Formats Data Properly.
DATA TEMPLATE
Single Line Text
Single-Line Text Field
Rich Text
Rich Text Field
Image
Image Field
PAGE TEMPLATE
No Fields Necessary

Architecture Pattern 4: Clean Energy

The final architectural pattern lies somewhere between the last two patterns described. The "Clean Energy" architectural pattern relies upon multiple Page Templates in order to allow for organization-specific business context. It also breaks out content into contextual Data Templates that allow for reusability. Like the last pattern, this pattern associates the Data Items with Page Items via the datasource field of the components that comprise a page. Finally, depending upon the implementer and the client Author structure, this pattern may see some duplication of Fields between Page and Data Templates. The design of this duplication is to allow for more straight-forward content authoring to be utilized whenever DMS functionality is not applicable to the specific content, while not inhibiting DMS functionality for the future or for other content of the same type. Organizations that have everyday Content Authors handling content as well as dedicated Digital Marketers who use Sitecore for its personalization capabilities may see benefit from this duplication. The Content Authors have some straight-forward fields within a context they understand, while the Digital Marketers have the flexibility to create separate content that can override the base Field values via a component's datasource.

By blending the contextually specific Templates with the datasource association method, this pattern results in the ability to have full DMS functionality, but also reduces the amount of complexity faced by the everyday business users of Sitecore. This pattern also allows Content Authors the freedom of utilizing both the Sitecore Content Editor and Page Editor for their authoring. The downside of this pattern is that it will often result in longer development cycles than the previous patterns, both for the upfront site implementation development and ongoing enhancement development. This pattern can also face more challenges in implementing search indexing than the previous pattern does. Organizations that wish to take advantage of Sitecore DMS and the Sitecore Page Editor but that also want the content to be contextual for business users should strongly consider this architectural pattern.

Clean Energy Example Visualization


BANNER TEMPLATE
Banner Image
Image Field
Populated via Banner Component Datasource Property
Page Banner
Page Title
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam sit amet est nunc. Aliquam ac cursus lorem, non dictum orci. In consequat dolor nec nunc ullamcorper cursus. Curabitur suscipit, quam eget interdum porttitor, nisi leo rhoncus tortor, eu elementum velit erat a ex. Nam ornare lacus id lacus convallis feugiat. Integer enim est, interdum eu tellus viverra, imperdiet fringilla sem. Phasellus ac commodo sapien. Curabitur vehicula scelerisque sagittis. Sed in tincidunt purus. Phasellus auctor blandit arcu eu porttitor. Donec dictum urna sit amet maximus condimentum.
Page Image
Promotion A
Promotion B
Promotion C
PAGE TEMPLATE
Page Title
Single-Line Text Field
Page Text
Rich Text Field
Page Image
Image Field
Each Populated via Promotion Component Datasource Property
PROMOTION TEMPLATE
Promotion Image
Image Field

A Look to the Future

Sitecore 8.0, which is scheduled for release toward the end of 2014, will present a new feature to Sitecore called the Federated Experience Manager (FXM). The FXM will allow organizations to utilize Sitecore DMS functionality on websites they own that are not on the Sitecore platform. This will be done through the placement of a special Sitecore generated JavaScript tag on the code of these "external" websites. While this feature is very exciting for that intended purpose, it also provides promising potential to organizations whose sites are already architected upon Sitecore. Many existing Sitecore sites have been architected without DMS functionality in mind, and the cost to retro-fit these sites to allow for DMS can be quite high. In theory, the FXM will allow the placement of that same special JavaScript tag to be put onto a Sitecore powered site, regardless of its content architecture. Suddenly, it will become much less costly and take much less time for existing sites to have full DMS functionality even if they were not architected with that functionality in mind!

Conclusion

The goal of this white paper was to help educate both clients and implementers in a few different architectural patterns toward Sitecore content architecture. The pattern that fits each client will be specific to the client themselves as well as their business and functional requirements. Armed with the information found above, partners and clients should be able to better choose a pattern that will work best for them and have confidence that their Sitecore implementation will meet their organization's needs.

About the Author

Jamie Stump is a Sitecore Architect at Delphic Digital and was honored to be named as a Sitecore MVP for 2013 and 2014. Jamie has been in the technical consulting realm for 7+ years. Jamie specializes in Sitecore architecture and development and his broad experience includes Sitecore installation, configuration, and CEP development, including custom DMS implementations. He has implemented Sitecore solutions for a number of industry verticals including manufacturing, healthcare, financial services, advertising and retail and energy. In addition to architecting and implementing Sitecore sites and eCommerce solutions, Jamie also works with other Microsoft technologies including the .NET platform and SQL Server. Jamie's can be kept up with on Twitter at @jstump29 and tweets links for all his Sitecore blog posts as they go live from that handle as well. He has a B.S. in Information Systems Development from York College of PA. Originally from Philadelphia, PA, Jamie now resides in San Diego, CA with his wife and son.