A Proposal for a Customizable, Composable User Interface to Fractal

Edwin Simpson, Michael Rhodes

HP Laboratories
HPL-2009-351

Keyword(s):
user interface content management, map, zooming

Abstract:

Fractal is a cloud-based service for managing and working collaboratively with content. In this report we consider the requirements and design principles for a modular, extensible user interface to Fractal. We also propose a novel user interface style using zooming to access editors for modifying the logic and appearance of the application, and map-like navigation to move between related functions of the application.

External Posting Date: November 6, 2009 [Fulltext] Approved for External Publication
Internal Posting Date: November 6, 2009 [Fulltext]

A Proposal for a Customizable, Composable User Interface to Fractal

Edwin Simpson: edwin.simpson@gmail.com
Michael Rhodes: mike.rhodes@gmail.com

Abstract

Fractal is a cloud-based service for managing and working collaboratively with content. In this report we consider the requirements and design principles for a modular, extensible user interface to Fractal. We also propose a novel user interface style using zooming to access editors for modifying the logic and appearance of the application, and map-like navigation to move between related functions of the application.

Introduction

We described our vision for Fractal in [1], where users coordinate content-related tasks such as editing, sharing and processing around collaborative workspaces called **Content Spaces** [2]. Content spaces will be used for a range of different activities that require a variety of tools for browsing and displaying content, visualizing the results of content analysis, tracking activities of team members and using services to carry out work and retrieve information. Fractal provides a model for composing and automating the use of these services called **active behaviors** [3]. Active behaviors can be created or modified by end users as they go about their tasks.

The variation in applications and tools for content spaces demands a flexible user interface where the addition of new tools and modification of active behaviors is intuitive to non-experts. This report considers a number of motivations and principles then sets out the proposal for a modular user interface design.

Design Motivations and Principles

These motivations and principles originate either from requirements set out elsewhere in the design of Fractal, or are principles that we believe will help develop an effective user interface. Many of these motivations and principles stem from the idea that users can develop, add or publish customizations to Fractal, packaged as single extensions [4]. The extensions can then be placed together in a content space to produce a content space that fills the role of a bespoke application. The user interface should therefore satisfy the points set out as follow.

Self-contained tools: users can add self-contained user interface components called *tools* to their content space, enabling them to add new functionality over time.

Arrangements of tools: several tools can be arranged in the same browser window for complex interaction with content. Users may modify these arrangements when they add new tools, and bookmark them for quick access.

Bindings: tools can be attached to *subspaces* or all content - subsets of a content space defined by a query or list of items – taking the subspaces as inputs to produce a view. Tools can also be bound to active behaviors or services to provide them with input parameters and trigger their execution. A single tool may have multiple bindings. For example, a search box can be bound to three search behaviors that search different indexes, and displays results from all engines in one list.

Users must be able to bind new data sources graphically, without modifying the definition of the tool itself. Inputs to a tool should implicitly take multiple bindings. This allows users to quickly add new search indexes to the search box, for instance.

Different tools will require different types of data: for instance, a chat tool can display short pieces of text with a date-time stamp. It must be easy to attach tools to new data sources of the right type, so they can be bound to a query on a content space, which includes data produced by an active behavior. They can also be connected easily to external data sources via a query on an RSS feed.

Pervasive context: automatically highlight related tools to the currently-selected item. When users are working with multiple tools, when they make selections or choices in one tool, they want the other tools to also use that context.

Map of a Content Space: tools, content and subspaces should all form part of a navigable map that allows user to select content and apply tools to it by following a common pattern. The map should guide the user from a broad view of the content space to the specific content and tools they need to use. This is motivated by the desire for a content-centric platform, where the focus is not on tools and navigation between tools, but on content and actions on content.

One Map Only: all objects in the content space are laid out on a single map. Alternative designs could place subspaces in their own maps. However, we prefer to lay out the subspaces on a single map so users can organize subspaces by arranging them in the map, navigate between subspaces and drag items between subspaces. Separating subspaces into different maps makes them feel like separate content spaces: a content space should feel like one physical space. If people treat subspaces like separate content spaces, there will be security implications that could cause problems. However, we acknowledge there are cases when a large number of users need to use different tools on the same content, and may thus need to ensure they can focus on the relevant tools, and may sometimes need to make those tools completely private. Users may also want to test out modifications to the space before saving them for other members to see. Our design should allow these possibilities within a single map.

Physical Feel: each part of a content space a should physical feel with a constant location, so users can easily relate high level objects to their inner workings, and can memorize the relationships between each thing in the space.

Pervasive Visual Concepts: the interface will use similar visual concepts for more detailed editing tasks, such as editing active behaviors or *pipes.* The same representation can be used for processing elements, inputs, outputs, filters etc.

Inline and Online Editing: users will want to access various uses of tools, active behaviors and services as part of their daily work. In other cases, they want to make modifications with a full understanding of how they will affect the content space, without leaving the user interface they are comfortable with. It is therefore important to make certain modifications accessible without entering a separate editor (inline editing) and to show changes immediately (online editing), for example by displaying the results of a pipe that is being edited.

What should users be able to customize? Different users will have different needs for customizing the space, and at different times. End users will mainly want to organize tools and items in their space so that related things are grouped together and can be viewed together. They will also add new tools either to the space or to process a specific set of items in a subspace, or the set of items currently displayed by another tool. On occasion they may also change settings on active behaviors that run automatically, for example, a behavior that creates backups on Amazon S3 could be switched from running weekly to nightly.

More technical users may find that tools and behaviors do not work in the way they want: behaviors are not responding to the right events, or operating over the wrong subspace, or some additional processing steps need to be added, meaning a pipe or active behavior needs modification. These points must be customizable.

Other technically-minded users and third parties who wish to market extensions to Fractal will want to create tools, behaviors and pipes from basic components. This is a less everyday-task and the work they do may not be closely related to a particular content space. Finally, users with the most expert knowledge will create their own components. Since components may typically be used in a range of different pipes for different purposes, a particular content space is not usually needed as context when developing components, so this type of editing can be further removed from everyday tasks in the UI.

Palette of Widgets: developers should also be able to create tools from a palette of elementary widgets such as lists, buttons, and document thumbnails. They then connect the inputs to widgets to those of the tool.

Connecting: tools can be connected to one another so that an action in one tool affects the display in another. For example, the set of items displayed by one tool could be used as an input to filter the items displayed in another tool. This means that users interact with a set of related tools rather than a single tool in isolation.

Merging: to simplify each view of the content space, it should be possible to avoid having multiple panels that serve the same purpose from the user's point of view. For example, four or five different news feed panels. In some cases it may be desirable to show these as separate panels, but in many cases users want a single list of interesting news as this is easier to navigate, and prevents the large number of news feed panels from taking up a disproportionate amount of space. The user must therefore be able to merge the news panels into a single list.

Interface Types: to create a clear yet flexible UI, a single browser window should show tools that are complementary, serving different purposes and showing different kinds of information. The aim is that merging tools will mean that most browser windows show only one or two tools of the same *interface type*. High level interface types include notification

lists, content viewers, navigation list (links, search results), inputs for modifying what is displayed (e.g. search box). This principle will shape the layout and design of the user interface.

Considering the principles above, the following sections explain the idea of tools in more depth, then discuss the key parts of a content space user interface:

- the *content space map* for navigating between different elements in the content space;
- a *content browser* for finding or selecting content to work on;
- the *toolbox* for accessing tools from any location;
- the *landing page tools,* the view the user sees when they arrive at a content space;
- *zones,* for organizing groups of related tools.

Tools

Tools are modular pieces of functionality that provide a user interface. Most features integrated into Fractal for viewing, manipulating, processing and interacting with items will be provided by tools. Any self-contained part of the user interface that provides a separable piece of functionality could be implemented as a tool. Other parts of the web application will provide basic functionality for navigating between tools, installing new tools, and viewing a list of items in a space and possibly also for modifying the members of a space. In most cases it is logical not to think of these parts as tools but as basic, standard features, although they too could also be modular.

Tools can work in various ways. Usually, tools contain a UI template including some JavaScript and some variables that are set by the platform when the tool is viewed. Some variables can take subspaces as their values, so the template can operate over the content of the subspace. In this way, the template can be bound to a subspace. Tools may also consist of pipes, active behaviors and internal subspaces. Inside a tool, an active behavior may produce data in response to events, and store it in a subspace. The UI template may then be bound to the subspace. The UI may also be bound to a pipe, so that when the user makes a specific request, such as viewing the tool or pressing a button in a tool, the pipe is executed. The UI template itself could be generated by a pipe acting as a URL handler or an active behavior responding to other events. In summary, a tool specifies a user interface that can be passed subspaces to process, and its appearance and logic may be implemented using active behaviors and pipes acting as URL handlers.

Easy navigation between tools is important to users. They may switch tools to switch to a new task involving a new set of items; they may switch tools to perform a different task on the same item or items. Some tools must be used in the context of a single item, a subspace or the content space. Certain tools may also be used over a period of time, storing state between uses, while others may have no state or may be used on a one-off basis.

Content Space Map

The logical map defines a pattern of relationships used to navigate between content, subspaces, pages, tools and active behaviors. It also defines their relative locations. When interacting with a content space, the browser will show part of this map. Users navigate by zooming in on different items and panning from left to right. Zooming is used to reveal a greater level of detail about an object, while panning is used to move between different parts

of the content space at the same level of detail. Objects in the map can therefore be placed close to one another so they can be viewed concurrently, or arranged on the map to allow easy navigation between them. The map therefore provides a single layout for working with and customizing the content space.

In some cases users who are zoomed in on an object in the map can follow its connectors to different parts of the map. Connectors represent relationships between objects. Reliance on following connectors should be avoided, however, so we can implant the spatial layout of the content space in users' minds to make it simple to navigate.

Links in the Map

To derive the logical map, we first consider the elements in the system and their how one would want to navigate from one to the other. This is shown in the diagram below, where *content* represents the pool of content in the space, and *subspaces* specify selections of this content. In the UI, the Content and Subspace objects are not expected to provide any advanced or specialized functionality themselves, such as Faceted Browsing. Rather, they are intended to represent the content or a subspace, showing its connections to related objects such as tools.

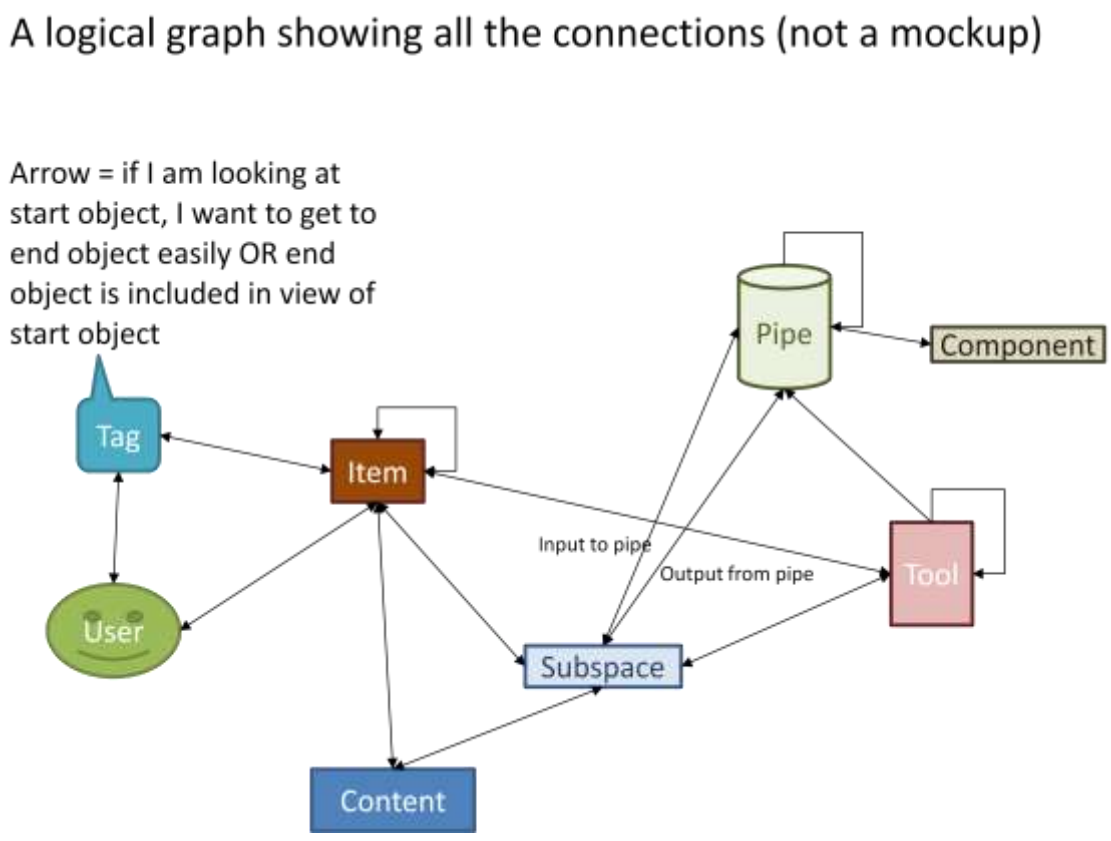

Figure 1: Graph of connections which represent both navigational links and logical relationships between parts of a content space.

The links in the diagram indicate that users can navigate from one object to another. Users will use different links for different activities, as described in the table below.

Navigation Link	Use Case	Activity

Content → Subspace	browsing content organization, finding the tools you want to modify	Browsing Items
Subspace → Content	modify the subspace, create a new subspace	Organizing Content, Customizing and Adding Tools
Subspace → Subspace	Browsing content organization, finding the tools you want to modify	Browsing Items, Customizing and Adding Tools
Subspace → Pipe	What pipes are creating content in this space? What behaviors are processing or managing this content?	Customizing and Adding Tools
Subspace → Tool	view this content in a different way; modify this content	Customizing and Adding Tools, Editing Behaviors, Working with Different Tools
Pipe → Component and Pipe → Pipe	see how the pipe works, modify it	Editing Pipes
Pipe → Subspace	see the outputs of the pipe; adjust the inputs to the pipe	Editing Behaviors
Component → Pipe	Where is this component used?	Editing Pipes
Tool → Pipe	see the pipes that create the content shown in the tool; see the pipes that are triggered by actions in the tool	Editing Behaviors
Tool → Tool	alternative views of the same content; alternative tools for modifying it; switch between viewers and actions on the content; move to a whole new task with a new set of content	Working with Different Tools
Tool → Subspace	see how the tools works, where the content it displays comes from	Customizing and Adding Tools
Tool → Zone	move from one tool to see other related elements in the same browser window	Working with Different Tools
Zone → Tool, Zone → Subspace, Zone → Content	show several tools and subspaces in the same view	Working with Different Tools
Content → Zone	see the top-level and most common views for the whole content space	Working with Different Tools

Tool → Item, Subspace → Item, Content → Item, User → Item, Tag → Item	preview or open the item; see list of related items; see list of related tags, users, subspaces and tools	Working with Different Tools, Browsing Items
Item → User, Item → Tag	find other items or information related to this user/tag	Browsing Items
Item → Tool	view this item using this tool	Working with Different Tools
Item → Subspace, Item → Content	see what else is in the subspace, see what's connected to the subspace, see what subspaces are related to the item	Browsing Items, Working with Different Tools
Zone → Zone	switch between various collections of tools	Working with Different Tools

To create a simple spatial layout for the content space, it is not possible to display all links at the same time. Links corresponding to the same activity should all be easily accessible when carrying out that activity. Other links need not be accessible from that activity, so to simplify the user interface we will instead provide a way to switch between different activities. In a spatial layout, each activity could correspond to a plane in a 3-D map. The activities can be ordered starting from the highest level, everyday activity to the low-level activities that will be the focus of more technical users and developers.

Figure 2: Different physical planes for different tasks or modes of working, namely pivoting between content and users in the item links plans, accessing and using tools in the work plane, and modifying the underlying logic in the content space in the behavior editing plane.

Working with Different Tools and Browsing Items are two activities that users will carry out every day, and where individual tools will provide most of the functionality. Working with Different Tools and Browsing Items are mainly separable tasks that require a different set of links. Browsing Items may require a large number of links that follow a totally different structure to the connections between tools, pipes and subspaces. Therefore, links for working with different tools will reside in the ***work plane***, while links for browsing items should be separated and may live in an ***item links plane***.

Customizing tools involves making simple modifications to how tools work and the set of content they operate on. Users should be able to grab a handful of items being displayed by a tool and throw them into another tool, as long as the items are of suitable type. The intention is for the second tool to process just these items. The user should also be able to make this a

permanent connection: if new items appear in the first tool, they too will be processed by the second. These are tasks that users must be able to carry out without understanding how tools work. In addition, the same links are also used for Working with Different Tools, so this may be kept on the work plane.

Editing Behaviors is a more complex activity for more technical users. This activity is unlikely to happen during normal, everyday work, so can be separated into a lower level ***behavior editing plane***. The links in the editing behaviors plane can be revealed by zooming in on objects in the work plane. Here, users can modify the links and items such as subspaces, UI templates, pipes and components. They can create new connections, modify old connections, modify subspace definitions and set configuration parameters, e.g. for events.

The diagram below shows how the work plane and behavior plane may be laid out.

The Logical Map

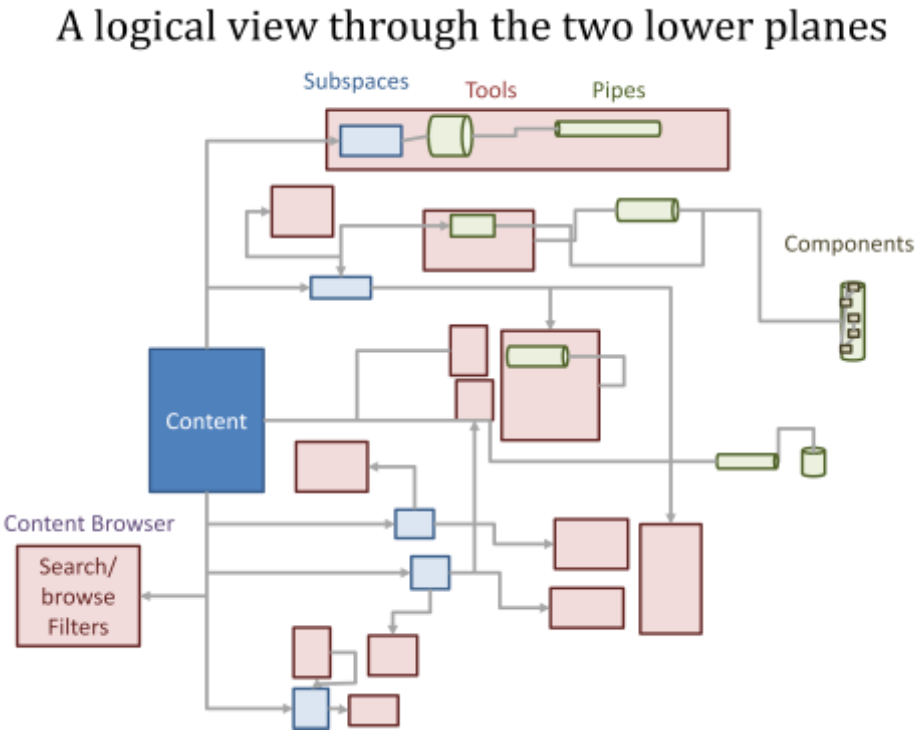

Figure 3: shows a zoomed-out view of the content space, showing the layout a user would see. The diagram differs from the user interface as it represents most objects as boxes with no details inside, and exposes components, pipes and tools in the same view to show their relationships.

The dark blue “content” box would be presented as a view containing thumbnails or a list of documents. Zooming in on a single thumbnail would lead to a document preview. Associated with Content are user-created subspaces, which may also show the items they contain. Subspaces are similar to playlists or queries, and items should seem to reside inside Content rather than the subspaces.

Alternative views of the content in a subspace are provided by tools. It is preferable to use tools to create new views of subspace contents rather than customizing subspaces themselves, as several combinations and arrangements of tools can be created, and subspaces have a consistent representation and location.

To separate out different tools and items relating to different tasks or users within a content space, users can arrange tools around subspaces in a particular area. That area of the map then focuses on a particular task, or provides the tools needed for a particular user. Testing in real scenarios would help us determine whether a large number of these different areas are created. If many different subspaces and areas are created, the map may need to expand to accommodate them. A question remains over whether the tools themselves should have permissions, so users can create private areas of the map for their own tasks.

Tools – the user interface parts that users can add – can be connected to subspaces or to Content, taking documents in a subspace as input and in some cases adding documents to another subspace as output. The grey lines here represent these bindings, which can be modified by users. Search and browse are examples of tools. Tools can be viewed individually so that they fill the browser window, revealing more details.

Tools can also be grouped together. What is shown in the browser is dependent on zoom and pan, so users arrange related tools close together so they can zoom in on one area of the map. If the user selects a number of tools at the same time, they can click an option to automatically group them. The user can choose to either rearrange the tools close together, or to duplicate them and move the copies close together. The latter option prevents users from disturbing existing arrangements of tools.

Tools may be implemented using internal pipes and subspaces. Internal subspaces filter can be used to filter the tool's input items before passing them to a pipe. As they go about their everyday business, users will rarely need to encounter these internal components, but some users will be able to zoom in on the tools to modify how they work – this is the behavior editing plane. Zooming in on a pipe reveals the individual components and allows the user to modify the pipe itself. Making subspaces and pipes internal to a tool reduces the need for connectors crisscrossing the map.

Earlier we stated that navigation between tools is of critical importance. The layout above gives users a strong logical idea of how tools are bound and implemented, and fixes them in a consistent location. However, it does not directly provide methods for quickly moving between locations. This suggests the need for some standard functionality that is available at all times for jumping to other tools. This feature would need to make it easy to jump to the most relevant tools from a given context. For example, when looking at a particular type of item, tools that display that item are shown. There may be other features such as this that are needed at different locations in the map. Therefore, we propose creating a toolbox of tools that are applicable from any location, and fixing this toolbox to the same place in the browser window. Please see the Toolbox section of this report for the full proposal.

Active Behaviors do not have to be contained within a tool, so can be visible alongside tools. However, as they are not the main focus of activity when using tools, they appear much smaller. Users can then zoom in to edit them. In a similar way, connections between objects on the map will also be made less prominent when the user is working with tools and items, and will become more exposed when they zoom in or select one of the connections.

Mockup of the Map UI

The Map Container

While a view onto part of the map makes up the major part of any browser window, a *container* around this enables users to zoom and pan by clicking on a frame around the map. The container may also show a trail of breadcrumbs representing the path across the map so the user can quickly jump to previous places.

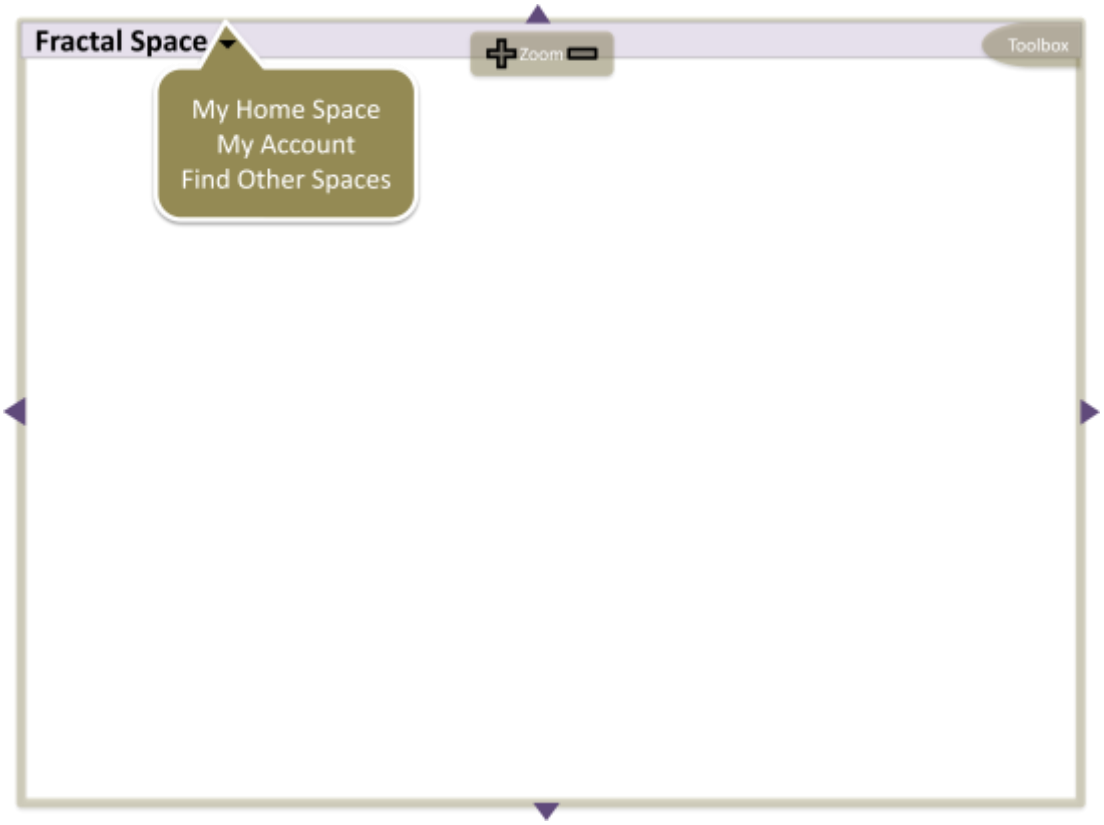

Figure 4: a view of the user interface showing only the Container

Top Level View

Based on the logical map, the mockup below shows the zoomed-out view of a content space within the container. Different content spaces would look very different at this view, as the tools they contain can be tailored to different users, teams, industries and jobs.

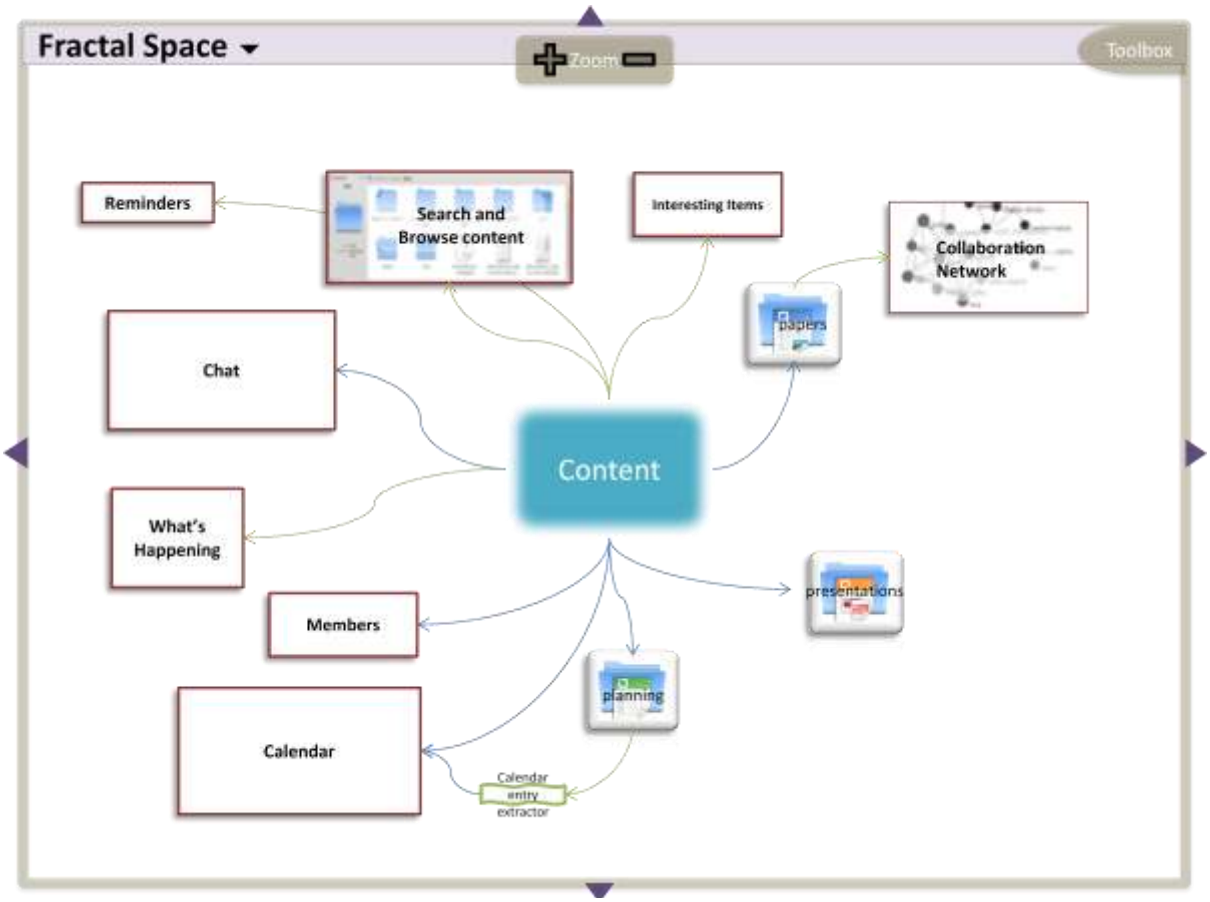

Figure 5: mockup of a zoomed out view of a content space.

Zooming In on the Map UI

In general, double-clicking will centre and zoom the map on the area you clicked, as it would in a geographical map application (e.g. Google Maps). Double-clicking on objects in the space will cause the map to focus on the selected object, so the map may zoom to different levels. Double-clicking on a tool may zoom so that the tool fills the map window. Where there is whitespace between several tools, double-clicking on that whitespace may zoom the map on the surrounding tools, so those tools fill the window.

Single-clicking on an object in the map would typically highlight all the connections related to that object.

Zooming in on a Subspace

Users often view a subspace to find out what tools are used to interact with it, or what pipes are used to modify its contents. Therefore, selecting and zooming in on a subspace first zooms in on the connected pipes and tools.

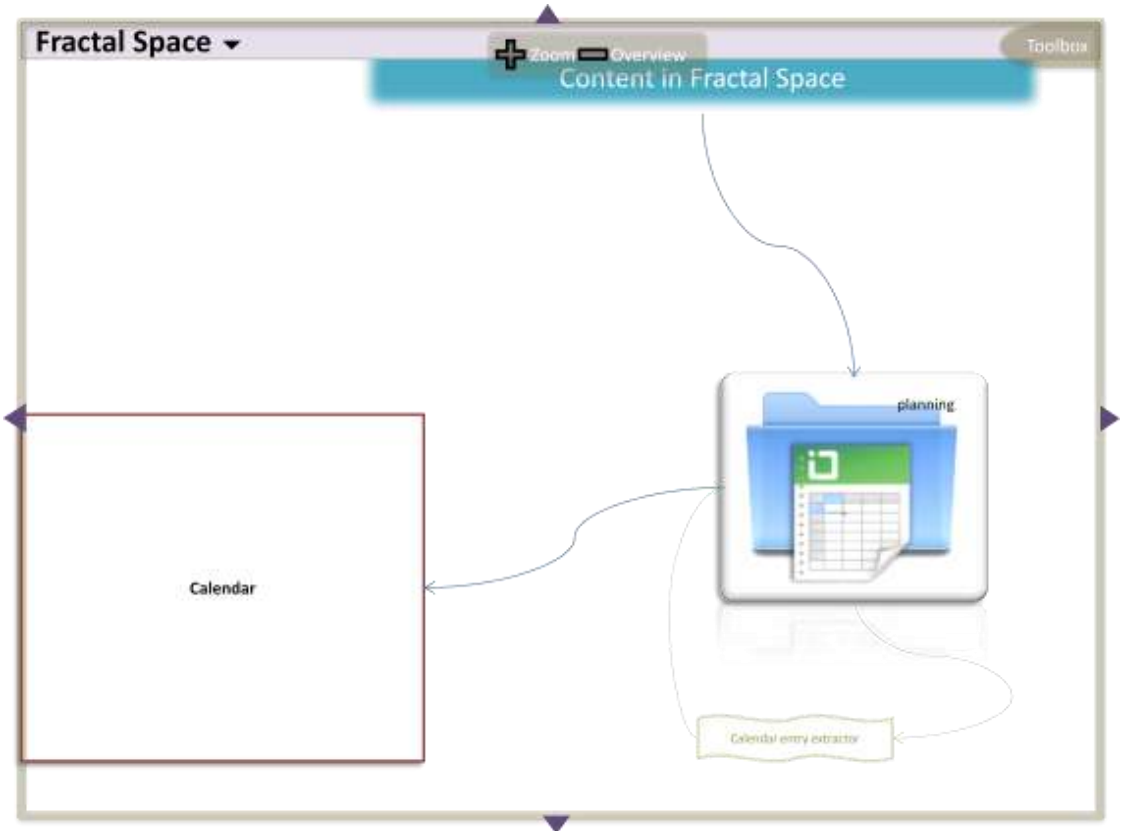

Figure 6: zooming in on the subspace initially shows related pipes and tools

Zooming further causes the subspace to fill the view.

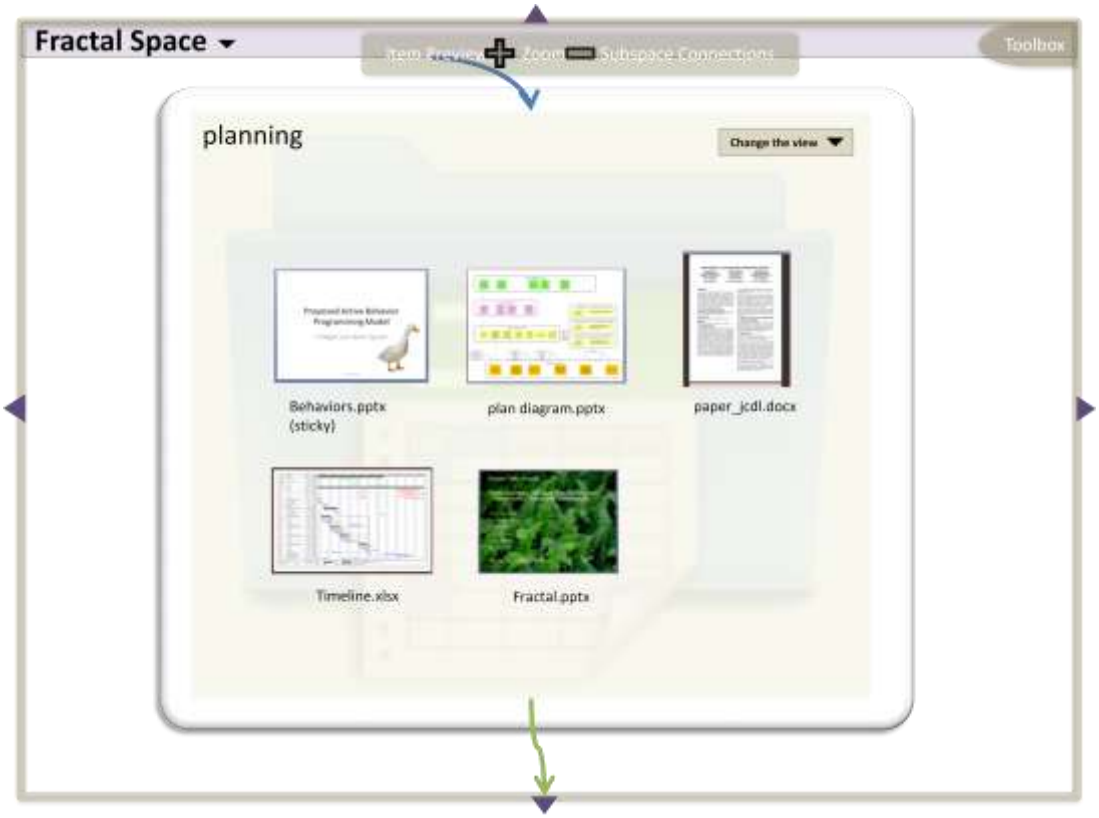

Figure 7: zooming further on the subspace reveals items inside it.

The subspace representation may be fairly fixed to clarify the concept of a subspace and make this appear consistent. The representation would provide simple ways to modify the contents of the subspace. Alternative views will be added by coupling new tools to the subspace.

Zooming further on an individual item may open the item in a previewer. Tools are associated with item types as previewers using a Preview Manger Tool, which is installed in new content spaces by default.

Item Links

Tools and subspaces in Fractal will have access to a standard representation of an item. This representation could provide thumbnail previews for certain item types. The item representation will also provide a popup containing links for easy pivoting to related items, tools and subspaces. In this way, users will be able to quickly find related content without having to open a separate tool.

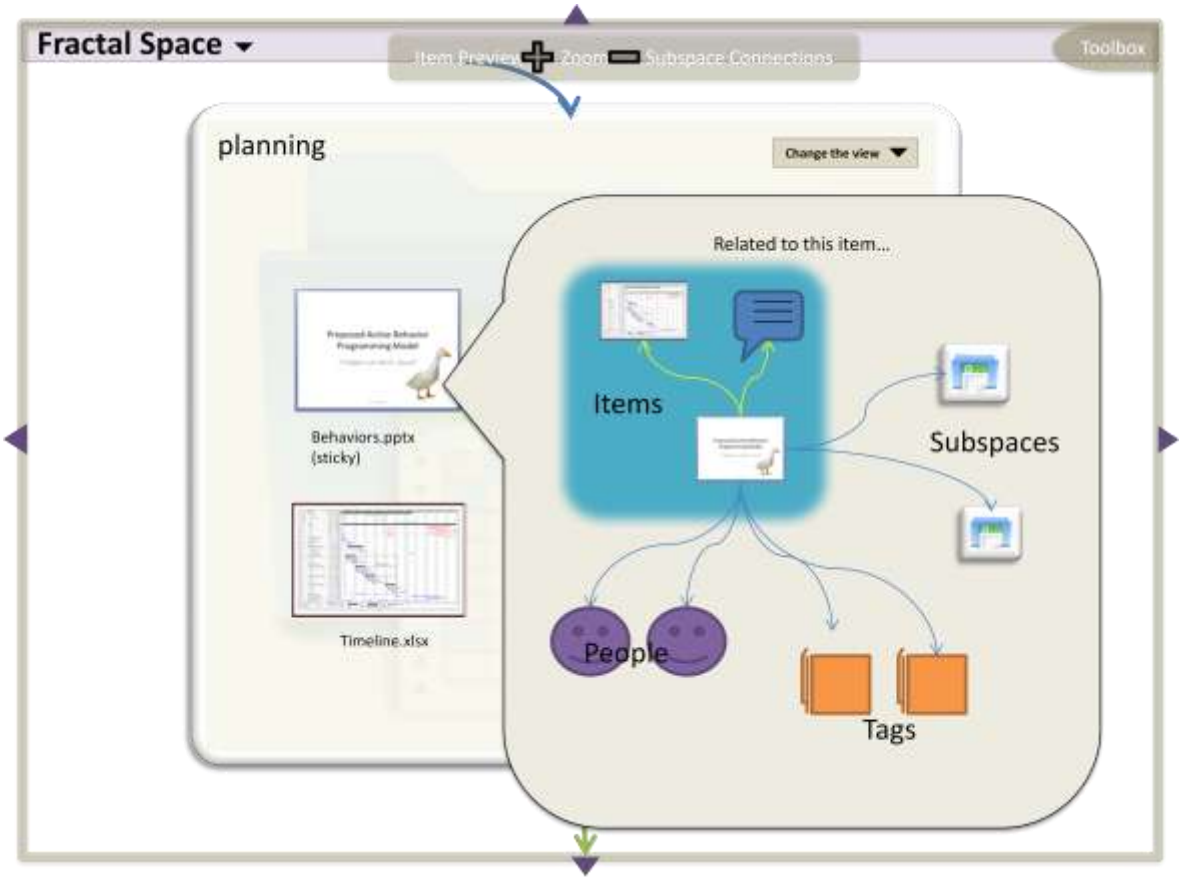

Figure 8: Exposing the top level item links plane from an item representation.

These links form the Item Links plane mentioned earlier, which is the top layer of links. Items can be linked to other subspaces, entities such as authors, other content items, tags and tools. Authors and tags may be represented by *implicit subspaces*, which contain all items with a given tag or author. These subspaces need not be declared explicitly be the user, but are generated and linked automatically when a new author appears in the space or a new tag is used on an item. Tools can create relationships to items, for example if they are used to author the item. Single-clicking an item in the item links popup would re-centre the popup on

the selected item. Double-clicking would move the main view to focus on the selected item, allowing the user to switch to working on different items or tools.

Dealing with Complex Maps

There are several strategies that could help users work with complex maps. These ideas may be incorporated into the UI if early testing suggests they are required.

Highlighting Relevant Objects: when looking at a zoomed out view with multiple objects, the user can select one object. The map will then highlight connected objects, dimming out others.

Following Links: links that leave the current browser window can be clicked on to follow to their end points.

Lookahead Thumbnails for Links: where we have connectors leaving the current browser window, a thumbnail is shown of the object at the end. The thumbnail can be clicked to move to the object. If the map is correctly animated – which is the challenge with this approach -, this maintains the idea of a physical location for each object.

Duplicating Tools: tools can be duplicated easily so that if you need to create a new arrangement of several different from different locations, you don't have to move the original tools. Instead you can copy them and move the copies together.

Expandable Areas: areas of the map can be enlarged (manually or automatically) when new tools and subspaces are added. This may disrupt the maps slightly but allows new tools to be placed alongside other tools without significantly disturbing the existing layout.

Paths: When navigating a map users may often want to return to the previous location and zoom level. Our map layout makes it possible to record the navigational path from the blue content box to any object in the map. For instance, the path to a pipe may take the form "contentspaceID/subspaces/subspaceID/pipes/pipeID".

Content Browser

Fractal will provide new content spaces with a number of generic tools for working with content. Designing these individual tools as well as the customizable map is important in helping users to get started with a new content space, and creating a strong basic set of tools that developers outside the Fractal team can build upon. The content browser is an example of a generic tool that would be useful in many content spaces with a large number of content items.

Users require the content browser in a number of situations: they want to find a specific document; they want to see documents relating to a subject, person or other; or to get an overview of what documents are in the space, their topics and relationships. Finding documents and creating subspaces is also important for using content processing tools. Thus we need a way for users to quickly express a specific query, or build up a query in several steps, and a way to explore the subjects, authors and other object related to documents. A powerful tool for browser large collections of content is a faceted browser and search engine

that helps a user build up a complex query over content and metadata. The browser could help users find...

- Documents, displayed in a manner that makes it easy to discover relationships to authors, subspaces and other tasks
- Other objects the user can manipulate, including subspaces and tools
- Other entities that help the user to browse, such as related users and tags

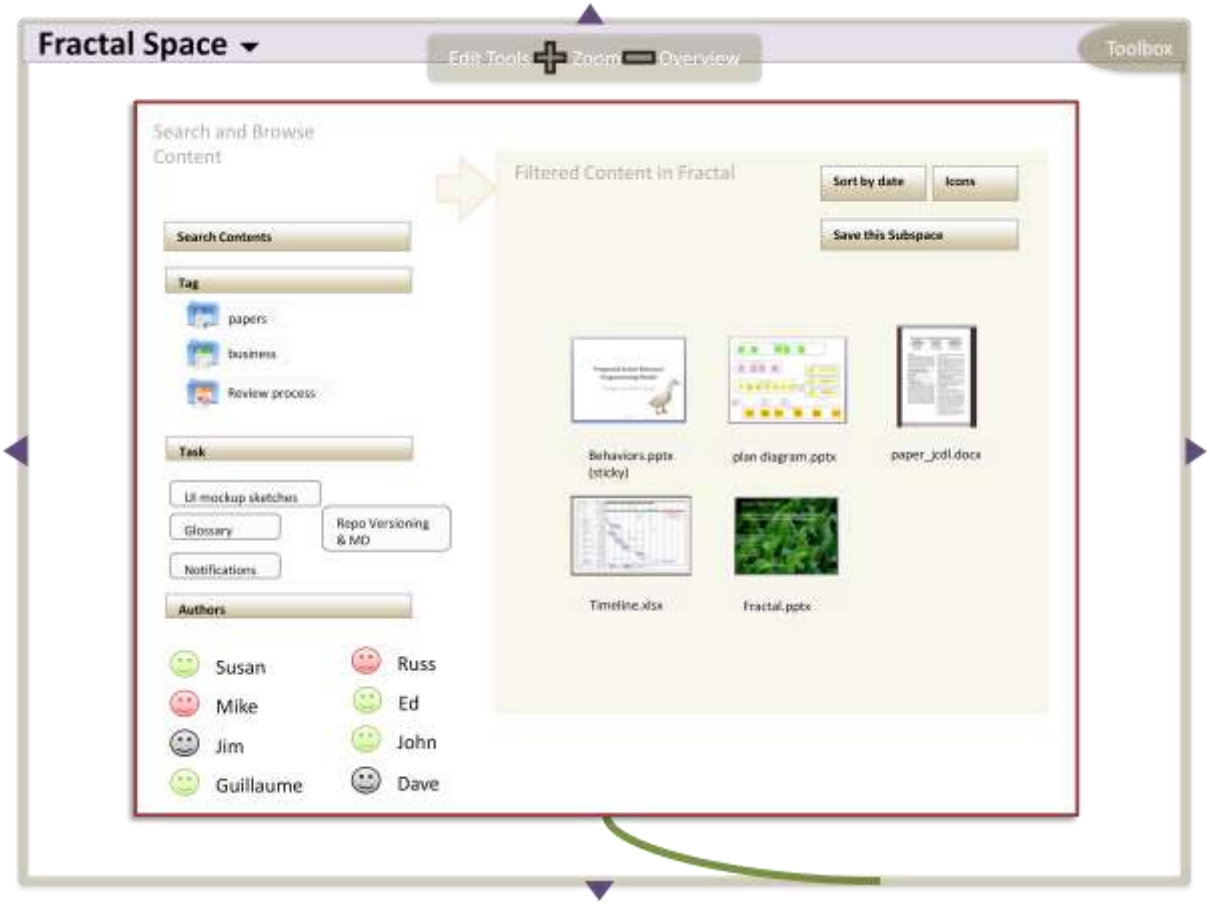

Figure 9: a mockup of the content browser showing faceted browse features on the left.

The figure above shows how a faceted content browser would appear. When displaying a set of items in the main pane, item links like those available in subspaces could be provided. However, the user would typically use the filters on the left hand side to constrain their search.

Hovering over an item in the faceted browser highlights related values in the left-hand-side. The item links popup is also available, and behaves in the same way as the item links popup in a subspace.

Users can click a button to create a new subspace from the set of items currently shown. The subspace will either be a manual subspace, listing the current set of items explicitly, or a smart subspace, where the query is used to define the subspace.

Toolbox

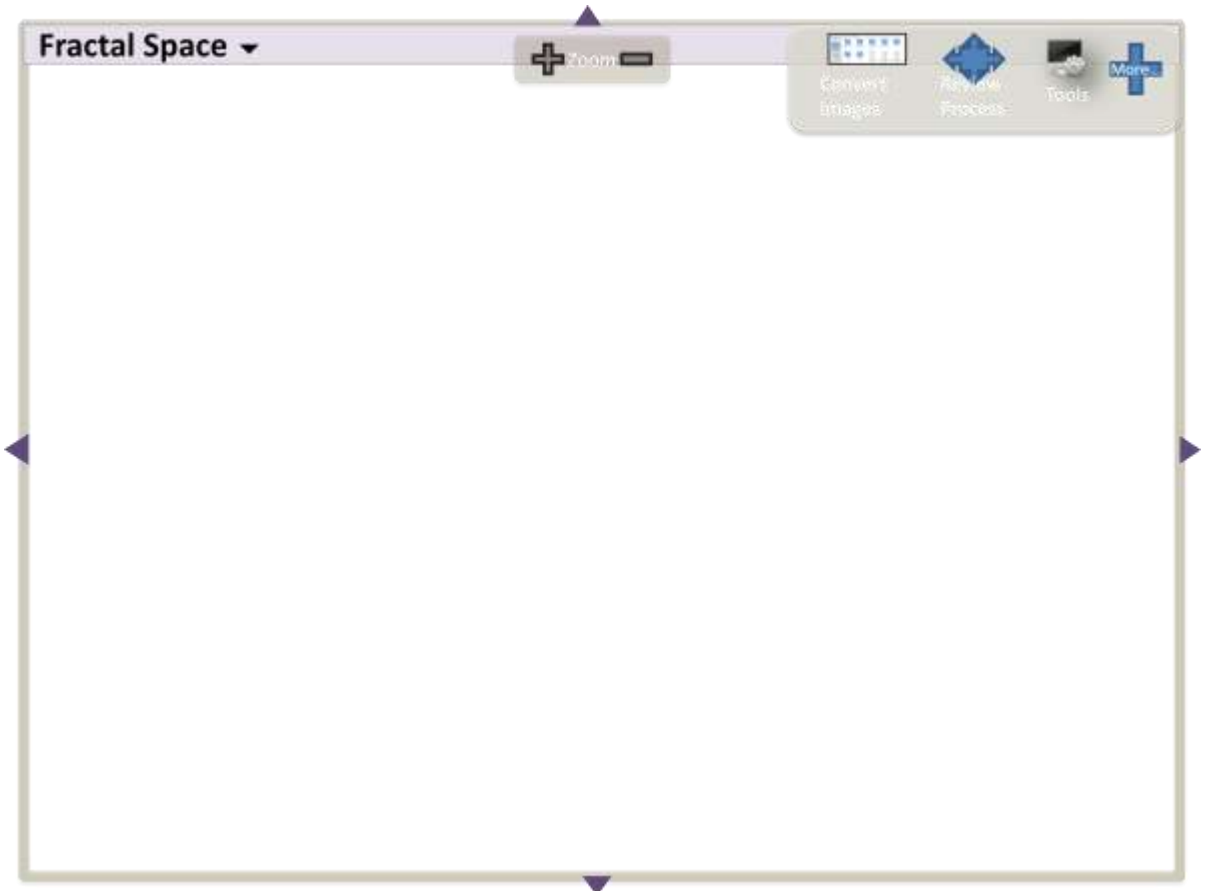

Figure 10: The toolbox on the top right of the container.

In the top-right corner of the container is a palette of tools that can be used from any place in the map. These include tools that are not currently bound to a subspace and may be run on a one-off basis, such as a document review process tool that can run a workflow on a given item shown in the figure below.

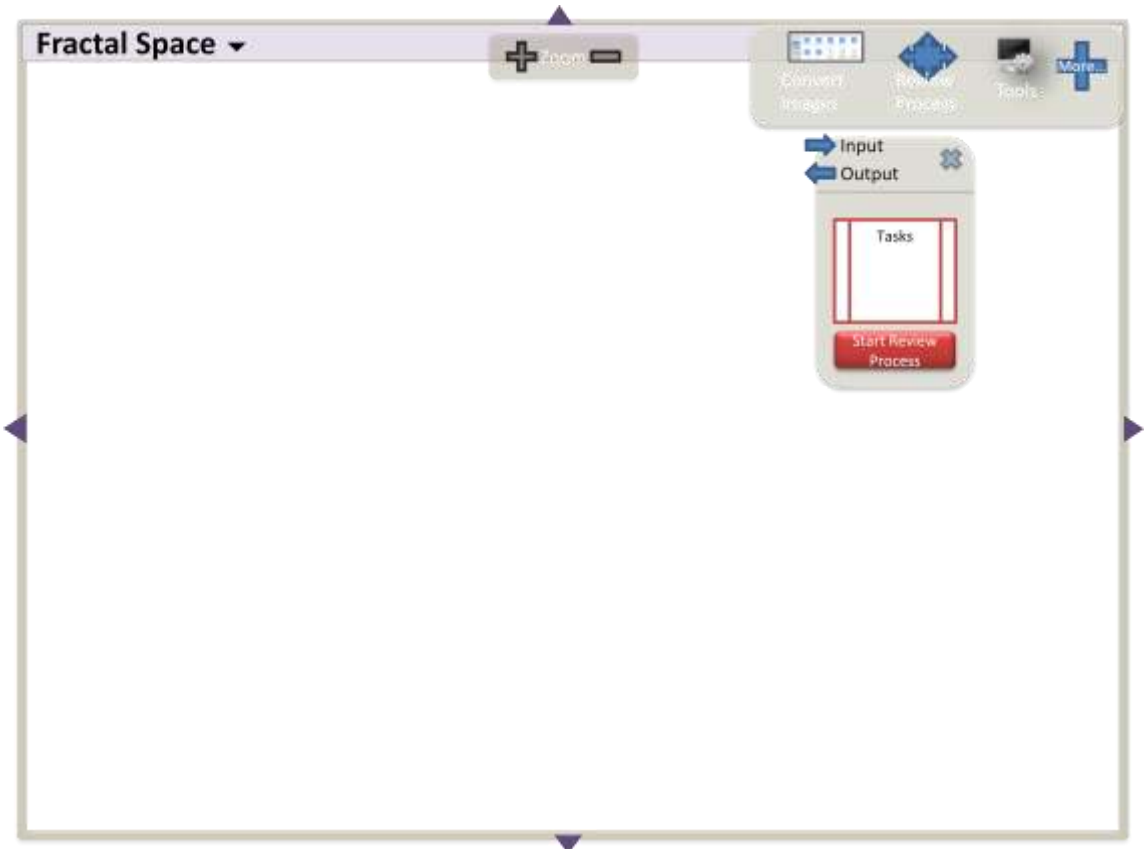

Figure 11: clicking on the Review Process in the Toolbox creates an instance of a review process tool that can be connected to a document or subspace that need reviewing.

Other tools could also be placed in the toolbox for instant access at any location in the content space. The figure below shows an example: a tool that lists all the tools in the space.

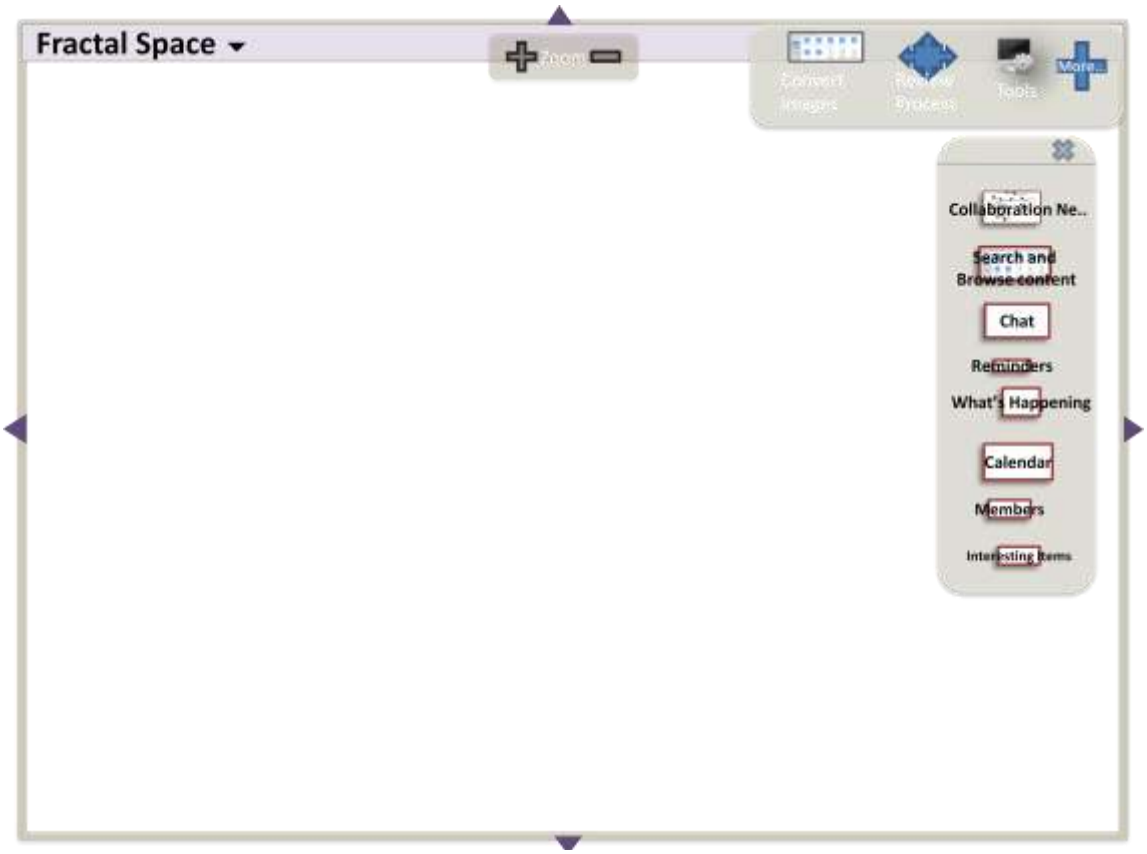

Figure 12: Clicking on "Tools" in the toolbox shows an instance of a tool that lists all other tools. This tool does not need to be connected into the map, and floats above it as you navigate the content space.

When the tools are selected they float above the content space, meaning the location in the map can change and the tool will remain in the same place. The review process tool also exposes input and output pins. These can be bound to documents or subspaces by dragging items to them.

Tools can also be given a fixed location in the space by dragging them from the toolbox to the map. Alternatively, once the pins are bound, and the tool is still floating above the map, the bindings can be made permanent by clicking a link.

To add new tools that are not yet in the toolbox, users click on the plus symbol. This navigates to the extensions marketplace where users can add tools to the map or the toolbox. The contents of the toolbox can change automatically to show the most appropriate tools and pipes in a given context. For instance, when viewing a subspace that contains mainly images, the toolbox may prioritize image-processing tools.

Landing Page Tools

Typically, one would expect a new user logging in to a content space to a see the top-level, zoomed-out view of the map. However, the map is designed for navigating between objects in a space, and users may also want to see useful information or important content as soon as they log in. We therefore wish to include a generic set of landing page tools that are included with new content spaces. Users can bookmark a view of the map zoomed in on a group of landing page tools, then use this bookmark to access the content space. It is also possible that

new users could be directed to this set of tools when they first log into a content space. The tools can be customized, removed or replaced as users see fit.

To design some landing page tools, we start by examining the typical needs of users by answering two questions:

1. What do users want to know about their collaborative project when they go to the landing page?
2. What do users want to do when they go to the landing page?

The answers to 1 are labeled *information requirements* and answers to 2 are labeled *activity requirements.*

Information Requirements

Updates

1. What have the rest of the **team** been doing?
2. What new **content** is there?
3. Are there any communications or important **news** relevant to my project?

Notifications: Tasks I have to do, events, notifications...

1. Are there any **discussions** I should reply to?
2. What **tasks** do I have to do today?
3. When is **event** x happening?

Team

1. **Who** has joined the team so far?
2. **How** are they contributing - just reading, or adding content?

Activity Requirements

My Documents: documents I have been working on.

1. Read or edit the **same** documents as last time I went to the content space.
2. Do some **processing** on a document I was working on.

Content: all content in the space.

1. **Read** a document someone has told me about.
2. Do some **processing** on some content someone told me about.
3. Find some information on **subject** x.
4. See different **views** of a set of documents or data.

Interactions with the team.

1. **Send** a document to the team or one person.

2. **Talk** to someone or the whole team about a document.
3. **Invite** someone else to join the team.

New Tools

1. Connect a new tool to my content space - one that I'm using outside Fractal, or one that I'm not using at all yet.

Proposed Tools

We now propose a set of tools that would give the user the required information or allow them to carry out the tasks described above. The table below shows the requirements a tools satisfies. To satisfy a requirement in (1) the tool or feature must show the information on the landing page. To satisfy a requirement in (2), the tool or feature must enable the user to carry out the task within a couple of clicks.

Tool or feature on the landing page	***Requirement***	***How feature satisfies the requirements***	***No. clicks***
What's Happening	Updates	Shows stream of updates to team membership, content, relevant news, upcoming events, discussions	0-1
	Notifications.discussions, Nofitications.events		0-1
Reminders	Notifications.events, Notifications.tasks	Lists tasks I must do and events I must attend	
Members	Team	Shows the team and graphically represents roles; allows you to click on a person and share content or converse with them; provides a link to invite users	0
	Interactions		2
Interesting Documents	My Documents, Content.processing	Shows recent documents, documents recommended by colleagues	1-2
	Content.views		2
Search, link to content browser	Content.read, Content.subject	Allows you to find a document quickly so you can read or process it by clicking on the document representation	2+
	Content.processing, Content.views		3+
Links to tools in content space	Content.processing, Content.views	Allows you to open tools that manipulate or process content; allows you to open different views of content, e.g. graphs, tables, diagrams	3
	MyDocuments.processing		3
New Extensions Link	New Tools	Allows you to open a search for extensions, or select an external tool to connect to the content space	3+

The “no. clicks” column shows how many clicks are required before performing the task or getting the information required, if the tool were to appear on the landing page. To prevent

information overload when the user sees the landing page, certain requirements have been prioritized and have a lower number of clicks. Features that enable users to view or process content that doesn't appear in the Interesting Documents View have been deferred to a dedicated content browser, so users can first find and select the content.

The image below shows the tools as they might appear on a landing page. *What's Happening*, and *Reminders* and *Members* are labeled in this UI mockup, and the Interesting Documents tool has been labeled simply *Documents*. Search, link to content browser, links to tools in content space and the new extensions link are provided by tools in the toolbox, and are not visible in this diagram. The toolbox always contains a tool for finding and adding new extensions, and will by default contain tools for searching, browsing and listing all tools in the space.

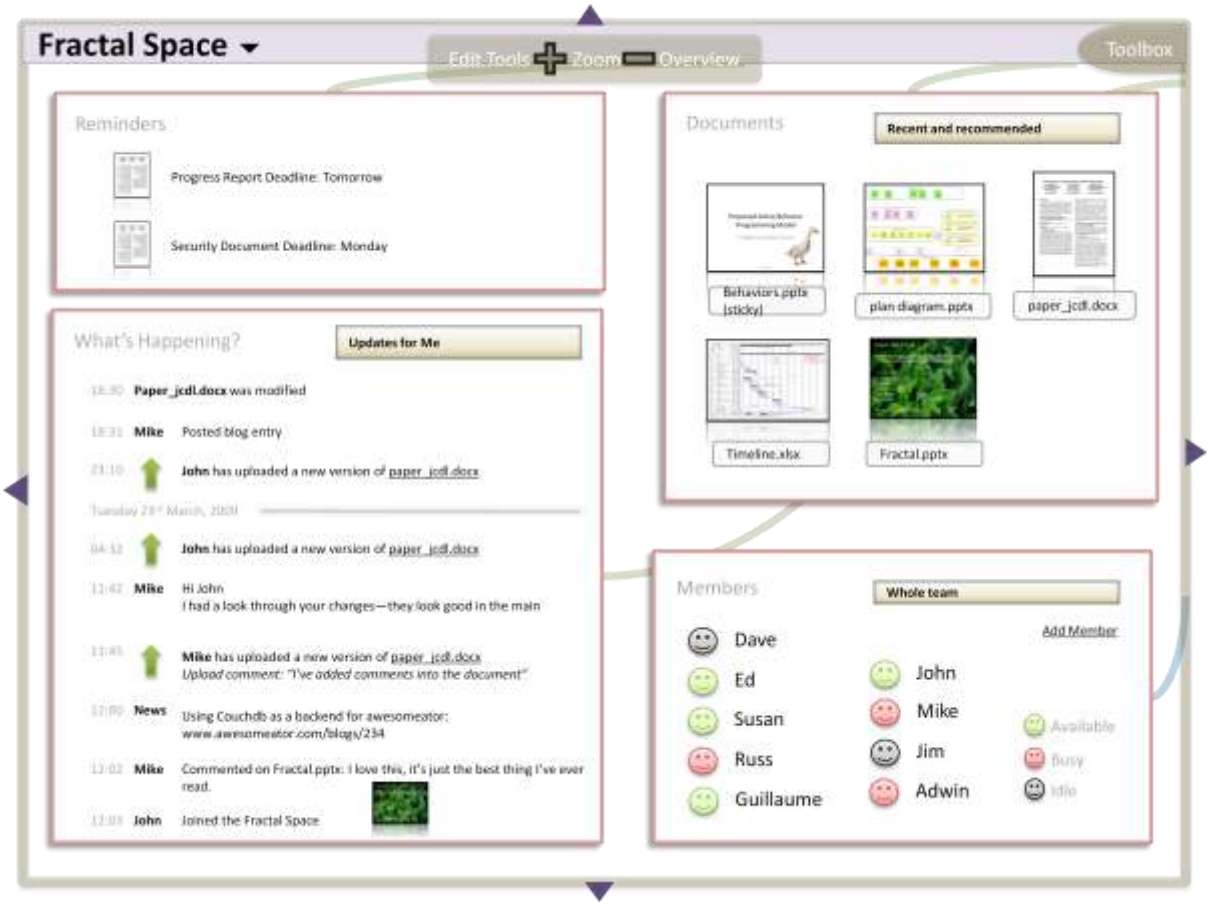

Figure 13: mockup of the landing or page for a content space

Zones

The landing page tools were grouped together in the map so that users can view these tools at the same time. Such groupings should be easy to find and focus on. Above, we suggested that the map will zoom in on a group of tools when you double-click on the space between them. Groups of tools or areas of the map could also be recorded as *zones* to make them easier to identify. Zones record a particular zoom and pan setting used to view an area.

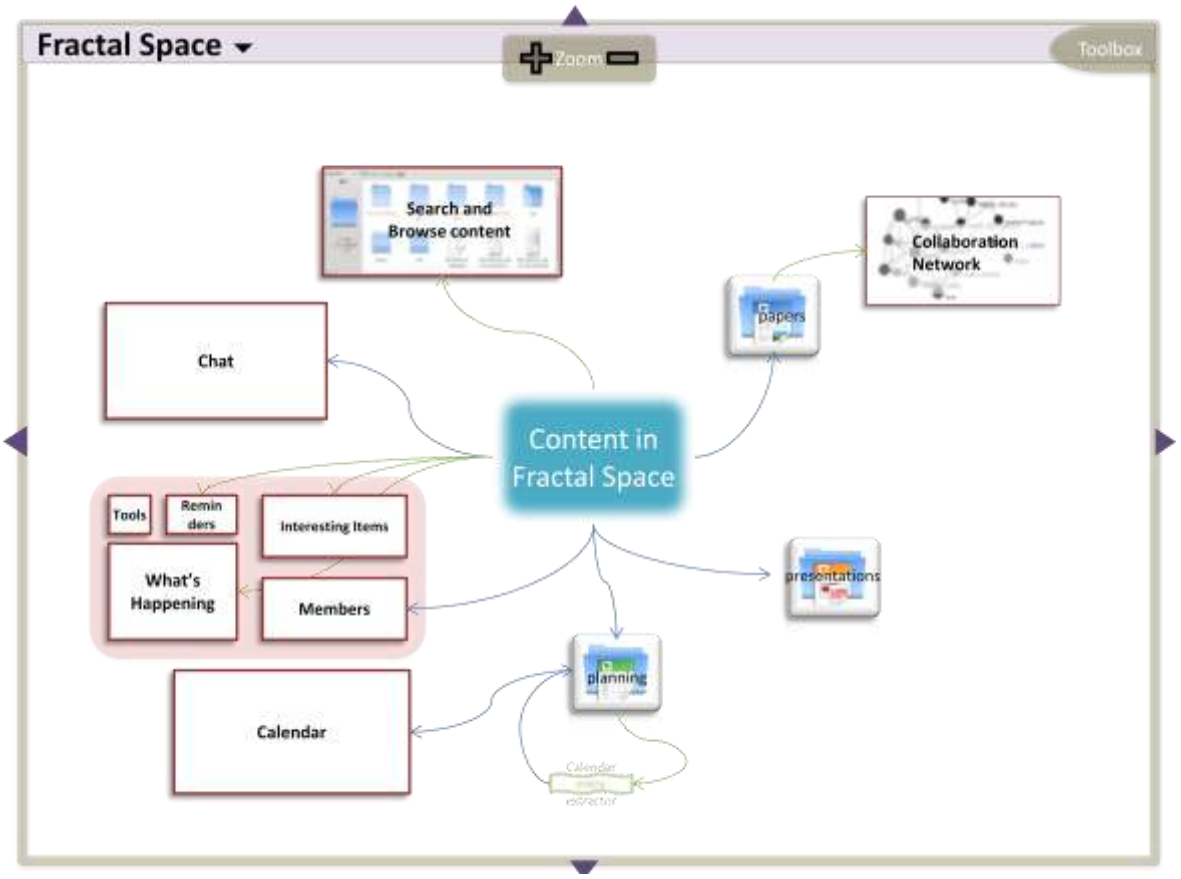

Figure 14: a zone shown as the pale red area on the left of the content space. This zone is used to create the landing page.

Zones can be created by manually marking the map or automatically when rearranging tools. Users would select a set of tools and ask for them to be grouped. A zone would be recorded automatically that the user could then name. A dedicated tool in the toolbox may be used to manage or list zones. It is possible that they could be entirely provided by a tool rather than built into the content space UI. Either way, zones should be provided by Fractal as a way to help navigate large content spaces.

Customization

The previous sections outlined the content space map and some tools for interacting with content and a content space on a daily basis. This section considers how to make customization features accessible from within this map, focusing on modifying active behaviors.

Active Behaviors in the Map

Active behaviors typically consist of pipes and event or URL bindings, and may also generate the user interfaces code for tools. Active behaviors can therefore reside in two places: inside a tool, as part of the tool's logic; or outside the tools, without a user interface component.

Active Behaviors inside Tools

The active behaviors inside tools can be viewed by zooming in on the tool. Zooming in on a tool at the top level of a content space causes it to expand to maximum size, possibly filling the window and revealing more user interface widgets. Earlier in this report we described the map as divided into planes: these views are views of the *Work Plane*. Further zooming will reveal the tool's innards: the active behaviors and user interface components. This zooming action should feel like the user is digging below the surface of the tool, giving the map a 3D feel. This is now the *Behavior Editing Plane*. Here, connections to subspaces and events, and other parameters can be modified.

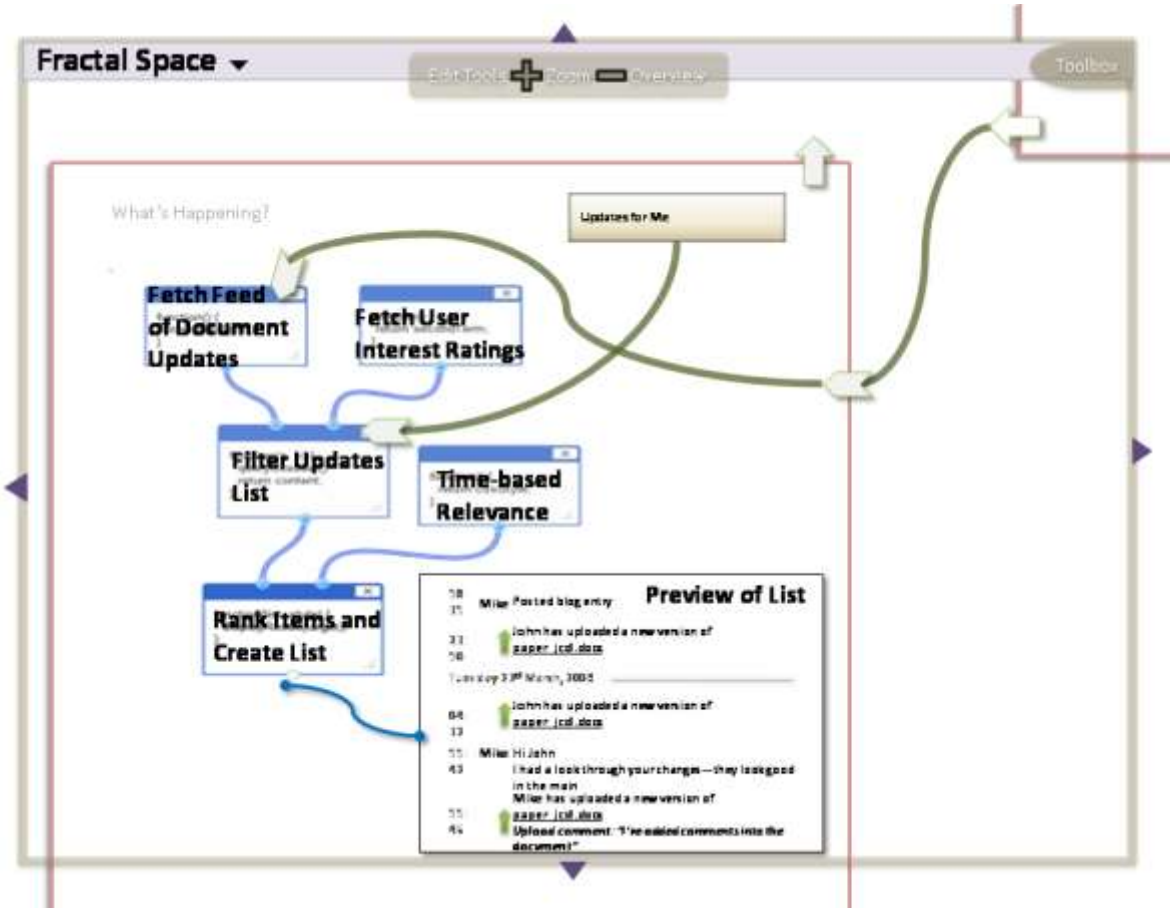

Figure 15: zooming in on the "What's happening" tool on the landing page shows an active behavior that defines the tool's logic, and a user interface component for a list displaying the output of the active behavior.

Top-level Active Behaviors

Top-level active behaviors do not live inside tools – these are automatic behaviors or rules that do not require a bespoke piece of user interface or user application, merely a configuration interface. As a result, they are visible from many everyday views of the content space, when users are navigating between tools. This is useful because it means users can access and modify the active behaviors when necessary, but they should not get in the way of tools. The intention is that these active behaviors also appear to be below the surface of the content space, in the same way that active behaviors can lie below the surface of a tool – all active behaviors lie on the lower Behavior Editing Plane. They still appear distant, and therefore small, and the content space's surface is transparent so they are still visible. Zooming in on the top-level active behaviors zooms in below this surface so that they appear larger and clearer, and can then be modified.

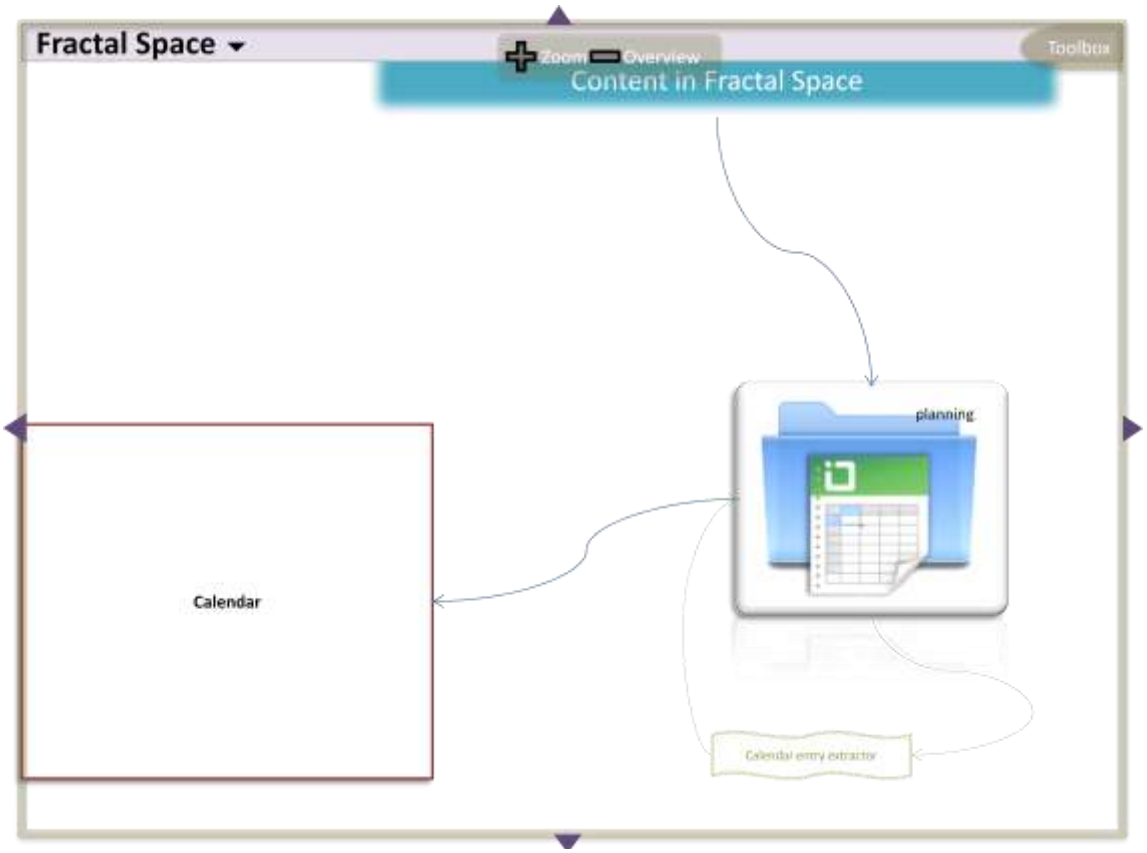

Figure 16: The Calendar Entry Extractor in faint green on the bottom right is an example of a top-level active behavior. Zooming in on it would reveal an editing view as in the previous figure.

Connecting Tools

In addition to modifying the way they work internally, most tools provide input and output *pins* that allow users modifying the data they work on or display. Pins represent attachment points for connectors that take data in or out of a tool. Usually, pins are hidden from users but could be exposed in various ways:

- Mouse over the tool causes its pins to appear
- Right-click on the space and select show pins
- A container button or tool in the tool box that shows pins

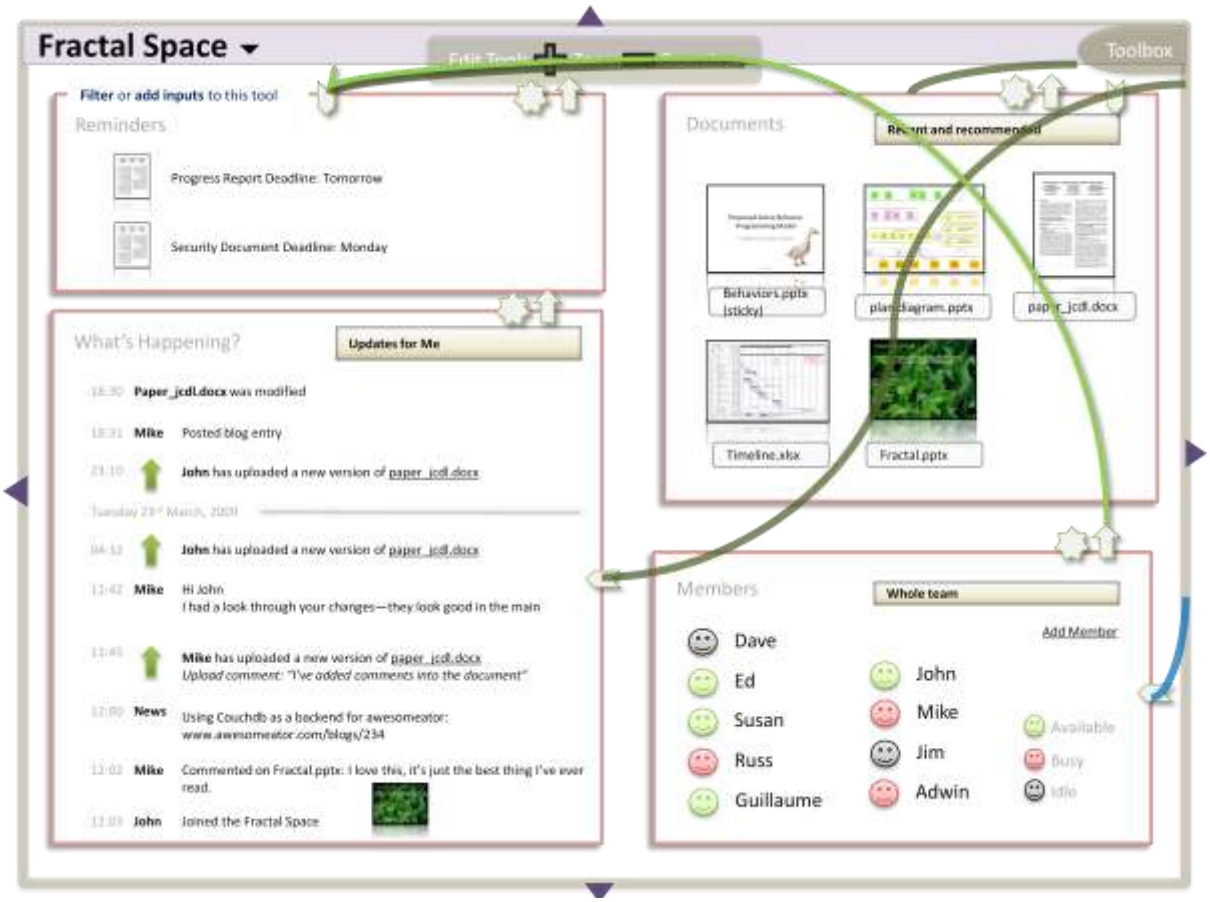

Figure 17: inputs and outputs can be connected at any zoom level, so data displayed by one tool (its output) can be fed as input to another tool. This is useful as tools that transform, filter or retrieve content can be connected to other tools that display, process or allow rich interaction with the content.

A design choice remains over how pins are exposed: they should be easy to see but not intrusive. Input pins on tools are the inputs of its internal pipes, active behaviors and UI templates. Output pins may be defined explicitly, but some can be created implicitly if the system exposes the output of internal pipes and subspaces. This would allow, for example, the data that is passed into the user interface of this tool to be passed to other tools so they could make use of the tools internal processing.

Other implicit pins may be derived from standard UI widgets used by the tools. For example, the list widget used by the "What's Happening" tool could expose an input pin called "filter". It would then filter the items in the list so it only shows items related to the incoming data. Output pins can also be derived from standard UI widgets. For example, a tool might output a list of selected items, which changes when the user clicks on different items in a list or icon view.

As well as tools, pipes and subspaces can also expose pins. Pins are also used in the lower level Behavior Editing Plane, and the same idea is used to connect components in a pipe. Any visible input pins can be connected to any visible output pin of the correct data type. Hovering over pins shows any existing connections, which may be partially hidden at other times, and highlights other pins the user could connect to.

Exposing pins on tools and subspaces means that when viewing several tools, users do not need to zoom in and expose their inner logic to make simple modifications. For example,

users can connect a graph view tool to show several datasets alongside each other by connecting pins. Each dataset is stored in a different subspace. Clicking on the subspace's pin highlights the graph tool's input pin, as it takes data of the type stored in the subspace. The user can connect several subspaces to the graph view tool, which will now display the datasets alongside each other.

Drag and Drop

New tools added from the toolbox or extensions marketplace can be connected using drag and drop. Dragging a tool's icon and dropping it onto another tool, pipe or subspace may have different effects: the system aims to work out what the user intended to happen and do the right thing, asking the user in case of ambiguity. Dragging and dropping may have the following effects:

- Connects input of dragged to output of other if types match
- Connects output of dragged to input of other if types match
- Creates a new, merged view if they both have same display type
- Asks if unsure

Tools already in the content space can also be connected using drag and drop, by dragging a handle that appears alongside the pins.

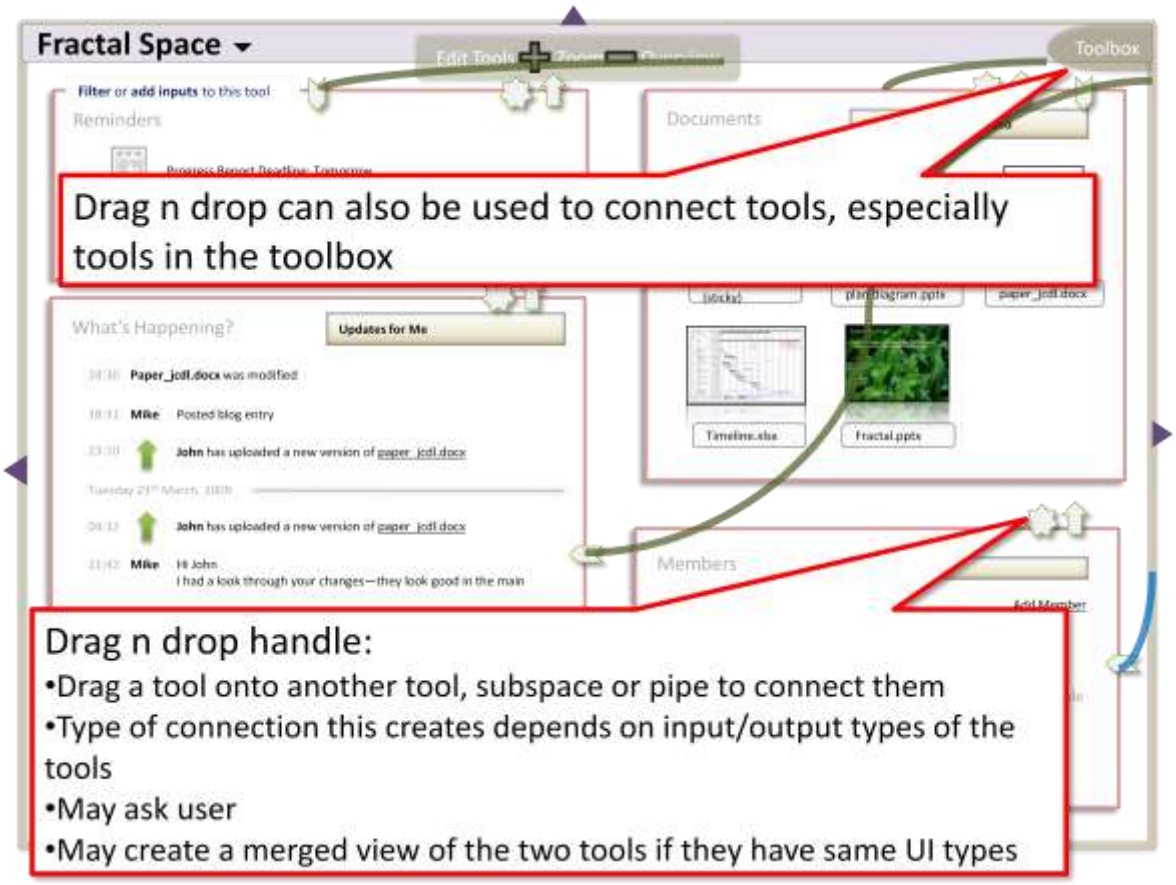

Figure 18: drag 'n' drop is used to add tools to the content space, connect pins and rearrange tools within the space.

Related Work

This interface is novel in the way it connects different aspects of navigation and editing. In most applications where the application can be modified, the user clicks a button in UI, which opens a graphical editor. An example is Yahoo! Pipes (http://pipes.yahoo.com/pipes/), where the implementation behind a user interface can be modified using a graphical editor by clicking "view source". The user does not have the feeling of zooming in, and the editing environment is completely unrelated to the UI. Unlike different parts of a single application, the pipes in Yahoo! Pipes are not related to one another.

Archy [5] and Microsoft Seadragon [6] are projects researching zoomable UIs. These projects use zooming in on a part of the UI to expose more details about visible items – for example, zooming on a hyperlink in Archy reveals the content of the webpage. Like this proposal, a large area can be used to navigate different content and tools, with zooming used as a natural way to gain an overview or access more details about an item. However, zoom is not used for modifying part of the application itself.

Hobnox [7] provides an interface where components in a virtual music studio are connected together using wires. The idea of connectors is similar, and the interface is zoomable but zooming does not allow you to edit the components themselves, or access links between pieces of content.

Some workflow and business process management technologies provide a graphical editor that lets you compose together workflow steps, showing connectors between steps. Examples are Yahoo! Pipes [8] and Taverna [9]. Double-clicking on some steps in a workflow may allow you to edit an individual step. This is a similar idea of drilling down into the implementation of a component, but in this case the components are workflow steps inside the editor rather than tools in an application's UI. This case doesn't help to fuse normal work use and editing an application because it only occurs inside the workflow editor.

In summary, some related work uses similar ideas to tackle slightly different problems: showing connections between parts of in a workflow, but not between components in the application UI, or zooming in to reveal more detail about an item but not navigating to an editor. Prior solutions use simple buttons or hyperlinks to switch to an editor, rather than dedicating an intuitive action like zooming. The editor is also used in a very separate context to the main UI, and what is displayed by the editor is not clearly related to the UI. Interfaces that layout a collection of tools in a geographical map are also unknown.

Conclusion

This report first explored some motivations and principles that we believe are required to fulfill the vision we have for Fractal [1]. This includes the idea of a modular user interface and functionality extensions (tools and active behaviors) that allow total customization of a content space. We put forward the idea of Tools, which are the type of extension in Fractal that can present a new, custom piece of user interface. Later sections propose a zoomable, map-like layout and navigation paradigm designed to help users find tools and information in a content space. The intention is to create a single, zoomable virtual surface that allows users to move smoothly and intuitively between editing and everyday use of a content space. We also suggested the typical view within that map that users may see when entering a content space, i.e. the Landing Page.

The proposed user interface design allows a large degree of flexibility, through separating logic and presentation into active behaviors and tools, allowing users to reposition and arrange content spaces as they wish, and enabling users to switch easily to intuitive graphical

editors to make changes on the fly. The design also has elements that can be simplified, or presented in a different way to the zoomable map we propose. This will ease the task of prototyping and testing, as elements such as tools and active behavior editors are not tied to the map-like layout.

References

1. John Erickson, Susan Spence, Michael Rhodes, David Banks, James Rutherford, Edwin Simpson, Guillaume Belrose, Russell Perry, **Content-Centered Collaboration Spaces in the Cloud, 2009**: http://library.hp.com/techpubs/2009/HPL-2009-11.pdf

2. Edwin Simpson, Guillaume Belrose, James Rutherford, **Fractal Conceptual Prototype: Content Spaces, 2009**: http://library.hp.com/techpubs/2009/HPL-2009-64.html

3. Edwin Simpson, Guillaume Belrose, James Rutherford, **Fractal Conceptual Prototype: Active Behaviors, 2009**: http://library.hp.com/techpubs/2009/HPL-2009-66.html

4. Edwin Simpson, Guillaume Belrose, James Rutherford, **Fractal Conceptual Prototype: the Extensions Marketplace, 2009**: http://library.hp.com/techpubs/2009/HPL-2009-65.html

5. **Archy**: http://web.archive.org/web/20080119202912/rchi.raskincenter.org/index.php?title=Home

6. **Microsoft Seadragon**: http://livelabs.com/seadragon/

7. **Hobnox**: http://www.hobnox.com

8. **Yahoo! Pipes**: http://pipes.yahoo.com/pipes/

9. **Taverna**: http://taverna.sourceforge.net/