Drupal 8 Page Building

A Strategy for an Outside-In User Interface System

A Drupal Community Project Sponsored by Myplanet Digital
By Michael Keara (User Advocate)
May 13, 2012

Table of Contents

This document will outline high-level user interface system architecture for page building and will identify a series of key UX design considerations that will need to be resolved by the UX design community.

The hope is that having a viable UI systems view in place and agreed upon, the discussions across and within teams can proceed more efficiently and the tasks of solving the individual UI design challenges can proceed more smoothly.

Executive Summary

Aims of this Document

The transition from Drupal 7 to Drupal 8 will represent a significant shift in both the internal architecture and the user experience design for the process of page building and content management.

One of the most powerful concepts in the redesign is that of *context*. The aim is to harness the power of contextual information to allow for intuitive and efficient page building and content management for both professional and non-professional site builders as well as content managers and authors.

Accomplishing this will put Drupal at the forefront of web development platforms and will allow it to go head to head with commercial solutions.

This is not a trivial task. In order to make this transition successful, it will be extremely important for developers and UX designers to achieve a clear understanding of each other's goals, priorities and concerns.

This document is an attempt to aid that process of understanding and will present an analysis of the page building process from an 'outside-in' perspective. That is to say, it will examine the technical requirements and foundations with an eye to how it affects the user experience. At the same time it will factor in certain UX requirements and concerns in order to arrive at some points for consideration in the technical strategy.

The mock ups presented here are not intended to be finished UI designs but to merely indicate where designs need to be done within a reasonably complete UI system and to outline the kinds of tasks these areas need to support.

Basis for Analysis

My analysis and resulting recommendations for a page building user interface system in Drupal 8 are based on a number of factors:

- Extensive designing and building web sites in Drupal
- Detailed examination of the Panels Page Manager system from both the UI and technical perspectives
- Careful study of technical documentation
- Interactions with the Drupal community (both development and UX teams)
- Over two decades of experience design and building user interface systems

Some of the ideas emerging from the analysis work are no doubt disruptive as they question some long held beliefs about how things are done in Drupal. In this document I have made a concerted effort to explain the rationale for these recommendations.

I have also tested these ideas within a variety of scenarios and through discussion with experienced Drupal developers as well as through my own development work on client web sites. So far I have not encountered any major obstacles other than the unfamiliarity of the ideas.

Recommendations

The recommendations listed below outline the broad brushstrokes of a user interface system that can, I believe, bring an effective interaction strategy to the new WSCCI based architecture for Drupal. Because they explore the middle ground between UI and technology some of them pertain to interaction techniques and some are suggestions for technical consideration.

Here are my main recommendations:

1. Use an 'outside-in' user interface system that allows users to work with fewer abstractions
2. Start with getting the workflow to make sense. Don't begin with a UI strategy that is based on a drag and drop technique. It may or may not be a viable approach.
3. Follow the principle of 'definition-usage pairing' to allow a natural workflow for contextual configuration
4. Support an Information Architecture Space approach for page definition and management
5. Use the IA Space definitions to enable a complete transformation to a Pull model
6. Allow 'empty' pages to be defined by the Page Manager and provide a default mechanism that directs the site builder to fill these gaps
7. Allow *implicit* contexts only for dependent (child) components and avoid them for the top most parent components
8. Move away from 'node/%node' convention for *front end display pages* because it is based on deriving implicit contexts at the parent level

What Are We Trying to Fix?

Certain key problems with Drupal's site building process have been identified from both the developer side and the UX side. Here are some quotes:

UX Perspective

"Drupal's site building user experience currently consists of moving between a lot of different tools and interfaces. Within Drupal 8 we want to unify these experiences (blocks, menus, views, panels…) more."

"Drupal forces users to build an understanding of its own internal implementation model."
http://groups.drupal.org/node/160144

Developer Perspective

"Blocks as they currently exist have out-lived their usefulness as a basic Drupal component. Placement of blocks is a common shortcoming of the core system, as is the use of multiple instances of a single block, which is impossible without contributed modules."

"Instead of building pages via a page callback as "primary" content, and then blocks and other bits of "secondary" decorating the page, the goal is to make any output on the page anywhere a block. Not a "block" as traditionally thought of in the limited Block module in Drupal 7 and below, but a "smart block" that is context-aware and utilizes per-instance block configuration."
http://krisandju.e-webindustries.com/blog/drupal-8-blocks-layouts-everywhere

The 'Blocks and Layout Everywhere Initiative' summarizes its goals as:
"This initiative aims to bring unity to a system of disjointed output components (blocks, page callbacks, menus, theme settings, and more) and provide a standardized mechanism of output, new tools for placing content on a page, and a potential for performance gains amongst other benefits."
http://groups.drupal.org/scotch

In general the aim is to refactor the internal architecture of Drupal to achieve a cleaner and possibly better performing system while at the same time, design a user interface system that takes advantage of this new architecture and make the site building UX significantly easier.

Interpreting User Feedback

User 1 on Task 3: Create an article about your subject, tag this content with relevant keywords. (introduction to taxonomy and tagging)

Quote: *"There's no tab called 'Our Songs' but there is a page named 'Our Songs' and it's not in this left nav which I expected it to show up in."*

Interpretation: The user is trying to build the site from the visible elements. He seems unaware that the menu on the side is a developer menu. He seems to assume it would be the menu that an end user would see.

Quote: *"I assume this 'add content' link means add a link to the navigation and not the content itself."*

Interpretation: He speculates that the side menu link 'Add content' will take you to an editor for the menu itself. He appears to be thinking about building a menu directly on screen in the same way that one might draw it in a design sketch.

Why an Outside-In Approach?

Two Ways to Look at a Web Site

Technical users, who have overcome much of Drupal's notoriously steep learning curve, must carry out various complex procedures for building web sites. Much of the esoteric knowledge that such site builders have about Drupal is a familiarity with the plethora of seemingly unrelated administrative areas required to configure the various components that provide content to web pages. This is a very 'inside-out' view of the web site.

Non-technical users have difficulty with such abstractions. They tend to look at things very literally. The user tests recently conducted at Google provided evidence to support this hypothesis and some interpretations of those tests appear in the left column.

When tasked with building a web site, non-technical users seem inclined to 'mould' the web site into existence by modifying the visible aspects of it that they find on screen. In contrast to Drupal's current administrative techniques that requires knowledge of abstractions such as nodes, views, modules, etc., the tendency is to try to directly manipulate the tangible elements as they are found (or looked for) on screen as if it were some sort of putty.

Site owners also seem to look at their web site from the outside. They typically have some clear, high-level sense of what the site should be doing. At this level, all web sites represent somebody's *business* whether it be a profit, non-profit or recreational undertaking. The 'business' of a web site is its *intent* – it's reason for being. I have yet to meet a client who cannot say something meaningful about what they are trying to achieve with their existing or planned web site. They often describe this from the point of view of a site visitor – from the outside.

There seems to be a substantial difference between the 'natural' way of looking at a web site from the outside and the complex, technical perspective that site builders must view it through. This can contribute to problems in the site production process.

The Importance of Information Architecture

The 'Push Model' – An Inside-Out Approach

I've had many conversations with site owners about what they want to do with their web sites in high level terms. They describe the various pages where they see certain aspects of their business goals being carried out: the home page; the about us page; the blog; the contact us page. These kinds of conversations are essentially about the information architecture behind the user experience.

The standard Drupal way of getting content into an Information Architecture is to create the content first and then 'push' it out into some menu structure. This presents a problem. The IA of a site can get established quite early in the process, based on those discussions with the site owner. But content is often the last piece to come into being. It is far easier to talk about an About Us page at a high level than to actually write the content. This, along with keeping track of evolving content nodes, can become a serious bottleneck in the process.

So even though the IA of a site is critical to the business objectives and is a fairly easy way for the site owner to understand the web site as a business tool, the design and implementation of an explicit Information Architecture is not easily supported in Drupal.

Implementing an Information Architecture as a by-product of content creation leads to a 'Push' model of site building. In my opinion, the Push model makes the learning curve significantly steeper and is likely to lead to non-technical user confusion as they try to *guess* how the resultant pages will look.

Arguably, a more natural way to build pages would be based on an outside-in 'Pull' model. One significant attempt to realize this 'Pull' model is the Panels suite of modules.

There are a number of ways that content is 'pushed' into an IA including:

- Associating a node with a menu from the node edit form
- Promoting a node to the front page
- Creating a page display in views

This decentralized method of building up an IA can be difficult to maintain and can potentially lead to unforeseen results as differing conditions determine the presence of page elements.

Panels is Based on a 'Pull' Model

The Panels and Page Manager modules go a long way to facilitating an 'outside-in' approach to site building. Although the user interface is far from perfect, the underlying architecture allows site builders to think in terms of building pages and assigning certain static or dynamic content to them.

This essentially reverses the relationship between an IA and the content nodes. Certainly, content can be created at any time in the overall process, but in the end, the content is *pulled* into specific pages that are located within a given IA structure.

Simply put, a site building process that is based on a Pull model will allow us to create a more natural feeling user experience because it can potentially remove many of the abstractions that currently hamper Drupal's UX.

The new D8 architecture for handling page management, content retrieval and display is modelled after the Panels approach. It has been described as 'envisioning Panels in core'.

But the consensus seems to be that the current Panels UI is too complex, so what remains to be done is to also envision an improved user interface for this Panels-like architecture.

There are some key user interface systems concepts that can be useful in this regard. The following section describes what they are.

User interfaces that contain all possible ‘tools’ that a user has permission to use, can be overwhelming and complex. The concept of Active Roles allows the UI to stay stripped down to the essential components.

Key User Interface Systems Concepts

Active Roles

Carrying out a given Role requires a set of tools or ‘screenware’ elements to get the task done. Logically, it makes sense that these elements must be present on screen when the user wishes to carry out the Role.

In Drupal we use the term ‘roles’ to describe what a user is *allowed* to do via permission sets. But permission sets by themselves do not tell us *when* a user chooses to carry out a given role.

The concept of Active Roles refers to a mechanism for indicating *when* a user decides to carry out a given Role. This information can be useful in UI design because it can trigger the presentation of the necessary ‘tools’ required for the Role related tasks.

In contrast to this, a UI system that is based just on permissions can easily overload the screen with *all possible* tools simultaneously.

IA Space

The Panels Page Manager system allows a site builder to create a ‘custom page’ and then assign one or more mechanisms to ‘pull’ content into that location. The simple, but important, conceptual question behind the ‘pull’ model is:

“What are we pulling content into?”

The simple answer is: some sort of space. More specifically, I would argue that this is an Information Architecture Space, or ‘IA Space’.

Logically, this means that Page Manager, is creating an explicit IA Space, based on URLs, and then assigning one or more ‘recipes’ for page assembly within each of those locations.

In Drupal currently, Page Manager tends to down-play the role of its pages by describing them as ‘custom pages’. In a full outside-in, Pull model system, the entire front end Information Architecture could be defined with this method.

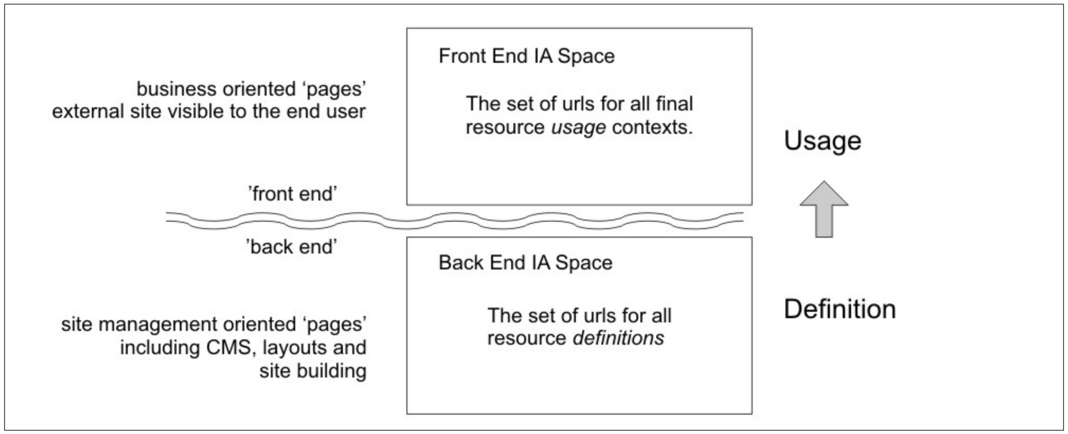
Definition-Usage Pairing

According to Kris Vanderwater, the Layout Manager will assemble pages in the following way:

"(url + context) determines the layout, which defines the components, which assembles into a page."

http://groups.drupal.org/node/218914#comment-720574

From the UX point of view, this implies that the Layout Manager is a mechanism for assembling components. This in turn implies that the content sources within components *already exist*, probably having been defined elsewhere – e.g. nodes, user objects, Views, etc.


business oriented 'pages'
external site visible to the end user
Front End IA Space
The set of urls for all final resource *usage* contexts.
'front end'
'back end'
Back End IA Space
The set of urls for all resource *definitions*
site management oriented 'pages' including CMS, layouts and site building
Usage
Definition

Behind this functionality there is the notion that things are *defined* (e.g. as 'components') in one context and then *used* in a different context. This definition-usage pairing is a principle that can be applied in many types of UI systems.

The definition-usage pairing idea fits nicely into the concept of IA Spaces. Web sites, as we know, consist of collections of public and non-public URLs. With that in mind, the diagram at the left shows the very broad IA Space division that distinguishes the public facing web site from its administrative areas. It can be thought of as simply grouping two kinds of URLs.

So we can think of any page that is used for administrative purposes (in the very broad sense of the word) as being essentially some sort of 'definition' of a resource that is ultimately 'used' when presented via the publicly accessible pages. The concept of a

To be consistent with the 'resource' terminology, we could even think of nodes as 'content resources' that are 'defined' via the node creation/edit page.

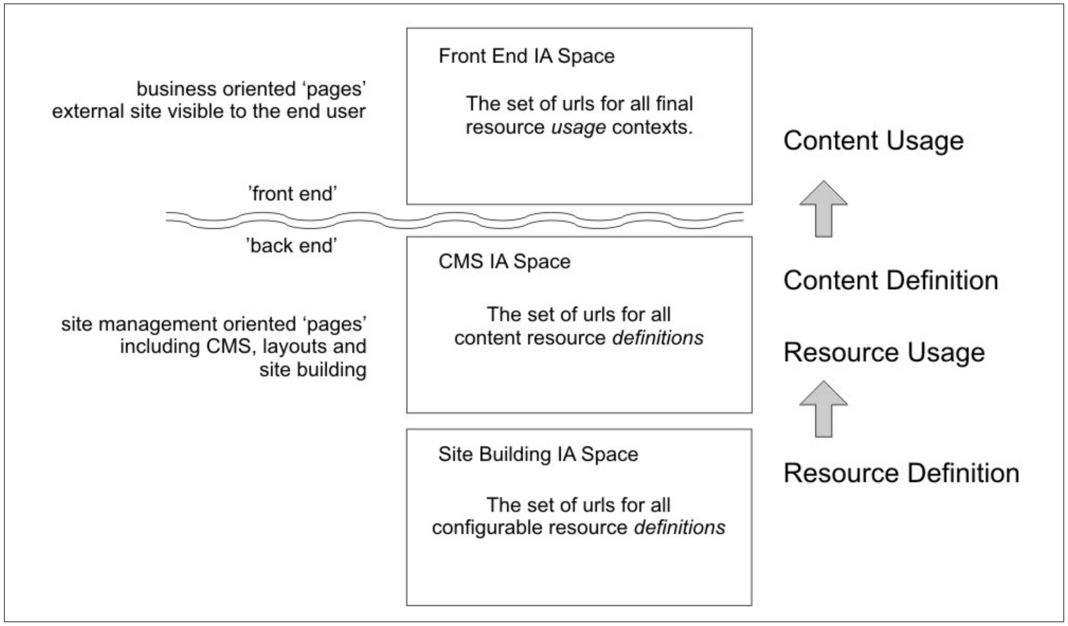
'resource' is useful when thinking about the general workflow of things being first defined and then used. For example:

- Resource definitions - where the stuff 'lives' as a data source. This resolves to a single address (URI).
- Resource usage - where the resource data is presented to a user. This could be multiple usage contexts and is dependent upon Role (site builder usage is different from end user usage)

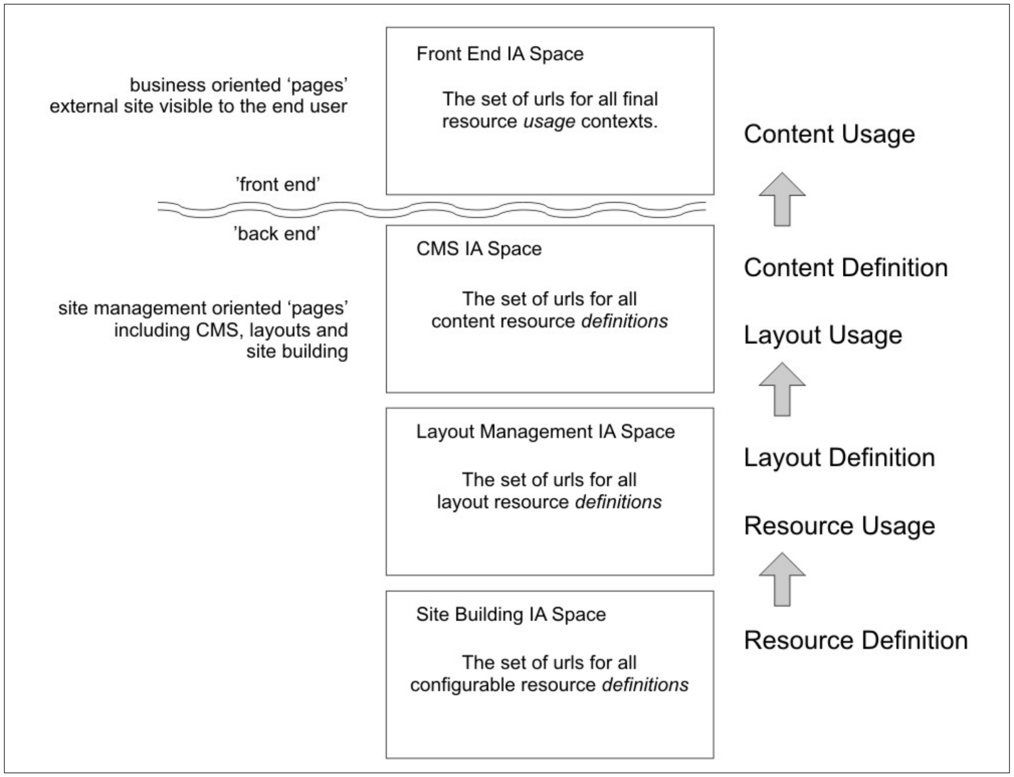
So now these questions emerge:

- Which user *defines* a resource?
- Which user *uses* a resource?

And this takes us into the area of Role divisions. Let's look at the same diagram with the 'back end' IA Space being further subdivided into pages that are used by Content Managers and pages that are used by Site Builders. Notice how the definition-usage pairs begin to chain.


business oriented 'pages'
external site visible to the end user
Front End IA Space
The set of urls for all final resource *usage* contexts.
Content Usage
'front end'
'back end'
CMS IA Space
The set of urls for all content resource *definitions*
Content Definition
site management oriented 'pages' including CMS, layouts and site building
Resource Usage
Site Building IA Space
The set of urls for all configurable resource *definitions*
Resource Definition

So we can look at the Site Builder's Role as being oriented around providing the various resources required by a Content Manager. For example they define content types, Views, input filters, Layouts and so on. Many of the 'resources' that the Site Builder defines are available for the Content Manager to use or build upon.


business oriented 'pages'
external site visible to the end user
Front End IA Space
The set of urls for all final resource *usage* contexts.
Content Usage
'front end'
'back end'
CMS IA Space
The set of urls for all content resource *definitions*
Content Definition
site management oriented 'pages' including CMS, layouts and site building
Layout Usage
Layout Management IA Space
The set of urls for all layout resource *definitions*
Layout Definition
Resource Usage
Site Building IA Space
The set of urls for all configurable resource *definitions*
Resource Definition

There is one more intriguing Role that seems to arise in real life scenarios: that of the Layout Manager. This Role is interesting because it can be looked at from two perspectives: the Site Builder and the Content Manager.

From the Site Builder point of view, the task of defining Layouts might be seen as a *strategic* step. In other words the Site Builder might want to identify and build all the possible Layouts that can be used for assembling content.

But I've often heard of scenarios where the Content Manager/Editor wants to build a *tactical* Layout to handle a special case in order to amplify the impact of some topical content. (The examples I've heard about came from news oriented and grocery store web sites.)

Given that, we could theorize that there is another intermediate Role, the *Layout Manager*, who could be responsible for the creation and management of Layouts. The definition-usage chaining is still in effect.

An interesting implication of this 'outside-in approach' is that it is also possible to create an entire front end IA Space simply by defining locations as the set of business-semantic URLs. In other words, there is no conceptual reason why a node has to exist before the IA Space location is defined.

These 'placeholder' locations can then be filled in with the proper recipes for content assembly as part of the site building process. This is an outside-in approach to web site creation that uses a 'pull' model.

URL Semantics

With these very different contexts for task sets in mind, let's reflect back on the semantics of the URLs themselves. Starting from the front end, the URLs exposed to the public should ideally have some sort of hint about the meaning of the pages. The meaning of the site's front end can be probably expressed by the site owner (client) in terms of the *intent* behind each of the front end pages.

Theoretically, it should be possible for a Site Builder to have a conversation with the Client (directly or via a Product Owner) about the purpose of the site and the intent for each individual page. Ideally the site builder might be able to create a set of URLs that reflect this in some semantic form. Doing this is a process of *explicitly* creating an IA Space and it is an essential step in site building from the outside-in and using a Pull model for content retrieval.

However there is a long standing practice in Drupal that works against this capability. When used for content *display*, the conventional 'node/123' path structure works *against* a pull model because it doesn't allow for articulation of the precise IA Space location (page) that components are being pulled into.

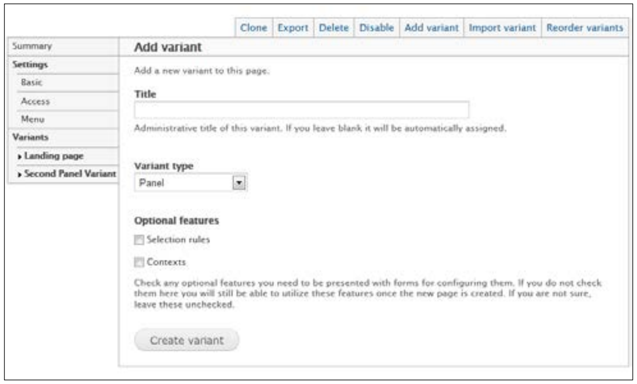
While the 'node/123' URL structure is a perfectly acceptable way to uniquely identify the place where the content for the given node is *defined*, from the point of view of the *business intent* of a given web page, the semantics are quite meaningless.

When we try to use this generic node definition URL for page display we end up having to do a lot of special tricks to get the screen to look the way we want it. We then rely on special properties of the given node (which cannot be known simply from the data in the URL) to decide how it should be 'decorated' with other screen elements. In this case the node is deemed to be an *implicit* contextual data source.

Later, I'll look at an alternative way of specifying contexts to accomplish the same result.

Two Kinds of User Interfaces

There are 2 general approaches to approaching a site building interface: through an administrative UI; or through some sort of 'in place' editing process with a 'toolbox'. The easiest one for us to build is the administrative UI. The game changer would be the 'in place' version because it would take the 'Pull' model to the next level.



Add variant

Add a new variant to this page.

Title

Administrative title of this variant. If you leave blank it will be automatically assigned.

Variant type: Panel

Optional features

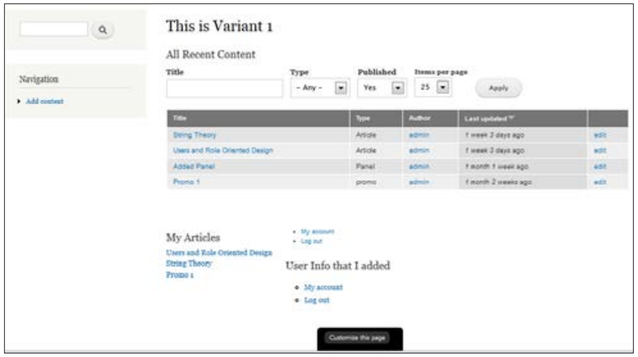
□ Selection rules

□ Contexts

Check any optional features you need to be presented with forms for configuring them. If you do not check them here you will still be able to utilize these features once the new page is created. If you are not sure, leave these unchecked.

This administrative interface for a Panel page is an abstract representation of the page itself. This means the user has to learn a different 'presentation language' in order to configure the page.

The difference between an 'admin' approach and an 'in place' approach is basically the URL where these tasks are done. The administrative UI approach might have a URL such as 'admin/page/123/edit'. In-place editing implies that the user navigates to the actual presentation page and flips some sort of admin Role switch so that the page becomes 'editable'. It may still be possible to design a page creation system that can support either/both.



This is Variant 1

All Recent Content

Title | Type: - Any - | Published: Yes | Items per page: 25

Title	Type	Author	Last updated	
String Theory	Article	admin	1 week 2 days ago	edit
Users and Role Oriented Design	Article	admin	1 week 3 days ago	edit
Added Panel	Panel	admin	1 month 1 week ago	edit
Promo 1	promo	admin	1 month 2 weeks ago	edit

- My account
- Log out

My Articles

Users and Role Oriented Design
String Theory
Promo 1

User Info that I added

- My account
- Log out

Panels In-Place editing has such a Role switch at the bottom of the screen. Clicking on it activates the tools for the page editor Role.

UX Consideration:
Do we want to approach the page building process through an administrative interface or through an 'in place' interface or both?

For the 'in place' method, more complex inputs might require some kind of dialog box. Is this an acceptable approach?

Outside-In Page Building

The Sequence of Steps

The potential behind shifting to a fully Pull based model is that we can straighten out all these twists in the road and make the UX of page building much more straightforward. The Pull model relies on things behind created in the right order:

1. **Create the page**
2. **Assign one or more layouts to a page**
3. **Specify the conditions for how/when each layout should be displayed**
4. **For each layout, choose, configure and assign components (content sources) to the regions**

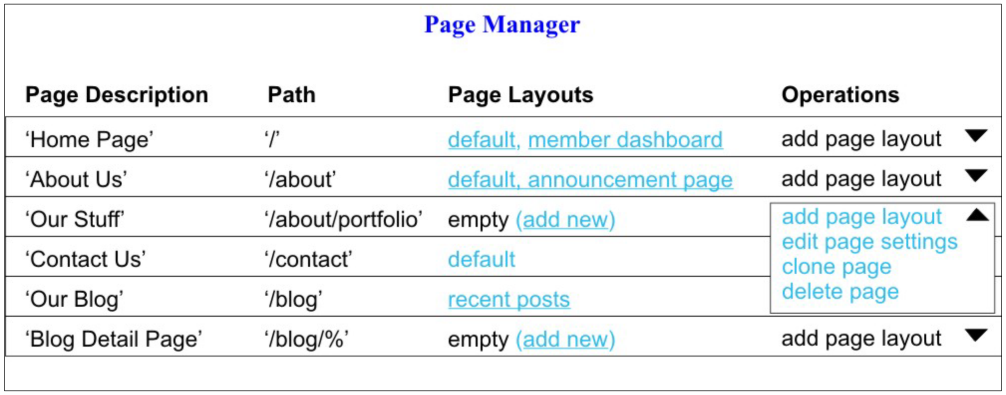
Although the specific UI used for any of this steps can have a huge impact of UX for given types of users, it need not affect the core *process* of page creation as a sequence of logical steps. In fact the value of identifying this sequence is what provides us with maximum flexibility for UI options.

Here are some details about each step.

Step 1: Create the page

A page is essentially a URL. How the URL is specified can vary. It can be done via a page-manager-like interface (for professional site developers) as part of a planned implementation of an IA. Or it could be a more ad hoc, 'outside-in' approach where users have a special tool box (like Wordpress) and never have to type a path. The specific UI method for doing this can vary and is not relevant to the actual sequence described here.

Here is a diagram of a stripped down Page Manager-like administrative interface for creating pages that form an explicit IA Space.



Page Manager

Page Description	Path	Page Layouts	Operations
'Home Page'	'/'	default, member dashboard	add page layout ▼
'About Us'	'/about'	default, announcement page	add page layout ▼
'Our Stuff'	'/about/portfolio'	empty (add new)	add page layout ▲ edit page settings clone page delete page
'Contact Us'	'/contact'	default	
'Our Blog'	'/blog'	recent posts	
'Blog Detail Page'	'/blog/%'	empty (add new)	add page layout ▼

One key difference between this and the existing Page Manager interface is that the page layouts for each page are visible from this top level.

Another key conceptual difference is the idea that a page can be 'empty' and have no layouts assigned to it, yet. This reflects a sort of 'work-in-progress' state and provides a very useful affordance for users who are planning a site based on a clear IA design. It's a way to support efficient IA design without being bogged down with the inevitable hassle of not have content ready even though we know what it *should* be when it gets written.

A logical question is what would the user see if they navigate to an 'empty' page? I would expect that the situation is generally the same as the 'empty front page' condition for Drupal. Visitors would see some sort of default message whereas logged in users with the appropriate permissions could see guidance for creating content in each of the specific empty locations. These guiding links would lead to the UIs described below.

Step 2: Assign one or more layouts to the page

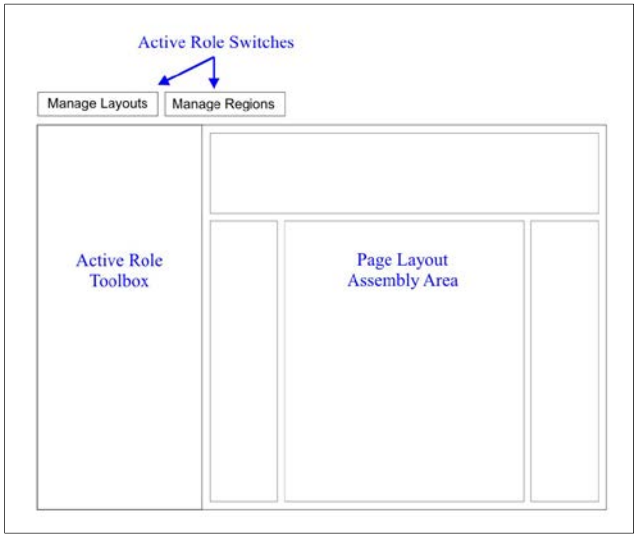
A layout is a set of regions that are containers for blocks (AKA components that specify content sources). The term 'layout' can potentially have two meanings: the 'template' data that defines the regions; and the instance of a given template in a given page. I'll use the term 'layout' to mean the instance and 'layout template' to mean the region design.

So Layout instantiation could be done either by assigning a pre-existing layout template or by creating some kind of custom region design. Custom region designs should be 'saveable' as templates for later re-use.

When a layout is assigned to a page it essentially forms a 'container' into which content can be placed. A given page could have multiple containers like this.

Let's assume the layout templates are available in some sort of library. The UI design question is how and where is that library presented to the user who is building the page? This is where we can use one of the two general approaches: through an administrative UI; or through some sort of 'in place' editing process with a 'toolbox'.

Below is an interface pattern that is a step in the direction of in-place editing. The various pieces can be designed in a variety of ways. The point here is to identify the parts needed to carry out the overall page building sequence.



The main parts of this interface are:

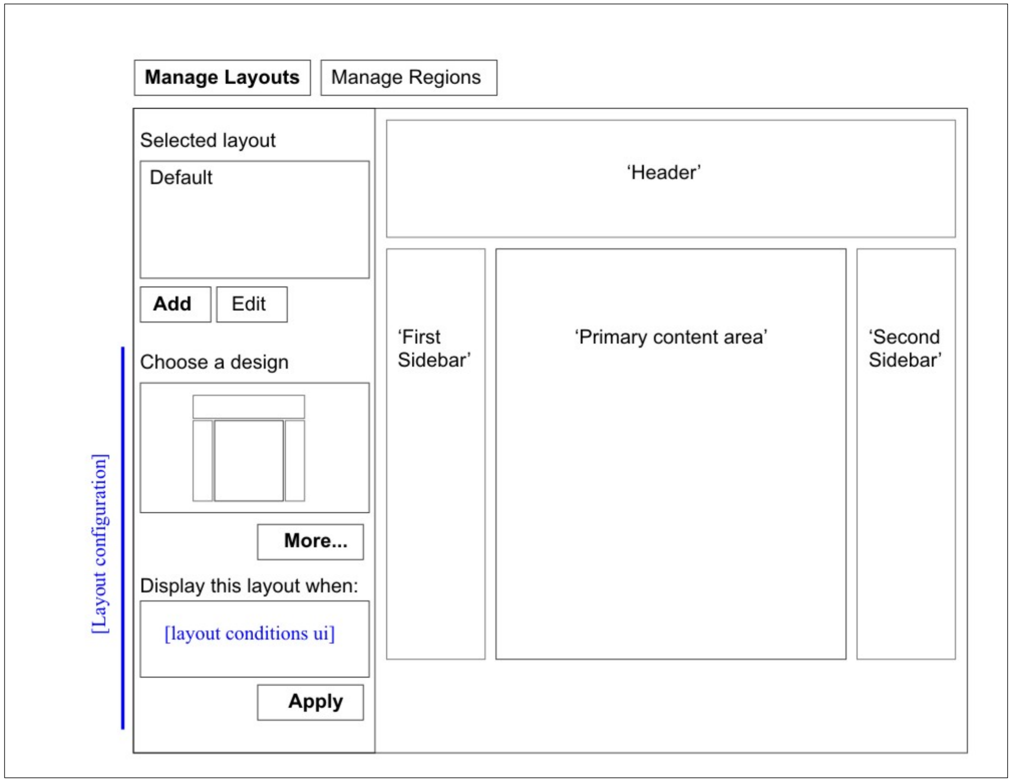
- A 'Role switch' – to determine the task set. For sake of simplicity, this example represents the Role switch as a couple of buttons on a tool bar. (A Role switching mechanism can be represented in a variety of implicit or explicit ways to adapt to any given UI design paradigm.) The two Roles or modes identified here are 'Manage Layouts' and 'Manage Regions'.
- An 'Active Role' toolbox – this is a sidebar that contains the UI for each of the two task sets. This kind of administrative sidebar allows the main screen area to potentially be a true representation of the page layout. For example this kind of mechanism could be compatible with an in-place editing approach.
- An assembly area – ideally as true to presentation format as possible. Again, for an In Place Editing approach this is the page itself but the concept can also work in a more abstract form as an admin interface.

Rather than go into a lot of detail about the layout selection and/or customization (which many people have good ideas about) suffice it to say that would happen under the 'Manage Layouts' mode of this system.

Operations that should be available within this layout workspace are: choose/assign layout template; create custom layout template; save custom layout template to a library; select layout (for editing); specify conditions for layout display. The last operation is the next step in the process.

Step 3: Specify the conditions for how/when each layout should be displayed

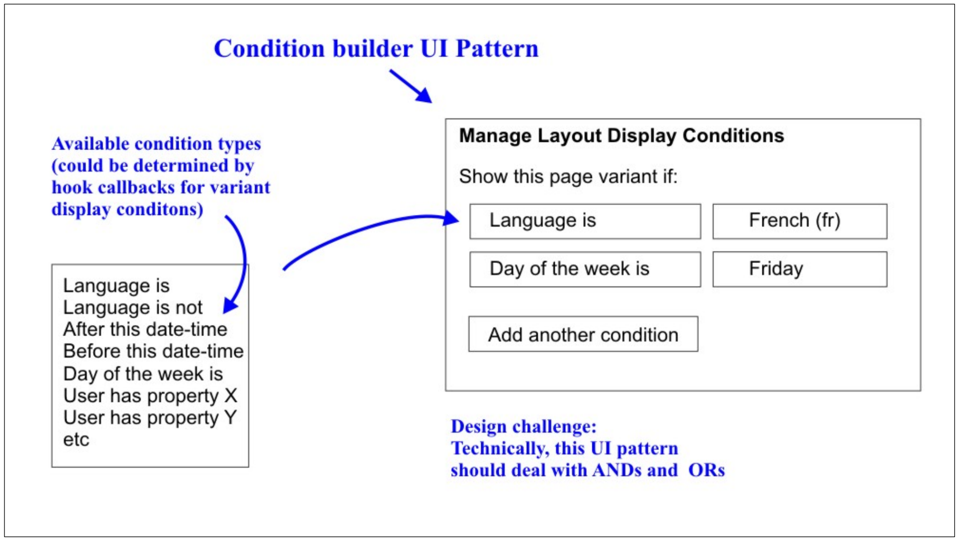
This is comparable to specifying Panels variant display conditions (selection). Only certain contextual factors can be used to do this if we are to achieve a reasonably simple UX. (Display layout X if this. Display layout Y if that.) We need to understand the kinds of dependencies that could lead to activation of a layout for a page. Presumably, they can only be *implicit* contexts and that could include date-time, language, active role, user-based data, (and perhaps user agent?)



But, in this paradigm, we can NOT use path as an implicit context for controlling layout display because the layout is already *explicitly* assigned to this page location. One exception might be a special variable path element such as an OG group.

The conditions for displaying a given layout can potentially be complex and will depend on certain contextual factors. These contextual factors are probably going to be very broad and not precisely the same as contexts used for configuring the *content control* of the components themselves.

What is required to handle the full range of configuration options may be some sort of pluggable system that allows chains of display/selection conditions to be applied. Here is a rough sketch:



Condition builder UI Pattern

Available condition types (could be determined by hook callbacks for variant display conditons)

Language is
Language is not
After this date-time
Before this date-time
Day of the week is
User has property X
User has property Y
etc

Manage Layout Display Conditions

Show this page variant if:

Language is	French (fr)
Day of the week is	Friday

Add another condition

Design challenge:
Technically, this UI pattern should deal with ANDs and ORs

The available condition types might be supplied by modules that can handle the processing. Some sort of integrator would be required of course.

It is quite likely that logical operators would be useful capabilities. For example, supposing (for some reason) a given layout is to be displayed between DATE X and DATE Y, we would need some sort of AND operator to kick in. Supporting logical operators like this always has the potential to make the UI rather complex. That is a tough problem to solve. (An example of one solution is given in the mock ups under the 'Context Configuration Scenarios' section.)

It should be said that a similar display control mechanism would be required at the *component* level also in order to support such scenarios as showing a 'log in' block to anonymous users and a 'logged in' block to those who have signed in.

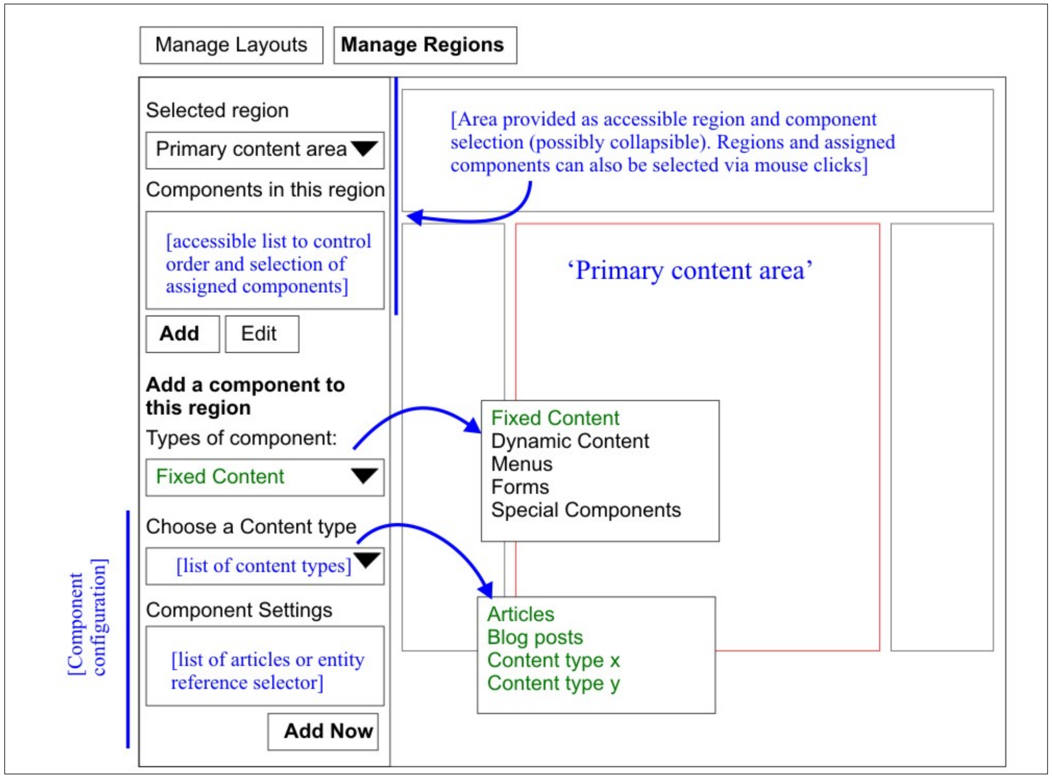
UX Consideration:
For an 'in place' UI approach we need a way of clearly showing components with conditional displays. If they are all assigned to a single layout instance, certain regions may become highly populated with mutually exclusive components.

This tests the limits of an in place editing approach but might be handled fairly easily with an administrative UI component. It may be possible to relieve the stress on an In Place system by showing these details only in a modal dialog or equivalent.

Step 4: For each layout, assign components (content sources) to the regions

The general task is to assign components to specific regions within a layout. There are three kinds of UI problems to solve here.

1. Provide a means of assigning a component to a region.
2. Reduce a potentially huge number of possible blocks down to a manageable set.
3. Configure the selected component - including specifying any contextual relationships to other components.



[Area provided as accessible region and component selection (possibly collapsible). Regions and assigned components can also be selected via mouse clicks]

[accessible list to control order and selection of assigned components]

[list of articles or entity reference selector]

1: Component Assignment

In the plan shown here, the region-component selection can be done by first selecting a region and then choosing a component to put into it. The assignment is done with a form action rather than a drag and drop mechanism. As sexy as drag and drop is, it does require a draggable object to work with. But that raises the question of whether it is dragged *before or after* configuration. To me, ease of configuration trumps having something to drag around the screen. So in this design concept, the adding to a region is the final stage of a configuration form ('Add Now').

The assumption behind this design concept is that regions in the Page Assembly area at the right, and any components within them, would be selectable through direct mouse clicks. However, for accessibility reasons, the regions and components within them are also represented in the 'Active Role Toolbox' UI on the left. This tool box, as a form based UI, supports focus control and provides option widgets for: region selection; selectable items in the assigned components list; explicit buttons for setting 'add' and 'edit' modes; as well as similar controls for being able to manipulate items in the component configuration form.

2: Block Population Management

There are many possible blocks that could be assigned to a region via a component. The number could easily become unwieldy for the user (and even the system) to manage. A way to possibly resolve this is by grouping blocks according to the *kinds of components* that can contain them.

Conceptually it seems there are certain classes of component that might offer the basis for grouping blocks. Choosing the appropriate group via a component type is the first level of component instantiation. Here are some possible component types:

- Fixed content
- Dynamic content
- Menus
- Forms
- Special components

A 'Special Component' is something supplied by a non-core module: e.g. a calendar, external feed like Twitter, etc. If the user selects this type of component then they will need to choose the specific component instance they want from a sub-list.

Technical Note: *How do we populate the component type list? Perhaps we have an info hook to define who wants to present themselves in this way. The hook could allow any module to declare that they provide blocks that are compatible with common or custom component types. Some common sense conventions can be used for tagging the common groups ('fixed content', 'dynamic content', 'menus' and 'forms').*

3: Component Configuration and Contextual Relationships

This is where the meat of the matter is. There are a lot of moving parts within both the task of configuration and that of defining contextual relationships.

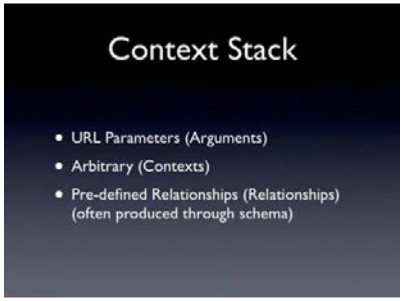
Before getting into this area, it's important to consider what a component is – or could be. Previously I had defined them as 'content sources' and had seen them as simply blocks that had been assigned to regions.

Having done further thinking about this outside-in UI system, I now see components as *wrappers* around blocks which in turn provide the output from content sources. The output of content sources is typically HTML but could also be formatted as JSON, XML, etc. They represent meaningful chunks of data/content that can be described in high level terms and managed by the site builder or content manager. This may or may not be consistent with Eclipse's use of the term 'smart blocks'. In either case I want to spell out how I see 'smart' components can potentially resolve certain subtle but critical UI problems.

Again to re-emphasize, components as wrappers around content source blocks, rather than being simply the blocks themselves, can provide some key points of leverage for the task of UI design.

Perhaps most importantly, components have the potential to resolve the difficulties around understanding, managing and configuring contexts. So far in the community discussions getting users to understand 'context' has been a major point of concern. At the time of this writing it looks like there is some agreement that there could be a logical unfolding of contextual configuration if the sequence of tasks is done correctly. I want to explore that process further in the next section. To do this I'll introduce a system of graphic notation that I think can help clarify the concepts as well as provide a handy system of designing complex component configurations.

Working with Contexts



Context Stack

- URL Parameters (Arguments)
- Arbitrary (Contexts)
- Pre-defined Relationships (Relationships) (often produced through schema)

Find EclipseGc's video at http://www.youtube.com/watch?v=gV0xs89mIOY

3 Types of Context

In his excellent presentation at Drupalcamp Colorado, Kris Vanderwater (EclipseGc) explains the details of contexts as they have been designed within the ctools module. He describes the three kinds of context types:

1. Arguments
2. Arbitrary (now renamed as 'Custom Contexts')
3. Relationships

Arguments come out of the URL. For example 'node/%node' or 'node/123' takes a node out of the URL and loads it for later reference. I refer to this kind of context representation as *'implicit'* because it lies within the object that is indicated by this path structure.

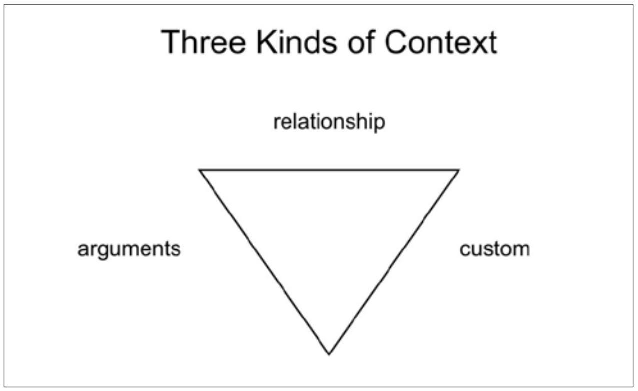
Custom Contexts are arbitrary object loaders. These are *explicit* contexts because the object is defined directly as a parameter to the loader. Examples are:

- I want node 1
- I want term 10
- I want user 13

Relationship contexts are defined by the schema – i.e. the fields that are predefined in objects such as a node or user. Examples are:

- Load the user who authored the node that we are currently looking at
- Load the node that is related through this node reference to the node we are currently looking at

Graphically we can see these contexts as a triangle:



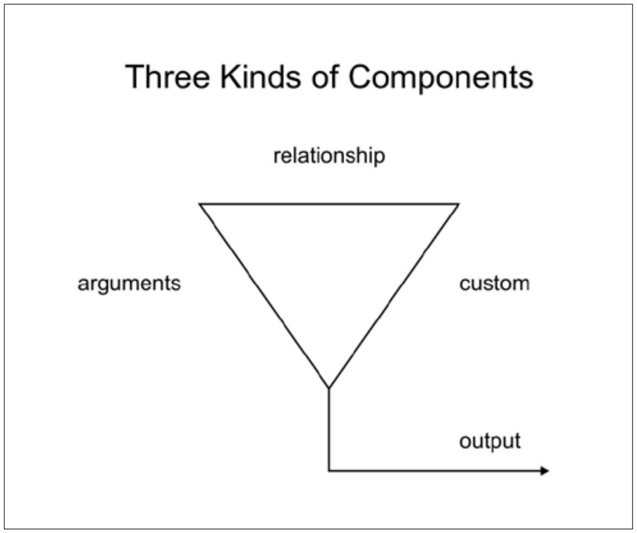
Putting Contexts in Components

Defining components as wrappers around content source blocks provides a means of capturing information about contexts that those source blocks might require. The most important context information that we can capture is the *type* of context that the blocks require to be operable.

So in that sense we could look at a component as having one of three basic types:

1. Argument driven component
2. Custom context driven component
3. Relationship driven component

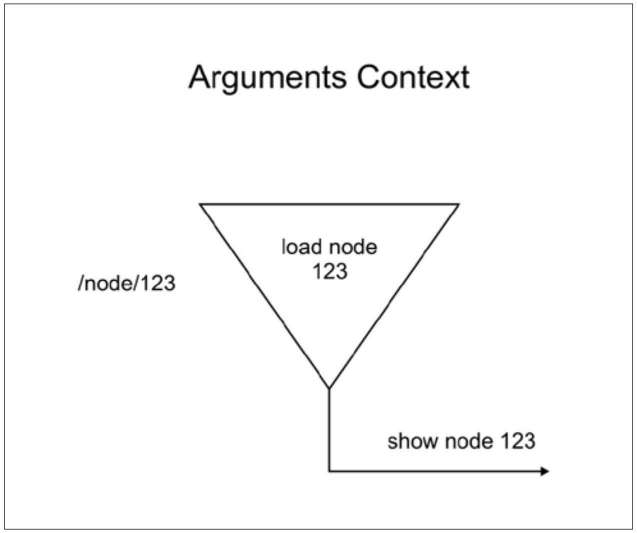
A component is ultimately about producing output so we can adjust the graphic representation to reflect this:



Looking at components as objects that connect content source blocks with contexts and layout regions has some powerful implications for shaping the UX into a more natural workflow. Let’s look at some examples.

A Fixed Context Component

We can use the graphic notation to visualize a variety of context configuration scenarios. Let's start with a simple case: use an argument component type to show a given node via the path 'node/123'



To achieve this, the UI would have to allow the site builder to select a component type that 'knows' how to interpret the path arguments so that the implicit context for node 123 can be extracted.

But in the UI wireframe shown earlier the component type didn't have an option for an 'argument' type. Instead it showed options for 'fixed content', 'dynamic content' etc. The actual *context type* used within any given *component type* is dependent on the type of content source the component type can support. For now I'm going to assume that the context type will be implicitly selected.

In this example we use the 'fixed content' component type which would lead to a second choice for the specific *content source* would be made. The items in this secondary list would be populated by modules that provide 'fixed' content source blocks. In this case it might be the node module that has such a capacity.

Technical Note: *Currently in Drupal, blocks do not take arguments so passing the node id to a content source block is problematic. But since Panels 'blocks' can take arguments and it is the model for D8 layout and page management, I'm going on the assumption that this will not be a problem.*

A Fixed Context Component - Revisited

As I mentioned earlier, the convention of using 'node/123' as a *front end display* URL structure presents problems around knowing precisely the intended meaning of the node content. In other words we can't tell from 'node/123' whether this is a blog post, a piece of static content or a product description. This ambiguity breaks the Pull based UI paradigm.

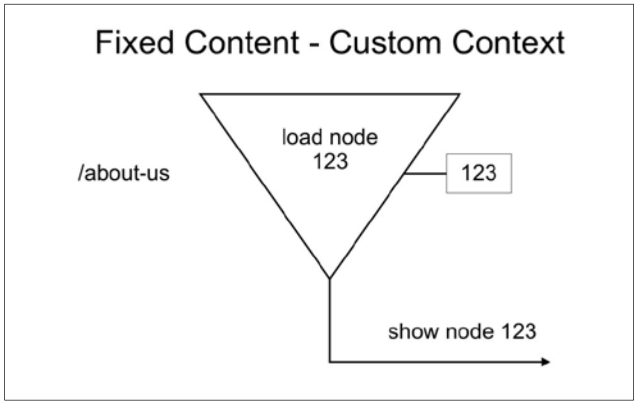
However I also mentioned that as a means of identifying a specific content node definition this URL structure is perfectly fine. In this example I want to show how the *custom context type* can be used to provide an equivalent to 'node/123' while offering an extremely important means of 'normalizing' the context configuration process. In other words, if we take some of the twists out of how we get context data, we can smooth out the actual UI significantly.

As I see it, the biggest twist in the 'node/123' method is the fact that the context is implicit. I strongly believe that implicit contexts are disruptive to the workflow when used at the primary component level.

Implicit contexts are extremely powerful and can be configured with a relative simple UI if they are constrained to the dependent components. Let me explain.

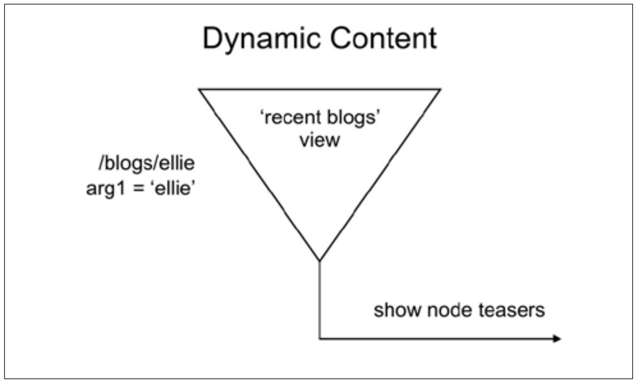
The primary component is what 'node/123' is essentially describing. It is the content provided by nid 123. Given this information, the page can be 'decorated' with contingent elements based on path and/or implicit context. It makes for a difficult UI.

Here is an equivalent method using a custom context type in a fixed content component and using a semantic URL:



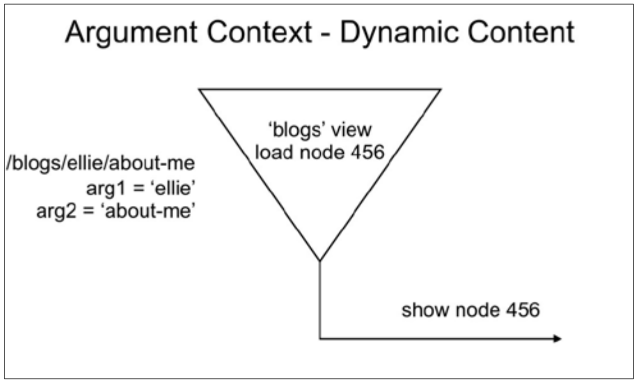
A Dynamic Content Component

The simple fixed content component is relatively easy to grasp. Things get more interesting when we deal with dynamic content as provided by content sources such as views. Here is a schematic for a list of teasers showing recent blogs posts by 'Editor Ellie':



As you can see the component is based on the argument context type and uses a view display called 'recent blogs'. That display takes an argument to filter by author and so 'ellie' is ultimately passed into the view's 'author' parameter. In an IA Space paradigm, this means that a dynamic page location is specified under the path 'blogs/%' where the argument represents author name.

If the user clicks on a teaser link they can read the full post. This implies a change to a different page location where the author name and the post id are provided as arguments.



In an IA Space paradigm, this means that a dynamic page location is specified under the path 'blogs/%/%' where the arguments represent author name and item id (in some alias form).

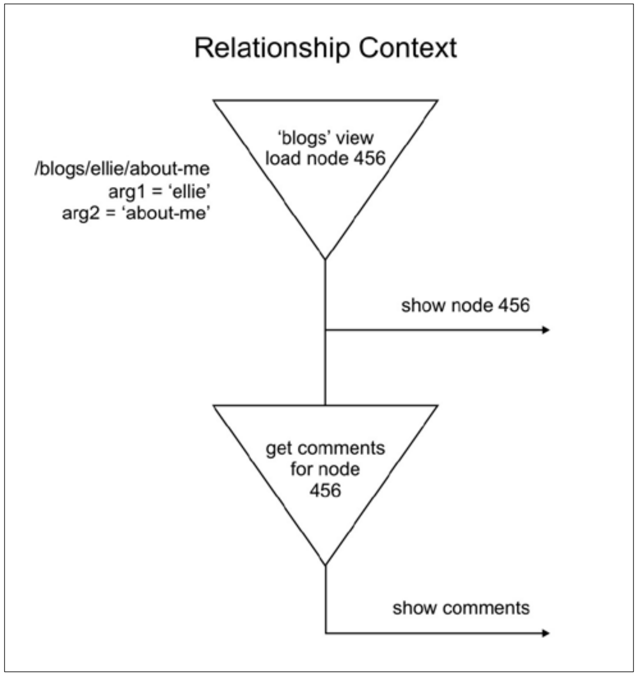
In page building terms this scenario implies that a dynamic content component carrying view display 'blogs' is assigned to the page at 'blogs/%/%'

UX Consideration:
If the linkage is simply to *fields* within the parent, then just having one additional type should suffice.

However if there can be dependent menus, forms and special components etc. then we will need to have more explicit options available.

Dependent Components

Relationship contexts are dependent upon extraneous data sources. In terms of a logical workflow these sources can be defined as components that have already been assigned to a given layout on a given page. Here is an example:



In this scenario we have two components the blog detail and the comments for that post. Because they are different components they can be assigned to different regions within the layout.

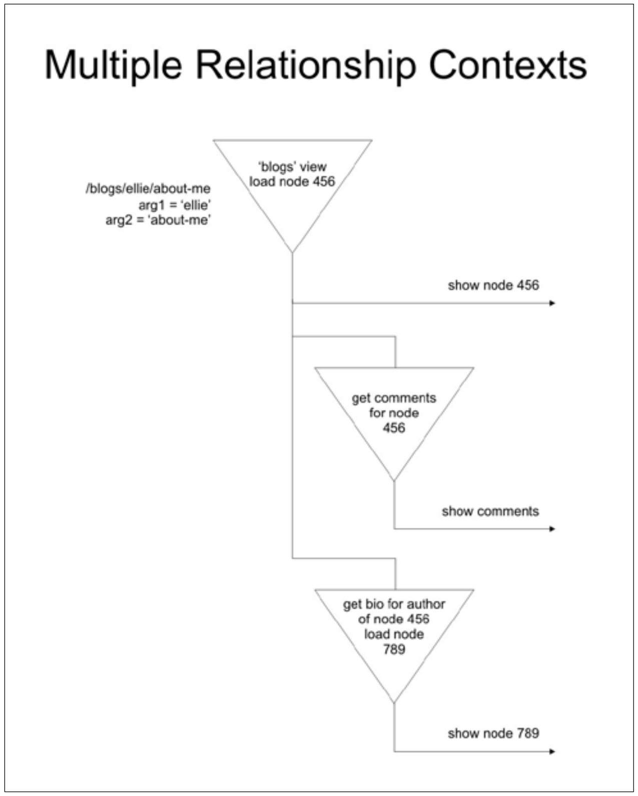
Diagrammatically we show this arrangement as two linked components each of which has its own output. It is important to both understanding the configuration as well as understanding the kind of UI we require to accomplish this, that the 'parent' (upper) component needs to be configured and assigned *before* the 'child' (lower) component.

This implies that we can have another kind of component type in our list of options:

- Fixed content
- Dynamic content
- **Dependent Component***
- Menus
- Forms
- Special components

Multiple Dependent Components

Many scenarios will require more than one dependent component to be referencing a given parent. Here is a schematic showing an extension of the previous scenario that has the author's bio on screen also:



The specifics of how the internal linkage from blog post to author bio is implemented can vary according to the site builder or developer's preference. A brute force (ugly) approach would be to use a node reference for each post. A more sensible approach would utilize a view that takes a node author ID as a parameter.

However it is implemented, the UI will need to be provided with the data that allows us to present a 'bio content source' as an option under the 'dependent' component type.

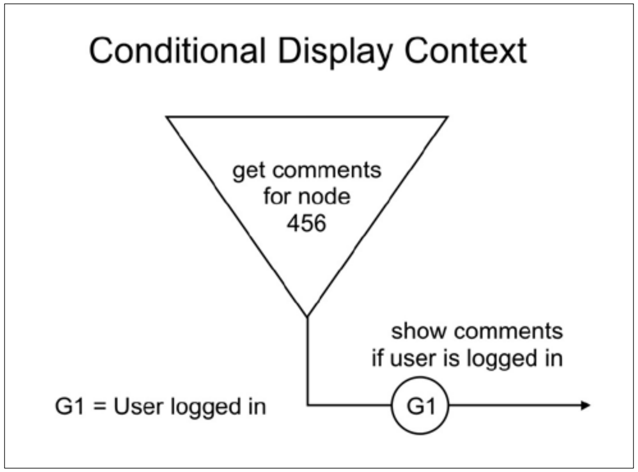
UX Consideration:
Since the linkage may be done through just the node author data, this particular scenario implies that dependent component types that take a node author as a contextual driver may be fairly ubiquitous.

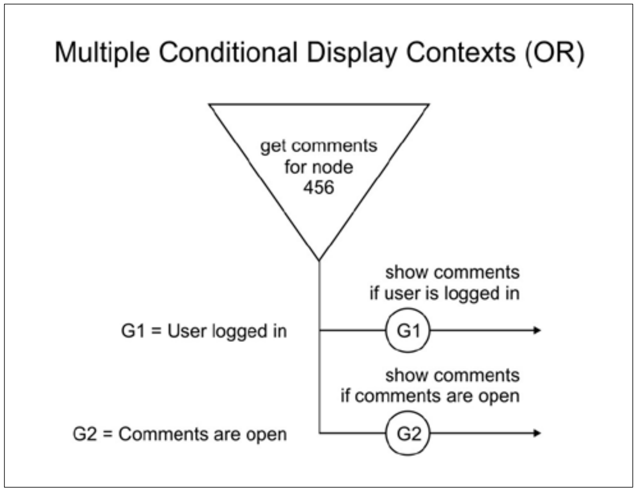
We'll need to examine how this kind of 'ubiquitous component' scenario affects the populations of the component selector lists.

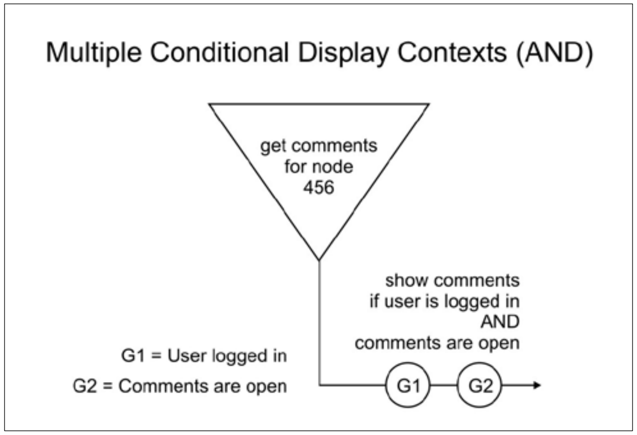
Conditional Display Contexts

It will be fairly common to have scenarios where a given component is displayed only under certain conditions. In contrast to situations where contextual data is used to determine specific content, this represents a different kind of usage of contextual control. We could call them 'conditional display contexts'. (I also like the term 'display gates'.)

It seems logical that these types of contextual controls can be applied to both Layout and Component levels. Furthermore they can potentially take complex configurations because they can (theoretically) support logical operators such as AND and OR. Here are some scenarios:


Conditional Display Context
get comments for node 456
show comments if user is logged in
G1 = User logged in
G1


Multiple Conditional Display Contexts (OR)
get comments for node 456
show comments if user is logged in
G1 = User logged in
G1
show comments if comments are open
G2 = Comments are open
G2


Multiple Conditional Display Contexts (AND)
get comments for node 456
show comments if user is logged in AND comments are open
G1 = User logged in
G2 = Comments are open
G1
G2

Context Configuration Scenarios

The following pages describe more detailed workflow for adding, editing and configuring components. The scenarios are based on those illustrated in the previous section 'Working with Contexts'.

Each scenario shows one or more mock ups and describes the step-by-step tasks that need to be done by the site builder.

The mock ups are all based on an 'in-place editing' paradigm because it is the most challenging approach. However the logic of the workflows could also be applied to an 'admin UI' approach.

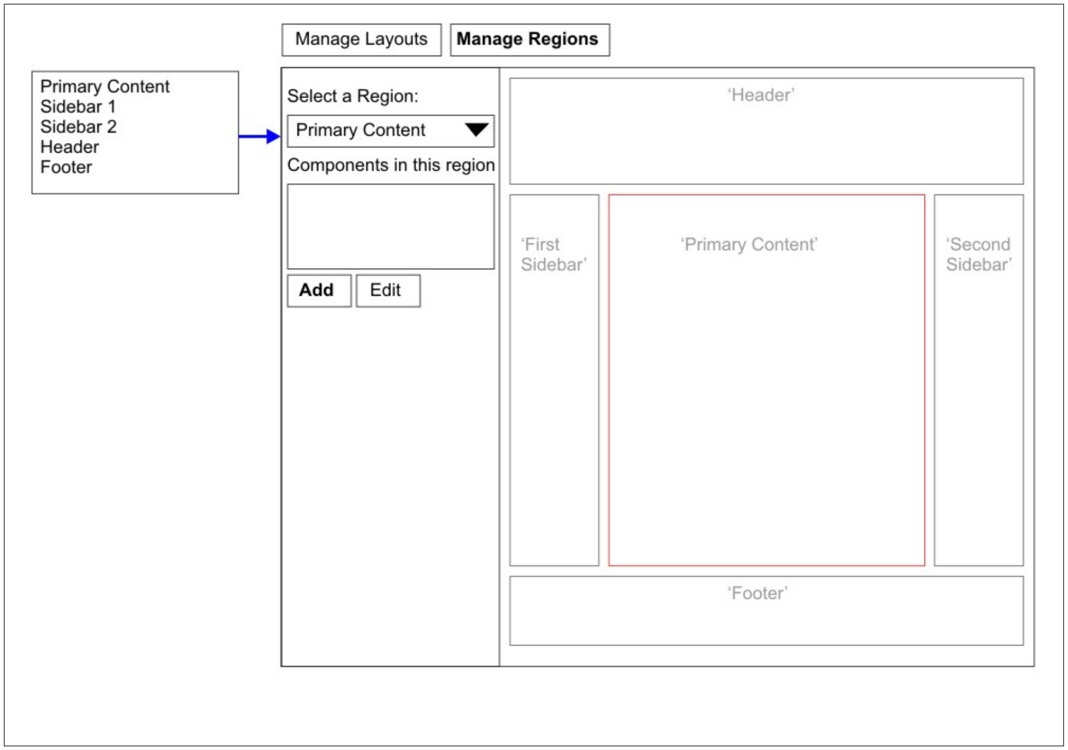
There has been a lot of discussion about the order of operations for page building and underneath this topic is the question of identifying some sort of 'main context' that can be used to supply data to various dependent components. So far the technical necessities seemed to dictate that the context must be identified as before the component is chosen. As I understand it, this is deemed necessary in order to keep the component selection lists down to reasonable sizes (context selection acts as a filter) and also to establish the user's choice for primary data source.

Many people (including myself) see the very idea of 'context' (along with the tasks of context selection and configuration) as being too abstract for non-trained users. Yet this data has to be established because it is the 'connective tissue' for the content in a given layout.

It may be already apparent but I want to emphasize that my approach is to use the more 'physical' notion of components selection and configuration as an alternative approach. I believe that following the principle of 'definition-usage pairing' can satisfy both the technical requirements as well as the need for a more intuitive interaction strategy. In the following scenarios, I'll expand on this approach.

Fixed Content – Custom Context

Step 1



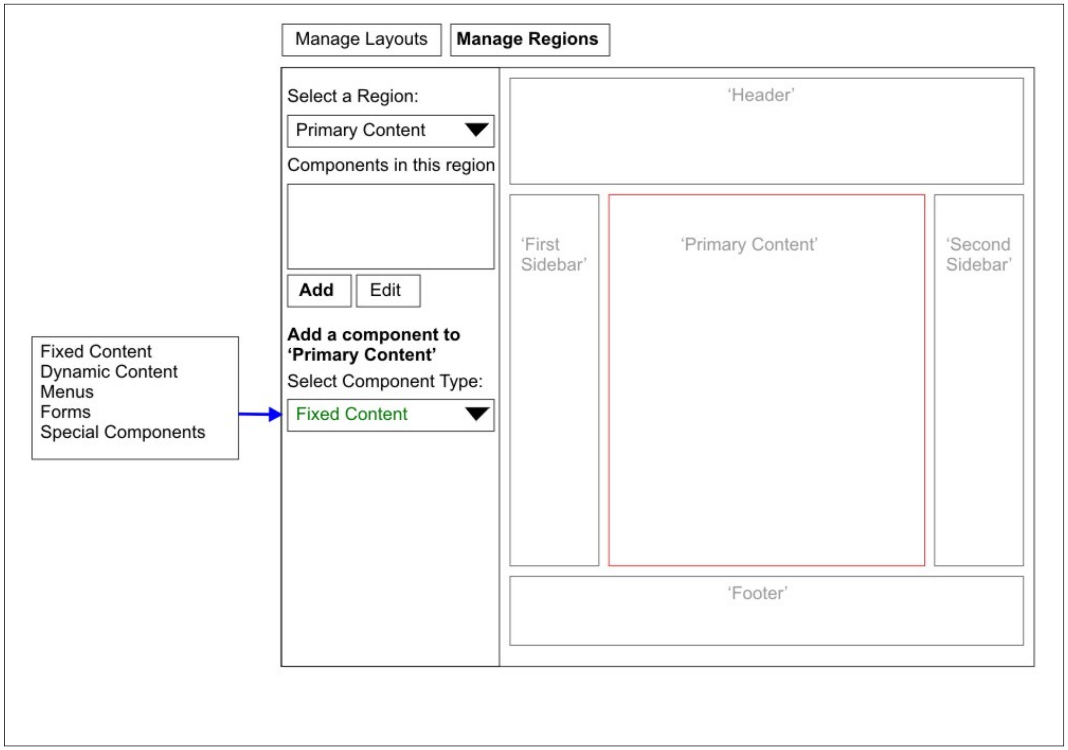
Assuming the layout for a given page has been defined, the first step in populating it with components is to select a region. For these illustrations I'm just going to use the main content region which I've named 'Primary Content'.

The selection of the region can be done through any kind of selector. Here we use a drop down list.

Ideally we would have intelligence in the system to allow this selection to be made by clicking on the region boundary in the page assembly area at the right. Also, to minimize the visual impact of the in-place editing system, it would be best if the selection technique did not involve artefacts on screen (such as special divs for selection handles).

There is a dependency implied in this first step. Once the region is selected, the 'Add' link has meaning. (Whereas the editing initially does not) Clicking on the 'Add' link presents the component type selector in the area below.

Step 2



Once the region is selected, the user must choose a type of component. This is a 'grouping' concept that allows the populations of content sources (blocks) to be more easily managed in the UI.

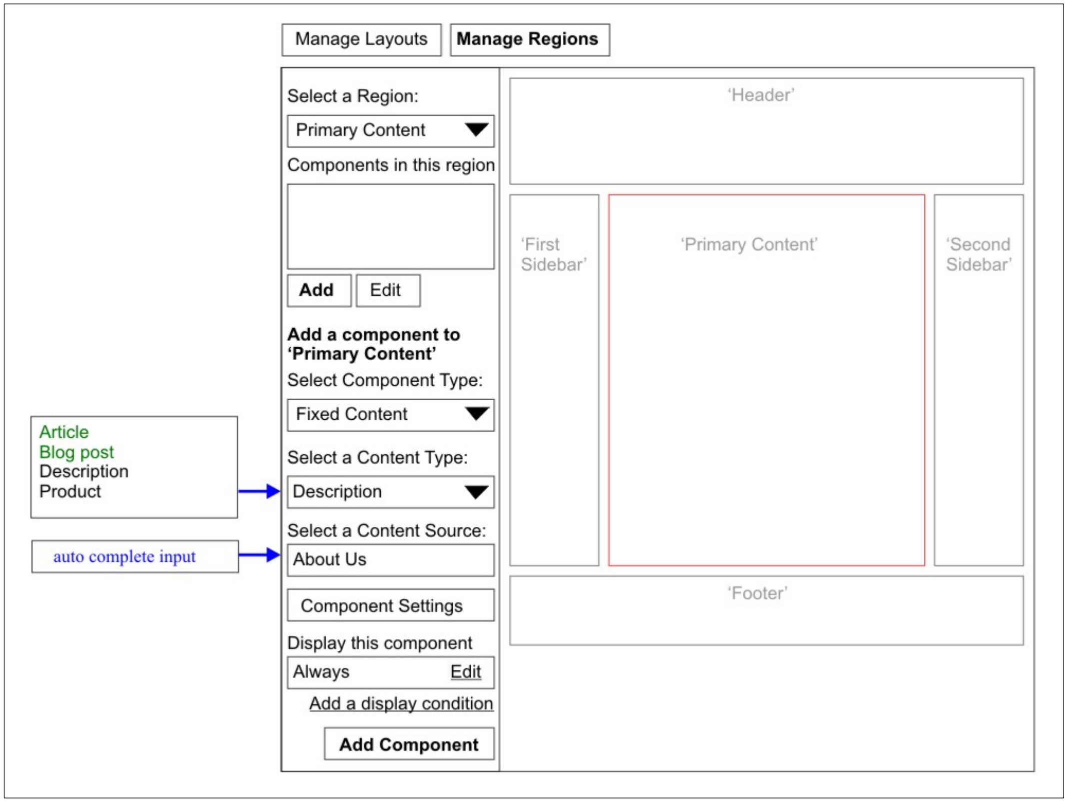
The groups (or types) could include:

- Fixed Content
- Dynamic Content
- Menus
- Forms
- Special Components (supplied by contrib or custom modules)

Once the type of component is set, the appropriate *configuration form* can be determined and presented in the space below.

UX Consideration: The mechanism for conditional appearance of the type-specific configuration form is a somewhat advanced interface technique. For now I'm going on the assumption that it is achievable.

Step 3



Select a Region: Primary Content

Select Component Type: Fixed Content

Select a Content Type: Description

Select a Content Source: About Us

Display this component: Always Edit

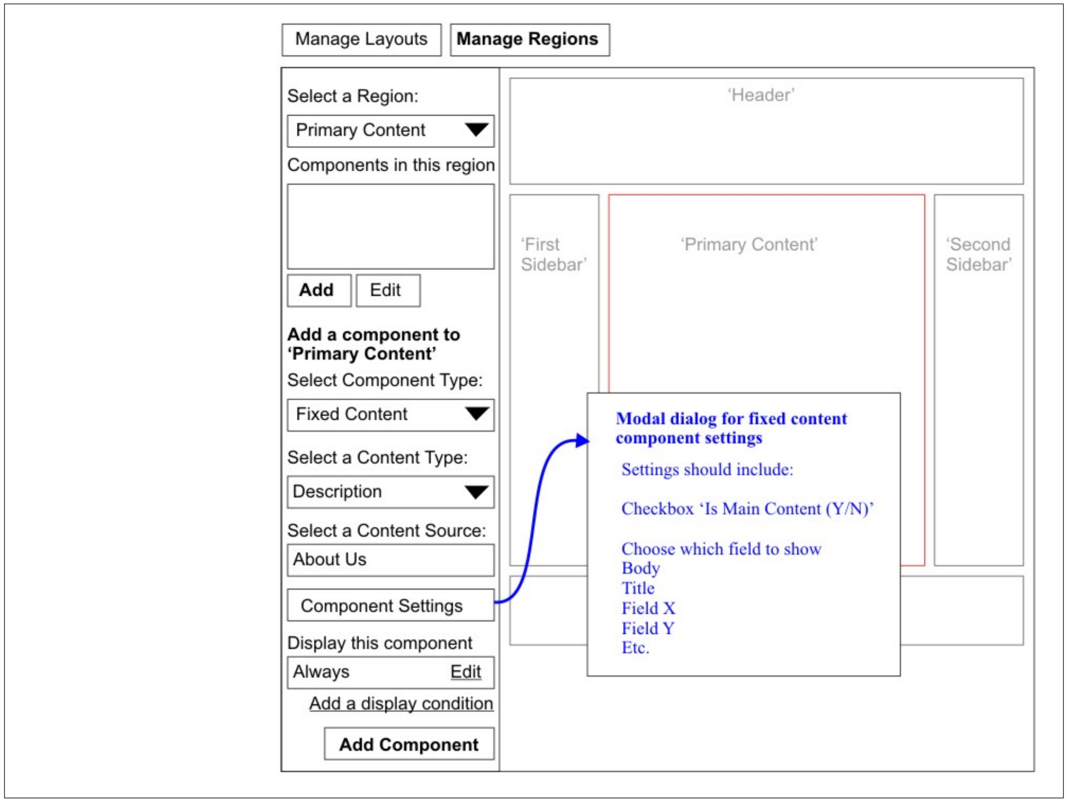
The Fixed Content 'Add' form has a sequence of steps. Fixed content uses the 'custom context' type and is configured by selecting a specific content item.

The user can select a specific content type here to restrict the candidate nodes. Having chosen the content type they can identify the exact instance of the content type using the next input field.

UX Consideration: Arguably, if the content item selection uses an 'auto-complete' method, there was no need to have chosen the content type in the previous input. However this two-stage technique may be helpful if some items are titled in similar ways or even to require less typing in order to resolve the setting.

The user can optionally adjust the display conditions if they need to. For example, if a component should only be displayed when the user is logged in or if the language is set to French then they can specify it here. Conditions can be added as needed to form OR logic. AND logic would be specified within each row. The default is to *always* show the component.

Step 4



Select a Region: Primary Content

Components in this region

Add a component to 'Primary Content'

Select Component Type: Fixed Content

Select a Content Type: Description

Select a Content Source: About Us

Display this component: Always Edit

Modal dialog for fixed content component settings

Settings should include:

Checkbox 'Is Main Content (Y/N)'

Choose which field to show
Body
Title
Field X
Field Y
Etc.

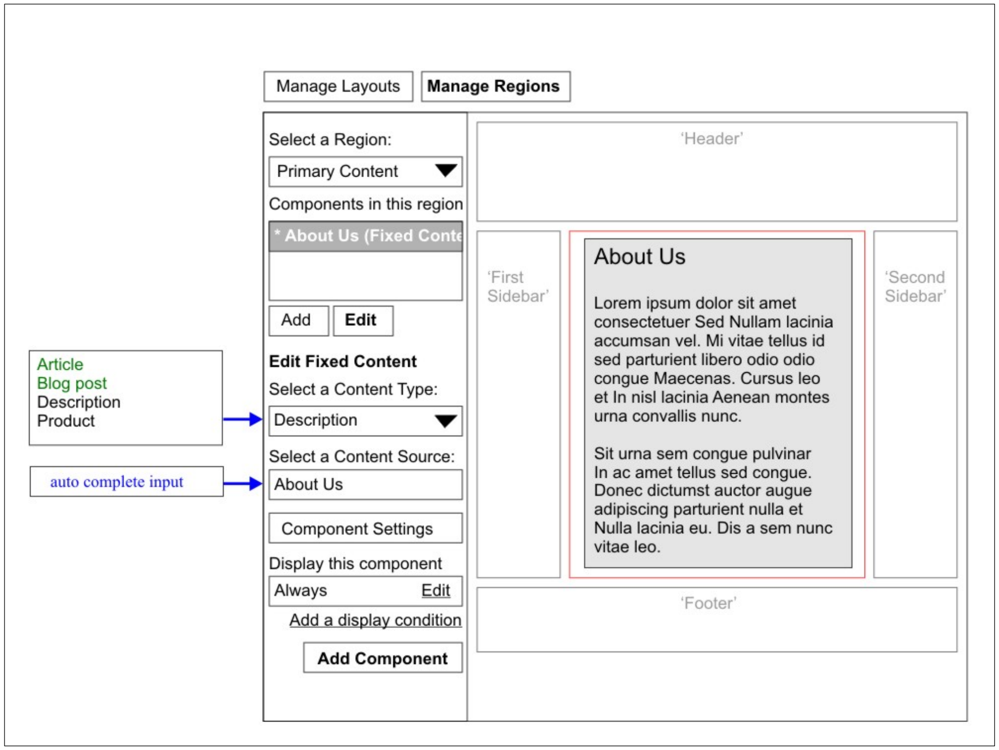
Choosing the content source (which in this case is a specific node) makes it possible to declare it as the primary context from which other, dependent components can pull data. This is an explicit (more 'tangible') alternative to specifying the primary context implicitly through a path such as 'xyz/%node'. Making the context selection explicit allows users to think in terms of things rather than abstract concepts.

The status of the component as the provider of the main context is done in the settings. It could be done with a checkbox for example.

Also in the component settings dialog would be the choice of a precise field that should be shown in the component. It might be a typical use case that site builders begin with placing the body or perhaps event title.

In order to normalize the UI for various kinds of content sources, the specific component settings are hidden from this level and made available through a modal dialog.

Fixed Content – Editing



Select a Region: Primary Content

Components in this region: * About Us (Fixed Conte

Edit Fixed Content

Select a Content Type: Description

Select a Content Source: About Us

Display this component: Always Edit

About Us

Lorem ipsum dolor sit amet consectetuer Sed Nullam lacinia accumsan vel. Mi vitae tellus id sed parturient libero odio odio congue Maecenas. Cursus leo et In nisl lacinia Aenean montes urna convallis nunc.

Sit urna sem congue pulvinar In ac amet tellus sed congue. Donec dictumst auctor augue adipiscing parturient nulla et Nulla lacinia eu. Dis a sem nunc vitae leo.

Once a component is added to a region, its name appears in the component list for that region. This name is selectable and thus allows the user to edit the configuration.

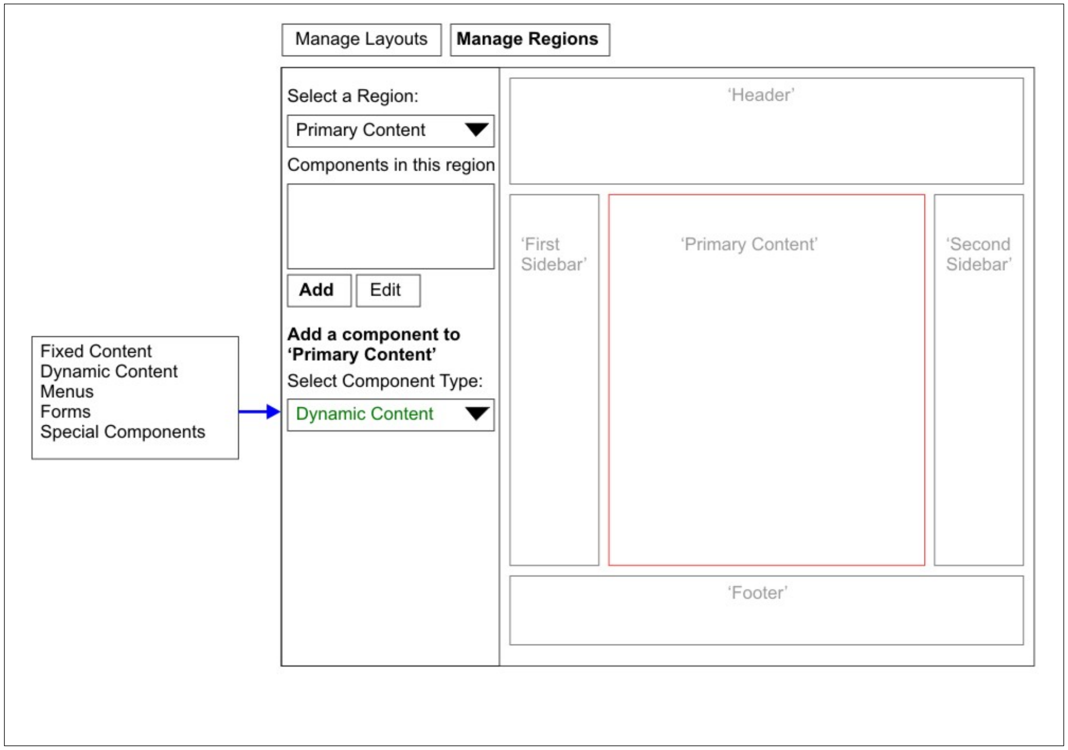
The same form as the 'Add' task is used but the 'Component Type' selector is not present for editing. From the user perspective there is no real sense in changing the type because it is really an 'apples and oranges' kind of decision. The same result can be accomplished by deleting the old component and adding a new one.

Again, ideally we'd see what the content actually looks like in this 'in-place' style of UI.

Note: A real world case may have separate components for the title and the body.

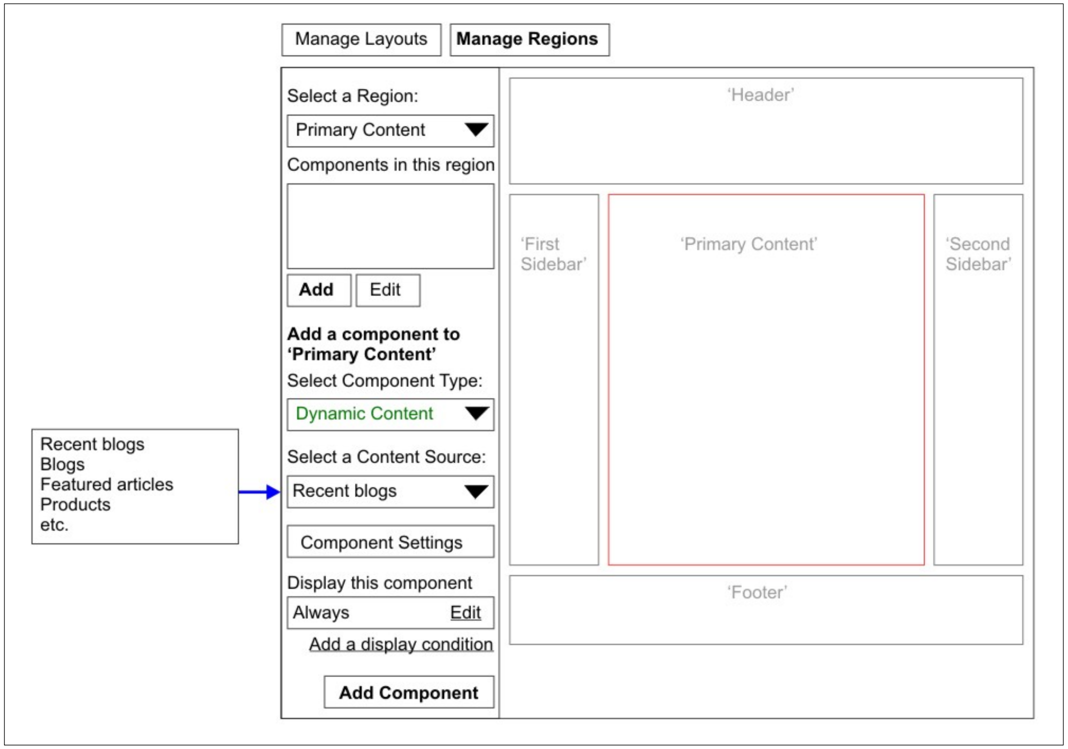
Dynamic Content

Step 1



The user begins by selecting the region and then selecting the dynamic content component type. This presents the 'dynamic content' component configuration form in the area below.

Step 2



Select a Region: Primary Content

Select Component Type: Dynamic Content

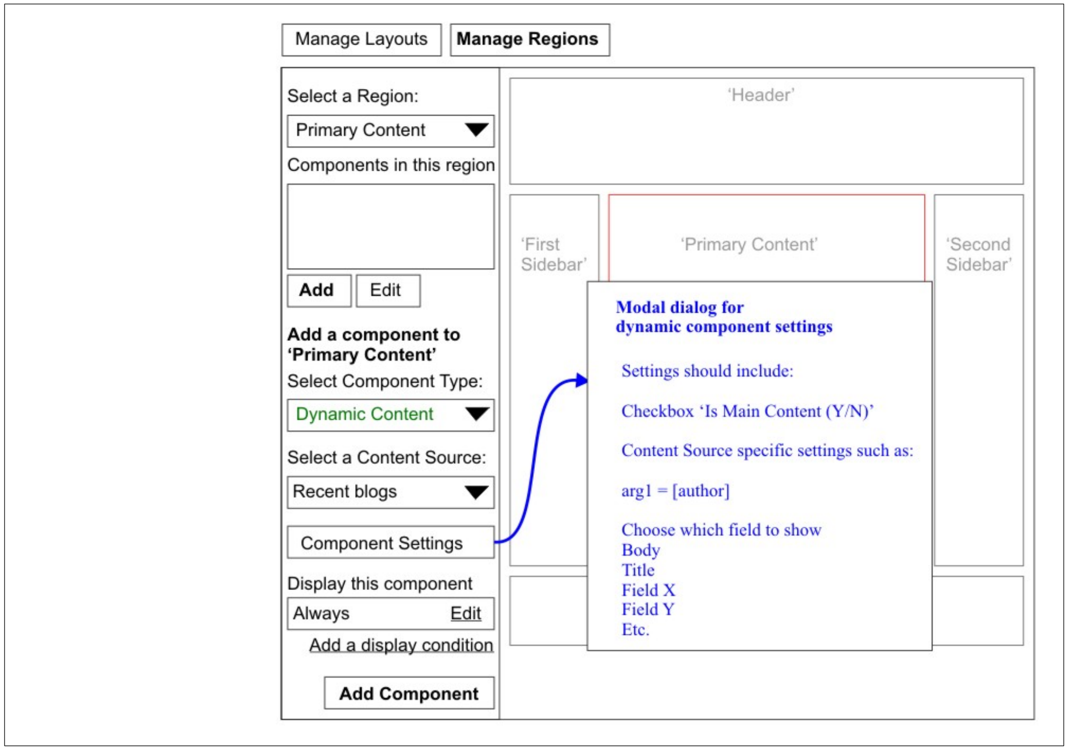
Select a Content Source: Recent blogs

Display this component: Always Edit

Once again there is a sequence of steps to the configuration form. The first is to select the 'Content Source' which typically would be a list of view displays or equivalent.

In this example we choose the view display called 'Recent blogs'.

Step 3



Modal dialog for dynamic component settings

Settings should include:

Checkbox 'Is Main Content (Y/N)'

Content Source specific settings such as:

arg1 = [author]

Choose which field to show
Body
Title
Field X
Field Y
Etc.

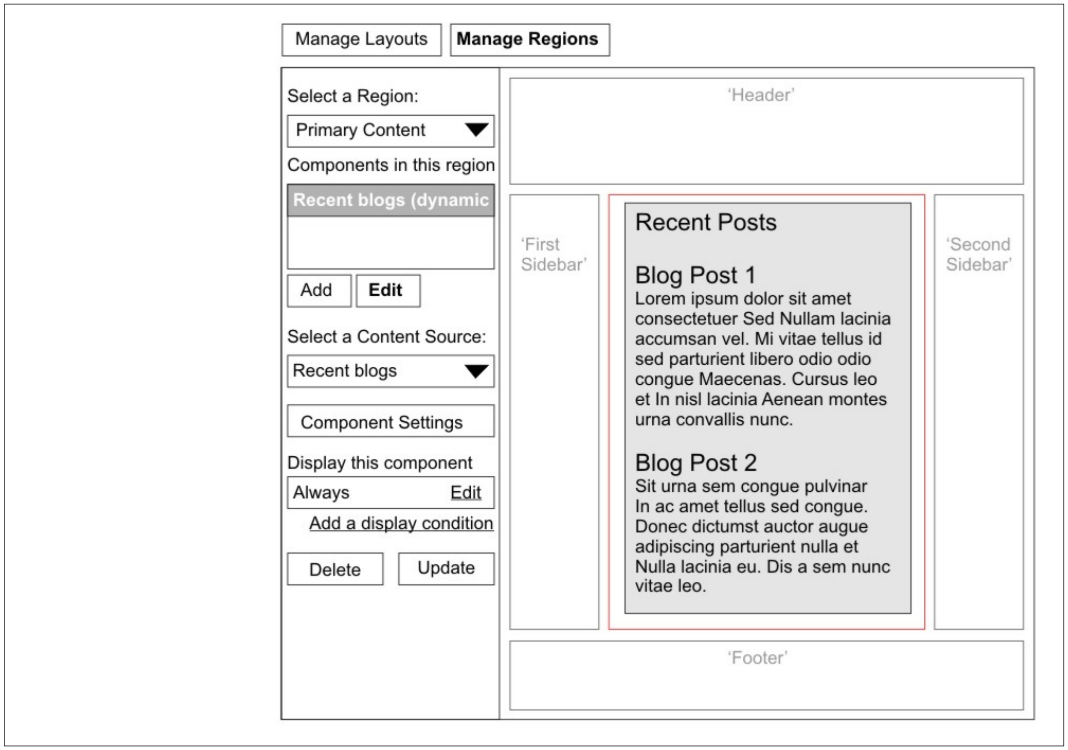
As with the fixed content component the specific settings are hidden from this level and made available through a modal dialog. (Alternatively an admin UI approach could represent this as a form shown directly on the admin page.)

Since an entity that is derived dynamically can also serve as a main context source, the option is provided in the settings dialog.
Note: My working assumption at the moment is that once a component has been given the status of main context source the option should not be available to other components.

In addition to providing a way of specifying the exact field or content data, the dialog would have to allow the user to specify the argument that is required by the view. In this case it's a filter for the author id.

The diagram shows: *arg1 = [author]* which is probably backwards since we are really trying to specify that the [author] data is coming from the URL. It might be better to notate it as: *[author] < arg1*

Dynamic Content - Editing



Select a Region: Primary Content

Select a Content Source: Recent blogs

Display this component: Always

Recent Posts

Blog Post 1
Lorem ipsum dolor sit amet consectetuer Sed Nullam lacinia accumsan vel. Mi vitae tellus id sed parturient libero odio odio congue Maecenas. Cursus leo et In nisl lacinia Aenean montes urna convallis nunc.

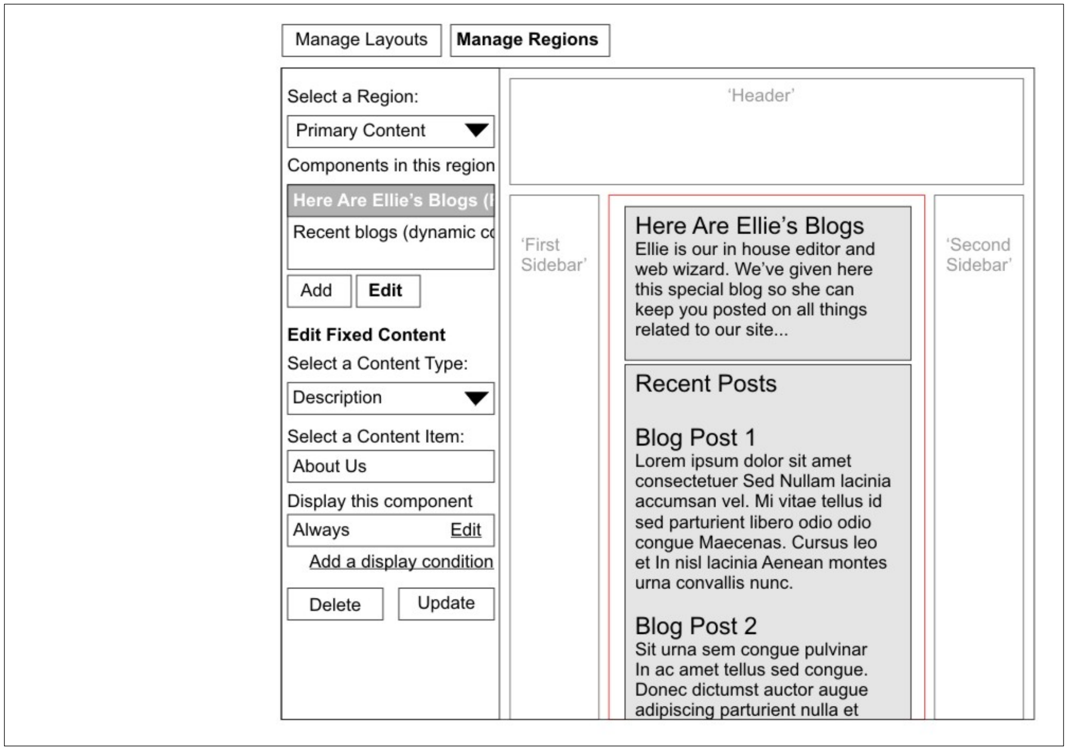
Blog Post 2
Sit urna sem congue pulvinar In ac amet tellus sed congue. Donec dictumst auctor augue adipiscing parturient nulla et Nulla lacinia eu. Dis a sem nunc vitae leo.

Again the user can invoke the editing UI by first clicking on the appropriate item in the component list and clicking the 'Edit' link.

UX Consideration: Alternatively the edit form could be displayed by simply clicking on the component and not having to click on the 'Edit' link. This may or may not be more efficient and the design choice should consider speed of the display, the cost of accidental clicks, etc.

Determining what constitutes a main context in this example is less obvious than in the fixed content case because there could be many scenarios. There are multiple nodes shown in teaser form in the middle region so it doesn't make sense to refer to them. However one viable scenario is that the blog posts shown are filtered to a specific author. Assuming this is done through a separate page that uses a path that includes a wildcard for the author id, then that wildcard could be used as a main context. This would allow other components to reference that data to show things related to that author.

Multiple Components



Select a Region: Primary Content

Select a Content Type: Description

Select a Content Item: About Us

Display this component: Always

Here Are Ellie's Blogs

Ellie is our in house editor and web wizard. We've given here this special blog so she can keep you posted on all things related to our site...

Recent Posts

Blog Post 1

Lorem ipsum dolor sit amet consectetuer Sed Nullam lacinia accumsan vel. Mi vitae tellus id sed parturient libero odio odio congue Maecenas. Cursus leo et In nisl lacinia Aenean montes urna convallis nunc.

Blog Post 2

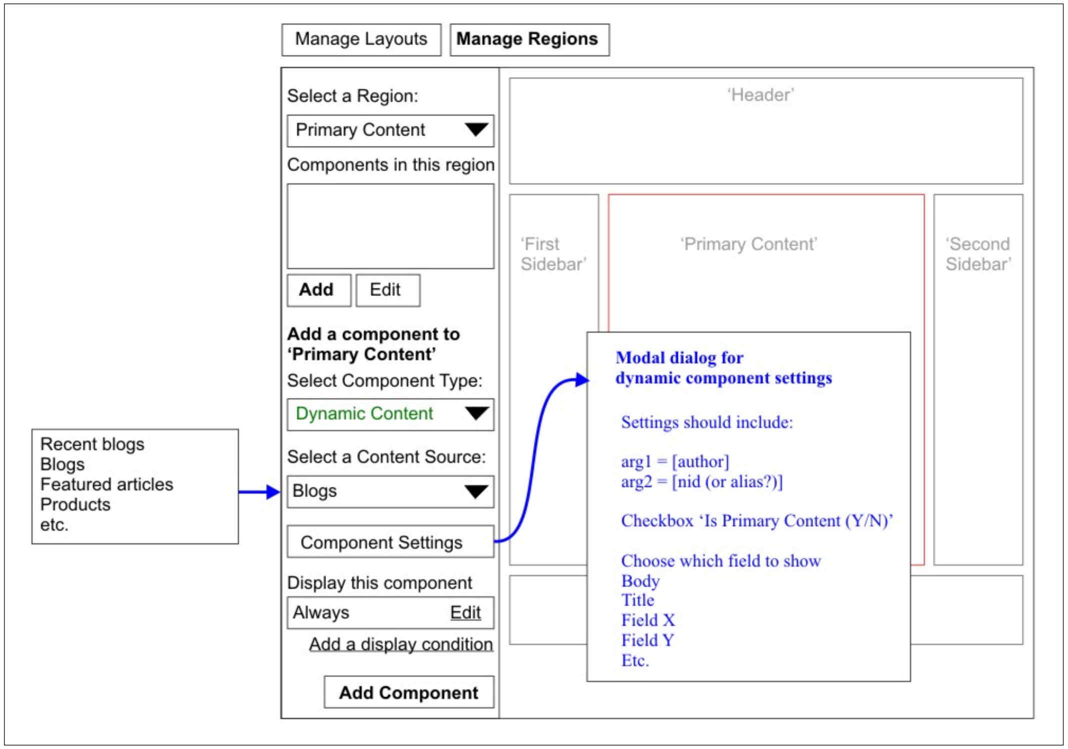
Sit urna sem congue pulvinar In ac amet tellus sed congue. Donec dictumst auctor augue adipiscing parturient nulla et

The population of regions should allow assignment of multiple and diverse components to a given region. In this example a fixed content component has been added above the recent blog teaser display.

Mechanically, this is easily done through simply adding another component and placing it in the right order within the component list. However there is a more subtle problem in this scenario stemming from the fact that it shows a filtered list of blogs. These are posts from Editor Ellie so some sort of argument has been applied to the dynamic content component. The path for this path would have to be something like 'blogs/%' where the wildcard carries the author filter data.

The complication comes from the 'static' intro above the teasers. If this was a fixed content component it would appear for everyone's blogs. Clearly that is not what is intended for this scenario. Using a dynamic content component that takes an argument would resolve this.

Dynamic Content - Filtered



Modal dialog for dynamic component settings

Settings should include:

arg1 = [author]
arg2 = [nid (or alias?)]

Checkbox 'Is Primary Content (Y/N)'

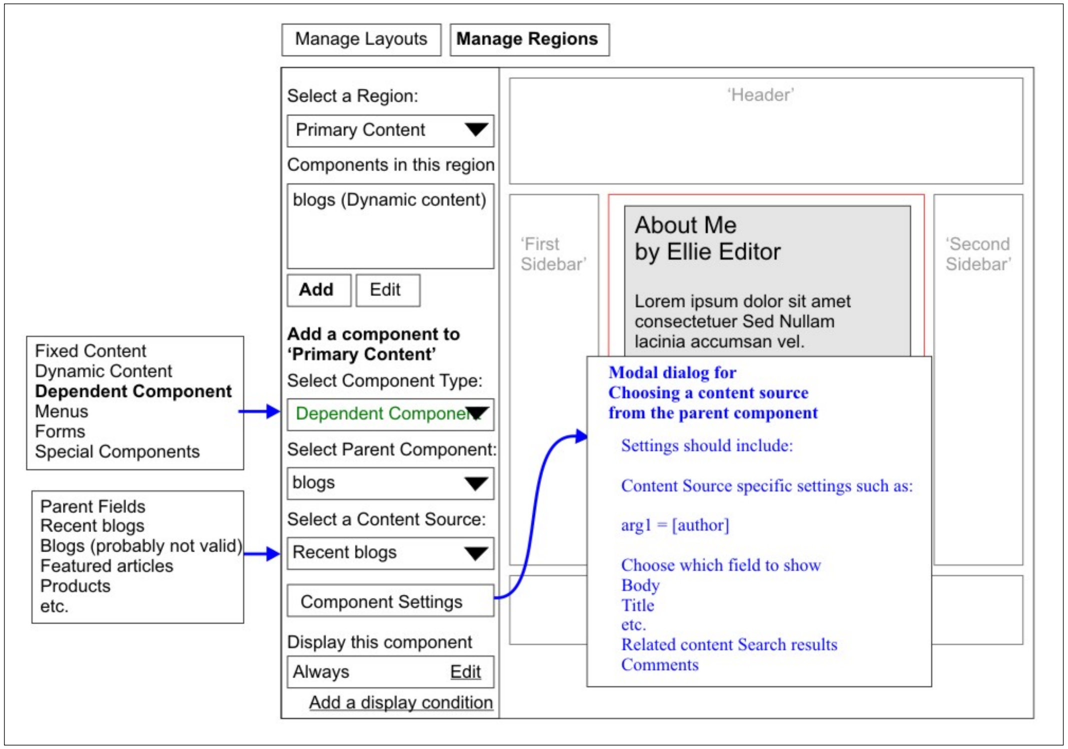
Choose which field to show
Body
Title
Field X
Field Y
Etc.

Using a layout assigned to path '/blog/%/%' we are able to show the details of a given author's blog posts. The dynamic component assign here is 'blogs' which takes arguments for the author and the node id.

Of course the node itself does not require the author ID in order to resolve which node we want. However this is an example of how the node is treated as a defined 'content resource' which could be referenced in a variety of usage contexts. In this particular usage context the navigation path taken was through a choice of viewing just Ellie's blog posts. The same node could be displayed under other paths also (such as 'blogs/recent' that could show recent post by any author).

Once again the details of the settings for this component are taken care of in the modal dialog in order to keep the main interface normalized across all content sources.

Dependent Components



Modal dialog for Choosing a content source from the parent component

Settings should include:

Content Source specific settings such as:

arg1 = [author]

Choose which field to show
Body
Title
etc.
Related content Search results
Comments

Using the principle of definition-usage pairing, a component can be 'defined' as a context source by simply being *assigned* to the layout. It can then be 'used' in the configuration of 'dependent' components.

In this example we have assigned a dynamic content component to the Primary Content region. As a result we now have the option for a 'Dependent Component' type in the drop down list.

Selecting this component type produces a variation of the component configuration form that includes a selector for the 'parent' component. In this case we select the 'blogs' component which was assigned to the 'Primary Content' region. (We should be able to select a component assigned to any region in this layout.)

The content source list now includes the various fields and other related items that can be derived from the selected parent component.

Stresses and Limitations

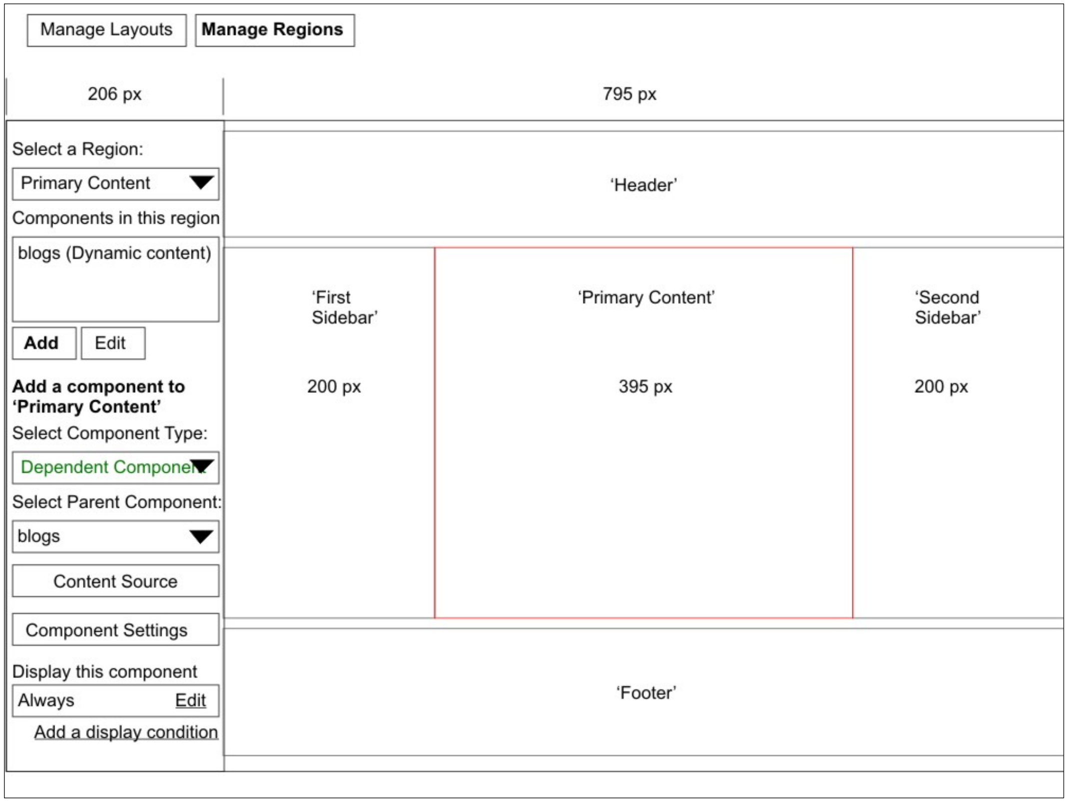
Context Identification

At this point I'm not entirely convinced that the 'main context' needs to be explicitly indicated by the user. The idea of clicking a checkbox to say that a given component is the one and only 'main context' for a layout seems awkward and rather ugly.

My original assumption was that simply identifying a parent component would be sufficient to establish the relationship. This checkbox idea was an afterthought that came up after a discussion about how to ensure that user makes the right choice about the dominant context for the layout.

The scenario presented in this discussion involved setting up a component to show 'related content' through a Solr search operation. The input for the search was deemed to be the title of the item in the primary content region which would be interpreted as keywords.

After further reflection on this, I believe we can achieve this by simply outputting the node title in a given component and then using that component as the parent of the Solr search component. Of course some configuration is required for the search component settings to identify that the title is the provider of the keywords.



Select a Region: Primary Content

Select Component Type: Dependent Compone

Select Parent Component: blogs

Display this component: Always

Screen Proportions

Throughout this document I've made an effort to focus on an In Place editing approach to layout and component configuration. The general UI technique shown in the wireframes places the configuration UI in a 'toolbox' on the left side of the screen.

However we need to keep in mind that this approach relies on ideal screen size conditions and is somewhat dependent on the layout template design. (A more accurate scale is shown here.)

Furthermore, the wireframe UI does not attempt to resolved actual element sizes, fonts, textual cues, etc. This means that in practice the real estate usage could be quite different.

And beyond all that there are factors around mobile and responsive design that should be considered.

Containers, Objects, and Scopes

The component configuration UI involves identifying nested items – pages, layouts, regions and the components themselves. In principle, this could be done with both in-place and form-based selection.

The in-place selection techniques used currently in Panels injects visually disruptive interaction mechanisms onto the screen. As I mentioned earlier, it would be ideal if selection could be done with some clever JavaScript that allows the presentation markup to be unaffected.

Beyond the question of how container-object selection is made, there is also the question of the scope of assignment. For example it is straightforward to see how a given node or view might be assigned to a single page. But in the case of a header certain menus, footer etc., the same component object will be required on many (if not all) pages. This raises the question of how the scope should be defined for a component's layout assignment.

The UI will require an additional input to capture the user's intent on this. Depending on the implantation strategy this UI could either be a checkbox:

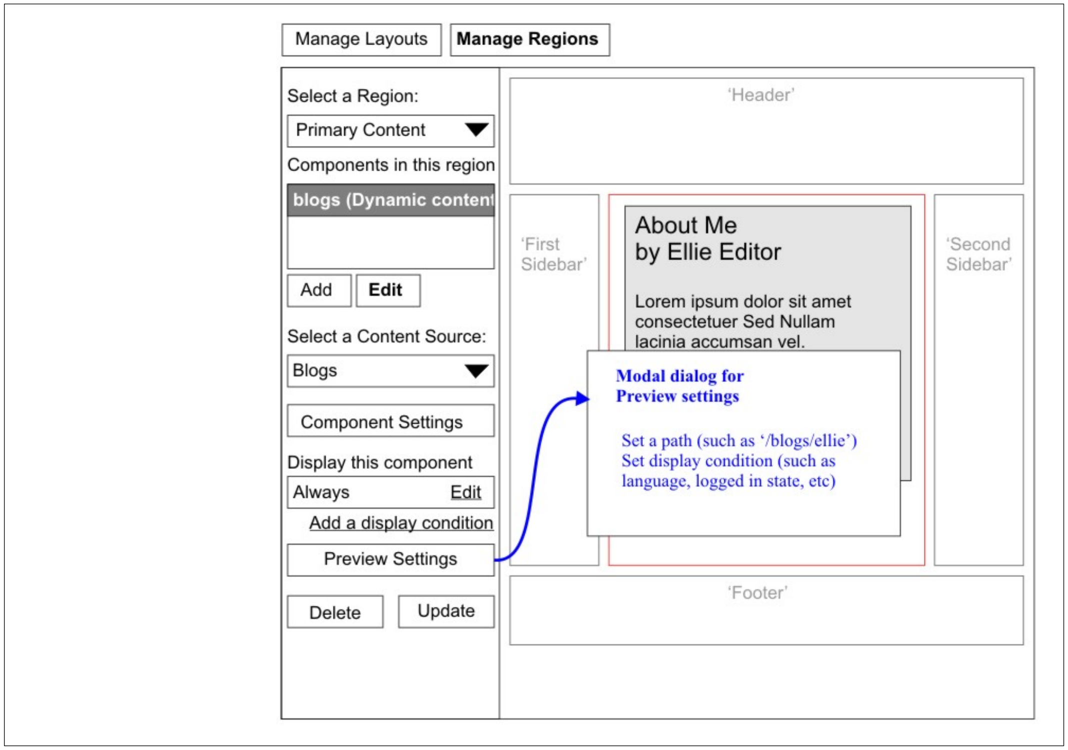
- Apply this component to all paths starting with X

Where X is the URL of the currently configured page:

- '/' (the entire site)
- '/blogs' (everything under the blogs section)
- 'products/hardware/tools'
- etc.

But there are still things to consider with regard to interpreting this information and again this is dependent on the implementation. Are we assuming that this configuration parameter is 'state-based' - meaning all pages, including those created in the future, will automatically get the component assigned to them. Or is it 'instance-based' – meaning it acts like a macro operation and simply instantiates the same component configuration to the same region in multiple pages. There are UX implications and concerns with either route.

Admin vs. In-Place UI Strategies



Lorem ipsum dolor sit amet consectetuer Sed Nullam lacinia accumsan vel.

Modal dialog for Preview settings

Set a path (such as '/blogs/ellie')
Set display condition (such as language, logged in state, etc)

The filtered list scenario suggested by the path 'blogs/ellie' raises another issue with regard to 'admin' versus 'in-place' UI strategies.

From an 'admin' perspective, it is not possible to preview specific content instances from wildcard paths such as '/blog/%' or '/blog/%/%'.

From the 'in-place' perspective, it is not possible to achieve multi-page componetn assignments without specifying a wildcard path, somewhere, in the UI.

No doubt an 'admin' UI would want to do what Views does and allow a path to be explicitly set to determinethe preview conditions. We may also want to support other preview conditions to allow for other contextual factors to be simulated. Examples are: language, logged in state, active role.)

Supporting Workflows of Different User Types

The underlying decision that needs to be made with regard to interaction strategy is how to handle the active Role states. In other words, how do we present the site builder or content manager user with the tools they need to carry out their tasks?

We can see from the logic of the workflow that there are many instances of contingent interfaces. This is probably an axiomatic condition that we can't avoid. It also suggests that a serial interface, such as a wizard, would be generally compatible with the workflow.

While the wizard technique can provide great advantages to new or non-trained users, it can potentially slow things down for more advanced site builders. These users frequently need the efficiency of a more parallel interface that allows them to be more precise about the adjustments they want to make without following the entire sequence of steps.

We may be able to salvage both use cases if we do this right. Let's work through some ideas about this.

First of all, the sequences described in this document can easily be transposed into a wizard format. We can break down the toolbox and elements within it into discrete steps.

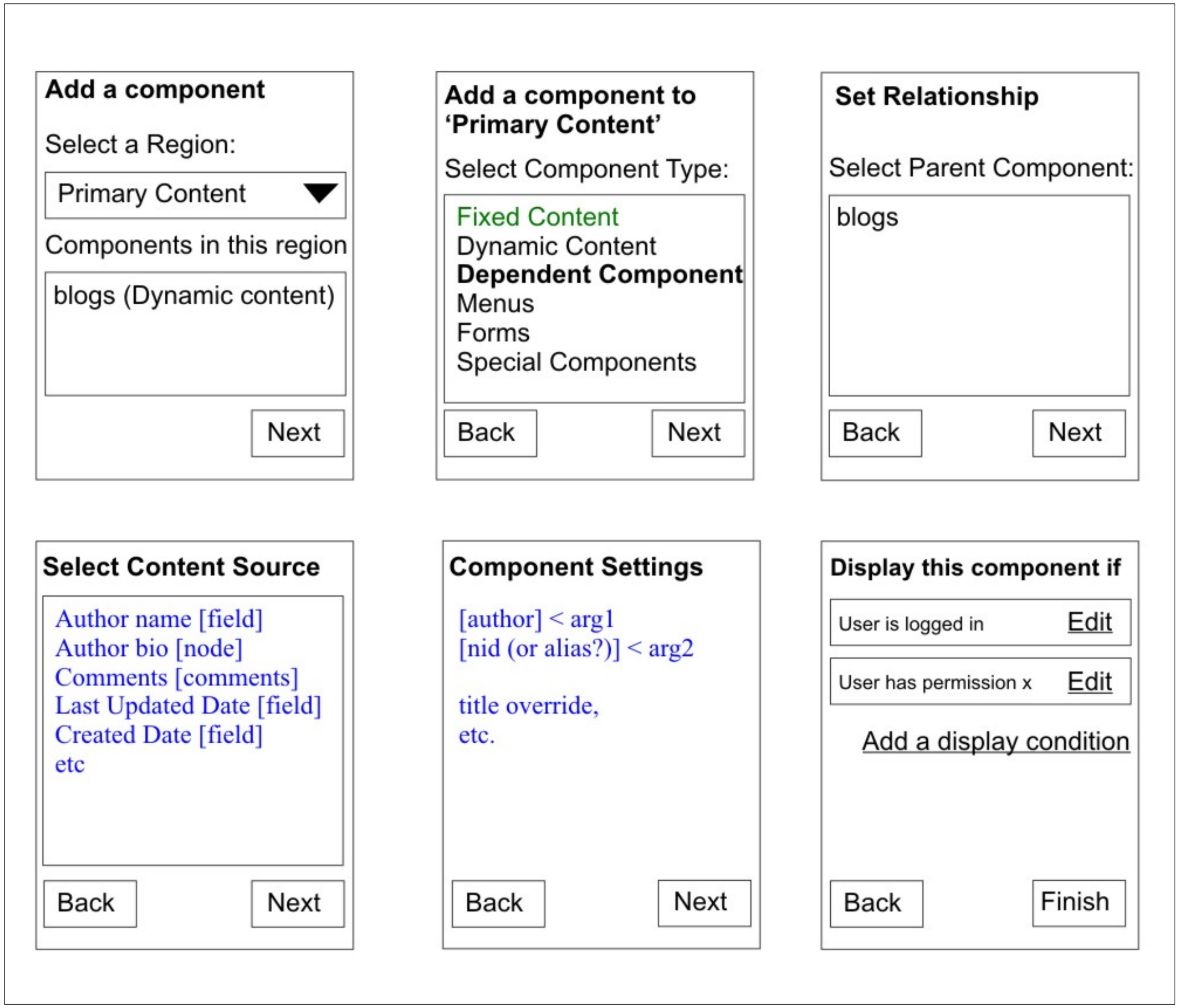
For example, the steps for Adding a component to a region are:

- Select the Region
- Select the component type

<branch point for configuration based on the component type – using 'dependent' type for now >

- Define the parent component
- Select a content source
- Define component settings
- Define display conditions

This breakdown is illustrated on the following page.



Add a component

Select a Region:

Primary Content ▼

Components in this region

blogs (Dynamic content)

Add a component to 'Primary Content'

Select Component Type:

Fixed Content
Dynamic Content
Dependent Component
Menus
Forms
Special Components

Set Relationship

Select Parent Component:

blogs

Select Content Source

Author name [field]
Author bio [node]
Comments [comments]
Last Updated Date [field]
Created Date [field]
etc

Component Settings

[author] < arg1
[nid (or alias?)] < arg2

title override,
etc.

Display this component if

User is logged in Edit

User has permission x Edit

Add a display condition

Optimizing for Productivity vs. Discoverability

User Interface designs can be optimized for helping full time professionals be productive or helping new users understand what to do. In general, there are two main kinds of user types to consider when working out ways to optimize a design:

1. The occasional user – requires discoverability
2. The frequent user – requires productivity

(There is also the potential for 'the occasional frequent user' who requires both when they periodically do repetitive tasks.)

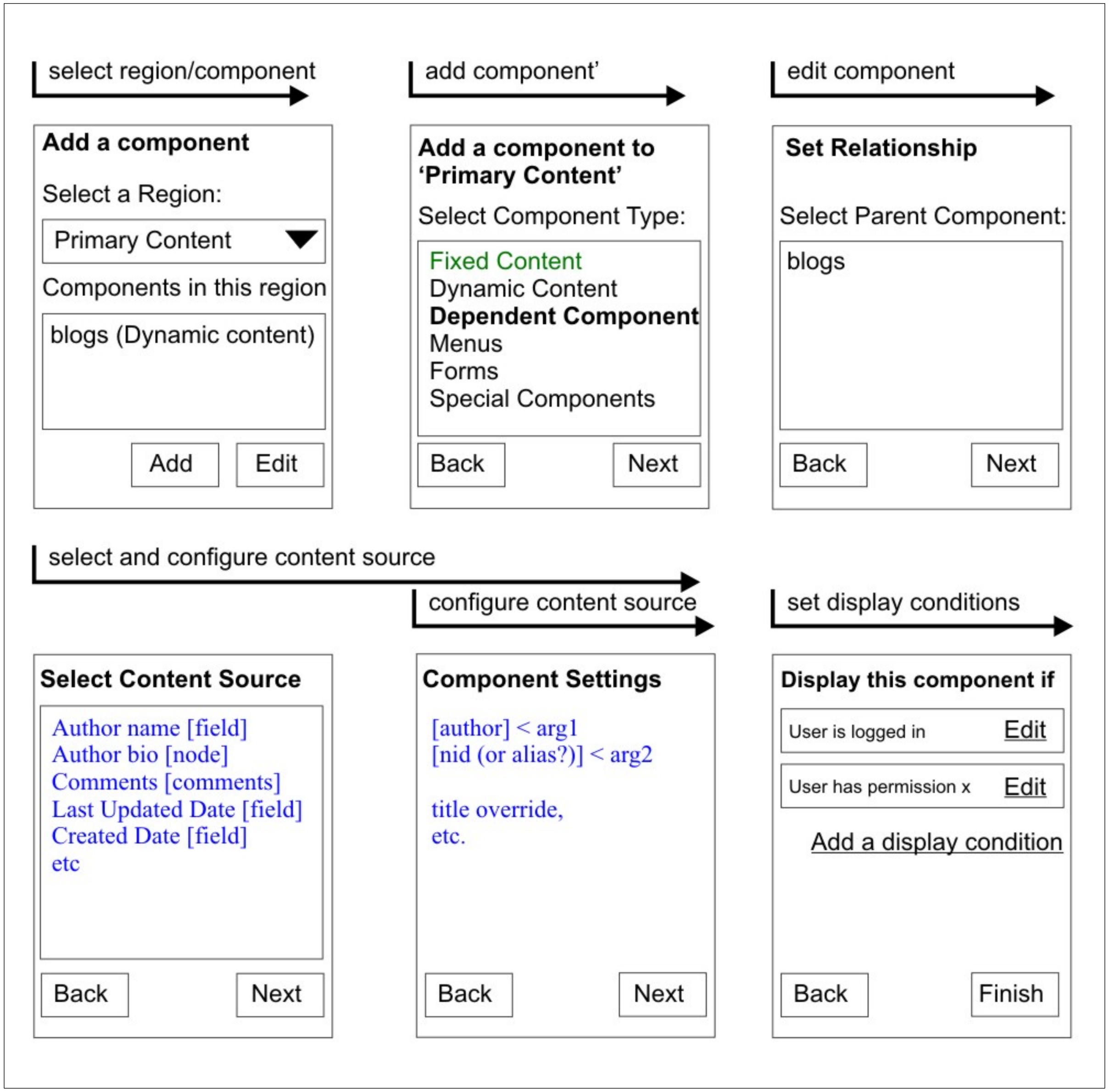
The techniques for providing smooth and effective workflows will involve some artful balancing of active Role switching, on-screen tools, admin screens, modal dialogs and selection mechanisms. A UI strategy for satisfying both user types will probably have to support both the serial (ease of use for occasional users) and the parallel (efficient interaction for the trained frequent user). Breaking down the configuration process into the steps described offers the possibility that we can achieve this. In general we could approach it with a combination of minimal on-screen tools within the 'in-place' URL context and a series of task specific dialog boxes.

The on-screen tools would invoke the corresponding dialog boxes:

- Select Component (includes region selection)
- Add Component
- Edit Component
- Select/Configure Content Source
- Set Display Conditions

Thee 'tools' need only be links and could be place in a very minimal toolbar either at the side or on top of the page.

The following illustration shows the commands as incoming 'usage vectors' into the workflow. The various dialogs could potentially be used as 'standalone' items for single tasks or they could be connected (via appropriate links) to the next dialog in the logic sequence. In other words, the 'wizard' flow could be either implicit or explicitly enforced.



Add a component

Select a Region: Primary Content ▼

Components in this region

blogs (Dynamic content)

Add a component to 'Primary Content'

Select Component Type:

Fixed Content
Dynamic Content
Dependent Component
Menus
Forms
Special Components

Set Relationship

Select Parent Component:

blogs

Select Content Source

Author name [field]
Author bio [node]
Comments [comments]
Last Updated Date [field]
Created Date [field]
etc

Component Settings

[author] < arg1
[nid (or alias?)] < arg2

title override,
etc.

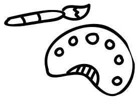
Display this component if

User is logged in Edit

User has permission x Edit

Add a display condition

Contact Info



Michael Keara is a User Interface Systems Architect at Myplanet Digital in Toronto, Canada. He has over 25 years of industry experience in technology design, development and creative problem solving. He can be reached at:

michael@myplanetdigital.com

@useradvocate

User Advocate

(He occasionally blogs on his site at http://www.tuag.ca)