

User Interface and Framework Recommendations

uDig

October 12, 2004



Submitted To: Program Manager
GeoConnections
Victoria, BC, Canada

Submitted By: Jesse Eichar, Richard Gould & David Zwiers
Refractions Research Inc.
Suite 400 - 1207 Douglas Street
Victoria, BC V8W 2E7
E-mail: jeichar@refractions.net
E-mail: rgould@refractions.net
E-mail: dzwiers@refractions.net
Phone: (250) 383-3022
Fax: (250) 383-2140

TABLE OF CONTENTS

1	INTRODUCTION.....	3
2	USER INTERFACE RECOMMENDATIONS	4
2.1	uDIG APPLICATION	4
2.2	USER WORK-FLOW	9
2.2.1	<i>First-time Users.....</i>	9
2.3	TYPICAL WORK-FLOW	9
2.3.1	<i>Wizards</i>	9
2.4	DRAG AND DROP SUPPORT.....	11
2.5	uDIG OBJECTS	12
2.6	DATA EDITING	13
2.7	MENU ORGANIZATION.....	13
3	FRAMEWORK REVIEW AND RECOMMENDATIONS.....	14
3.1	BEFORE WE START	14
3.2	COMMUNITY.....	14
3.2.1	<i>Community Review.....</i>	14
3.2.2	<i>Community Recommendations</i>	15
3.3	GUIDELINES.....	15
3.3.1	<i>Guideline Review.....</i>	15
3.3.2	<i>Guideline Recommendations</i>	16
3.4	ICONS GUIDELINES.....	18
3.4.1	<i>Icon Types.....</i>	18
3.4.2	<i>Organization of icons directory</i>	18
3.4.3	<i>Filename conventions</i>	18
3.5	IMAGERY GUIDELINES	20
3.6	PROJECT MODEL	21
3.7	RENDERING	22
3.8	PRINTING	24
3.9	TOOL KIT.....	24
3.10	WEB MAP SERVER.....	25
3.11	WEB FEATURE SERVER	26
3.12	EXTENSION POINTS	27
3.12.1	<i>Tools.....</i>	27
3.12.2	<i>Layer Operation.....</i>	27
3.12.3	<i>Style.....</i>	28
3.12.4	<i>Renderer.....</i>	28
3.13	CATALOG	29
3.13.1	<i>Catalog Review.....</i>	29
3.13.2	<i>Catalog Recommendations</i>	29
3.13.3	<i>Catalog API Recommendations.....</i>	29

1 INTRODUCTION

The purpose of this document is to make recommendations on ways to improve the user interface of uDig as well as ways to improve the overall architectural framework of uDig. The start of this document outlines the user experience with the current version of uDig and recommends a number of important changes that should be made to the uDig user interface. These recommendations are a result of conducting informal user and developer interviews.

Many of the users interviewed expressed concern over the following topics:

- Drag and Drop support
- Streamlined user work-flow
- Simple introduction to uDig for first-time users
- Improved editing work-flow
- Logical Menu organization
- Preference Settings

The user experience recommendations are rounded out with a set of recommendations to both the uDig platform extensions, and the uDig core application.

The following extension points have been reviewed, with recommendations for improvements:

- Tools
- Layer Action
- Decorator
- Style
- Renderer

With this set of extension point recommendations, some of the core functionality and interfaces also require updating. Most of the updates to the core that we recommend completing fit into the following categories:

- Model
- Rendering
- Printing
- Local Registry
- WMS
- WFS

2 USER INTERFACE RECOMMENDATIONS

The sections that follow document the recommendations for improvements to the user interface.

2.1 uDig Application

The following image shows a screen shot of the current version of the uDig application.

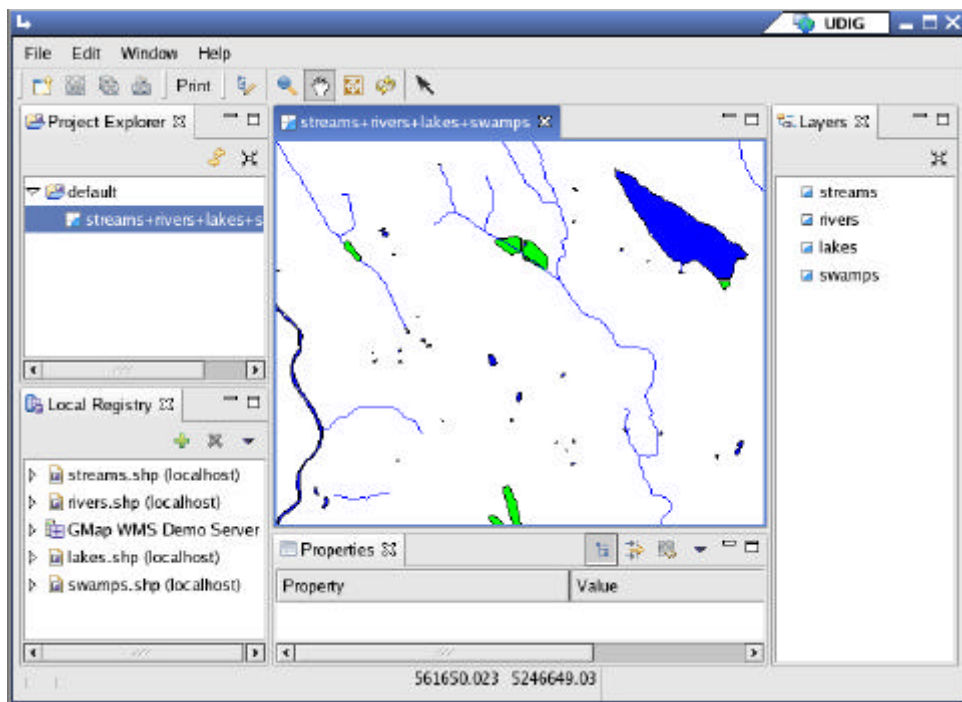
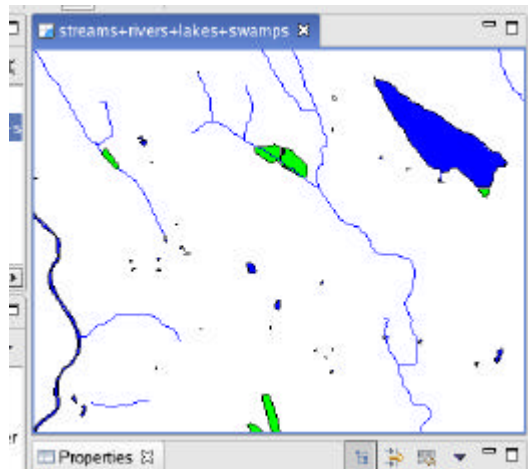


Figure 1 - uDig Application

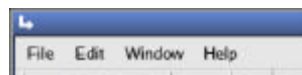
Major components of the uDig User Interface are:

- **Map Editor**



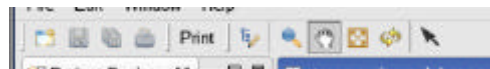
Located in the center of the application. The Map Editor in the image displays a map with four layers, each with a different style.

- **Menu bar**



Located at the top of the application just below the window title. The Menu bar contains functions for all of the important parts of the application. For example, the file menu contains items that allow a user to save and create maps.

- **Tool bar**



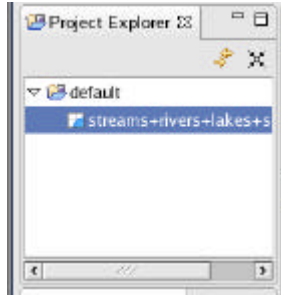
Located below the Menu bar. The Tool bar provides access to tools and functions used with the Map Editor.

- **Status Line**



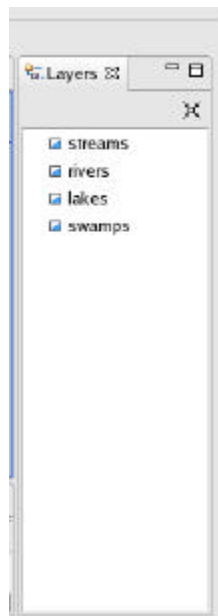
Located at the bottom of the application window. The Status line displays dynamic information for the user. In Figure 1, the Status line displays the coordinates of the current mouse location in the world coordinate system.

- **Project Explorer**



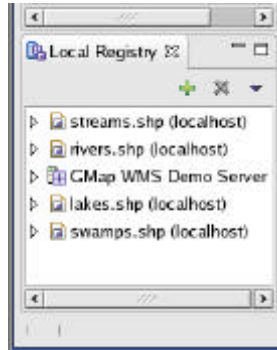
Located below the Tool bar and on the left of the Map Editor. The Project Explorer lists the known projects and the maps contained within each project.

- **Layers**



Located on the right of the Map Editor. The Layers View shows the layers of the map currently displayed in the Map Editor.

- **Local Registry**



Located below the Project Explorer. The Local Registry shows all the data registered with the current application. Layers can most easily be added to maps from the Local Registry.

- **Properties**



Located below the Map Editor. The Properties view shows the properties of the current selection, or the currently selected feature collection.

- **Wizards**

Wizards are accessed through menus and through tool bar buttons. Wizards assist the user with standard, repetitive, or tedious tasks. For example, wizards facilitate creating a new map or importing a data source. Figure 2 shows the new Map Wizard. The **Next** button takes the user to the style selection page; the **Finish** button creates a new map with the selected layers with default styles.

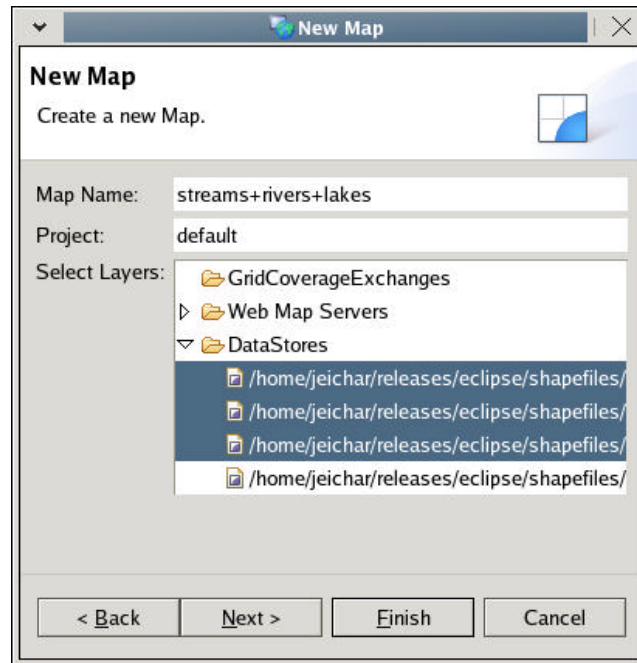


Figure 2 - New Map Wizard

2.2 User Work-Flow

2.2.1 First-time Users

The initial execution of uDig needs to be friendlier for first-time users. It should start with a welcome page similar to Eclipse's. Once the workbench has been opened, it should start with a default project and an empty map. The concept of a default project should be hidden from the user.

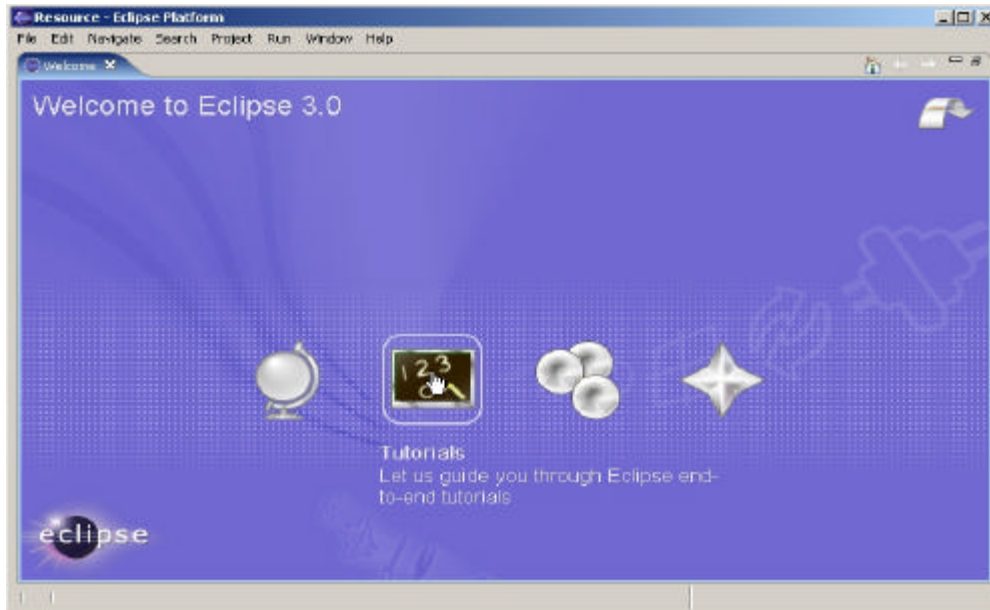


Figure 3 - Welcome Page in Eclipse

2.3 Typical Work-flow

2.3.1 Wizards

The overall work flow for most wizards needs improvement. All wizards should remember previously used settings and maintain histories, such as recent files accessed.

Import wizards need to be based on importing a file or URL rather than a Shapefile, Web Feature Server or Web Map Server. The first page of a file-importing wizard should feature a file browser. When importing a Shapefile that requires that custom properties be filled out, a checkbox should be displayed. If checked, the next page in the wizard will allow the user to fill out those custom properties.

The wizards responsible for creating new Maps or adding Layers to existing Maps should have an add/find Layer button which can import new data into the Registry.

Some other miscellaneous points:

- For all of the operations within uDig, instant feedback must be provided. At the minimum, a mouse cursor change is required.
- All of the tool tips should be in the active tense.
- Each plug-in should implement context sensitive help.
- The Layer View should allow reorganization of layers by dragging them up or down.
- In the Local Registry View, items with only one sub-item should display only the sub-item.

2.4 Drag and Drop Support

This section examines Drag and Drop functionality and how it should be incorporated into uDig. Version 0.4, the alpha release, does not support Drag and Drop.

Drag and drop support is a very intuitive user interface construct where the user selects an object, drags it to a new location, and drops the object. The two components of Drag and Drop are the drop source and the drop target. The drop source is the item selected and dragged. The drop target is the location that the source is dropped on. The drop target is responsible for recognizing the source and making meaningful use out of it. In addition to simply making meaningful use out of the source item, the drop target should use defaults as much as possible.

There are a number of drop targets that should be part of uDig:

- **Map Editor**

Accepted Drop Sources:

- URL - Interpreted as a layer or set of layers.
- File - Interpreted as a layer or set of layers.
- Layer from local Registry - Interpreted as a layer.
- Map - Interpreted as a set of layers.

The new layers derived from the source items are added on top of the map's other layers.

- **Layers View**

Accepted Drop Sources:

- URL - Interpreted as a layer or set of layers.
- File - Interpreted as a layer or set of layers.
- Layer from local Registry - Interpreted as a layer.
- Map - Interpreted as a set of layers.

If the source is dropped over a layer, a cursor should appear above or below the layer, and dropping the source will add layers at the cursor location.

Layers within the Layers view can be dragged within the Layers view in order to change the order of the layers.

- **Project Explorer**

Accepted Drop Sources:

- URL - Interpreted as a project, map or page if dropped on a project. Interpreted as layers if dropped on the map.
- File - Interpreted as a map or page if dropped on a project. Interpreted as layers if dropped on the map.
- Map - Interpreted as a map if dropped on a project. Interpreted as layers if dropped on the map.
- Page - Interpreted as a page if dropped on a project.
- Project - Interpreted as a project.

- **Local Registry**

Accepted Drop Sources:

- URL - Interpreted as a data source.
- File - Interpreted as a data source.

2.5 uDig Objects

There are several different types of uDig objects: Registry, Map, Project, Page and Layer. Each of these objects requires some changes in order to meet our requirement of being user-friendly.

The Registry needs to be renamed to Catalog as that is more reasonable and less programmer-oriented. Objects within the Registry itself should not be duplicated and should be re-nameable. Web Map Server layers within the Registry should display their icon, retrieved from the Capabilities document.

The Project should have options to copy Maps and Pages within it. It could also contain a special kind of Map or Page called “new map” or “new page” which, when clicked, will bring up the new Map or new Page Wizard.

When creating a Map, a default name should be provided which can be changed later. When adding a Web Map Server to a Map, currently every Layer contained within the WMS is added. One should be able to add specific Layers. It should also be possible to delete a Layer from a Map.

Each Layer should have an associated icon that provides useful information about its state.

All of these objects should have their own properties page.

2.6 Data Editing

Data editing currently demonstrates what can be done. There is a button on the tool bar, but not within a menu, that allows the user to begin editing. Once a feature is selected using the selection tool, the vertex manipulation tool can be used to add or move vertices. Improvements that should be made to the editing user interface are as follows:

- The begin/end edit mode should be accessible through the main menu and the map's context menu in addition to a tool bar button.
- The user should not be required to select a feature using the selection tool. The vertex manipulation tool should select a feature when the feature is clicked on.
- Clicking in an area where no feature currently exists should start a new feature. By clicking in the display area using the vertex tool the user should be able to create a new feature without being required to choose a new tool.
- As feature geometries are manipulated, the geometries should be validated, preventing the user from creating invalid geometries. For example, if a vertex is moved to an illegal position, the move should be undone and the user should be notified that they have made an illegal modification.

2.7 Menu Organization

The menu system in uDig needs to be defined. The addition of a menu for each of the Layer, Map and Project objects will allow users to easily locate and perform actions on them.

- The Layer menu should have a "Clear Selected Features" button.
- The Map menu should have an entry for each of these properties: "Start Editing", "Stop Editing", "New Feature" and "Clear Selected Features".
- The Project menu should have an entry for "Properties".

3 FRAMEWORK REVIEW AND RECOMMENDATIONS

This section provides a thorough review of the existing uDig Framework. The focus is on the usefulness and accessibility of uDig as a development platform.

While the structure of the application will be discussed, the focus is on the Plugin-Developer, and the community we wish to build around uDig as a GIS platform.

3.1 Before we start

Before we start the review process, we outline two important recommendations:

- **GIS Platform.** In keeping with the example provided by other RCP projects, the uDig Framework should be called a Platform. GIS Platform seems to be a sufficiently accurate term.
- **Catalog.** The current concept of a Local Registry (a mirror of the OGC Web Registry Service) has proved confusing at the User Interface level. We will carry the recommendation over to the GIS Platform.

3.2 Community

We have made the following preparations for a more active uDig development community.

- Community Wiki – for collaborative documentation including a User's Guide
- Email list – used for project communications
- Issue Tracker – customized for this project, with specific instructions and a reduced number of fields
- Releases
- Documents and Reports
- Version Control

3.2.1 Community Review

The email list has met with great success. New developers show up weekly and are enthusiastic. Similarly the series of releases has helped greatly in sorting out installation and connection issues. Community members have been active in creating bug reports in our Issue Tracker.

Less successful have been our documents and reports. The only report to gather significant feedback has been our comparison of JUMP, Eclipse and Netbeans.

We have yet to receive a request for version control access.

3.2.2 Community Recommendations

We plan to set up version control and website space for plug-in developers. In addition to gathering up uDig contributions in a single location this will help keep the email list alive and happy.

3.3 Guidelines

The existing Project Guide¹ provides a number of important guidelines² for framework development:

- **Sensible Defaults:** this system is "user friendly"; make the right choices rather than complicate matters.
- **Code for Developer Convenience:** this system is designed to be extended by others. A principle of "least surprise" applies to developers as well as users.
- **Eclipse RCP Guideline:** anything that references a "resource" is not intended for RCP developers.
- **Eclipse User Interface Guidelines:** Eclipse provides a strong set of user interface guidelines. We intend to follow them.
- **Decisive Coding (a.k.a. "code like you mean it"):** this project is on a short time line, so don't waste time in "Analysis Paralysis". We are developers; if we change our mind we can change our code. There is time enough for the code base to be entrenched later.

These are the criteria we have defined for GIS Platform development.

3.3.1 Guideline Review

These guidelines have held up reasonably well under heavy coding strain.

The practice of using sensible defaults has not yet begun in earnest (as this document has pointed out at numerous occasions).

Coding for developer convenience has been hampered by our growing understanding of what is considered "surprising". While we feel comfortable providing what is expected for OGC-based GIS development, the impressive precedent set by the Eclipse platform is a hard act to follow.

The *Eclipse RCP Guideline* has proved easy to follow, as a violation of this guideline prevents release. The main useful ingredient this guideline prevents is the use of IResource. For many developers the availability of IResource (roughly similar to a file handle or proxy) represents the usefulness of the Eclipse platform. We may be asked to include this construct at a later time.

The *Eclipse User Interface Guidelines* have become dated over time; the recent release of Eclipse 3.0 has seen an overhaul of Icon standards in particular. This

¹ Project Guide, <http://docs.codehaus.org/display/uDig/Project+Guide>

² Guidelines, <http://docs.codehaus.org/display/uDig/Guidelines>

lack has been addressed with documentation³ based on observation of Eclipse 3.0 practice.

The practice of *decisive coding* has benefited this project in many ways. This is a luxury we will enjoy as long as possible; as the project picks up steam we will be unable to retain this flexibility. Our one area of indecision concerns the relationship between a Layer and a Renderer.

3.3.2 Guideline Recommendations

We have discovered the **Eclipse House Rules**⁴ as an interesting target for where we should be heading:

Extender Rules

Contribution: Everything is a contribution

Conformance: Contributions must conform to expected interfaces

Sharing: Add, don't replace

Monkey See/Monkey Do: Always start by copying the structure of a similar plug-in

Relevance: Contribute only when you can successfully operate

Integration: Integrate, don't separate

Responsibility: Clearly identify your plug-in as the source of problems

Program To API Contract: Check and program to the Eclipse API contract

Other Rule: Make all contributions available, but put those that don't typically apply to the current perspective in an *Other...* dialog

Adapt to IResource: Whenever possible, define an IResource adapter for your domain objects

Strata: Separate language-neutral functionality from language-specific functionality and separate core functionality from UI functionality

User Continuity: Preserve the user interface state across sessions

Extender

Invitation: Whenever possible, let others contribute to your contributions

Lazy Loading: Contributions are only loaded when they are needed

Safe Platform: As the provider of an extension point, you must protect yourself against misbehavior on the part of extenders

Fair Play: All clients play by the same rules, even me

Explicit Extension: Declare explicitly where a platform can be extended

Diversity Rule: Extension points accept multiple extensions

Good Fences: When passing control outside your code, protect yourself

User Arbitration: When there are multiple applicable contributions, let the user decide which one to use

Explicit API: separate the API from internals

Stability: Once you invite someone to contribute, don't change the rules

Defensive API: Reveal only the API in which you are confident, but be prepared to reveal more API as clients ask for it

Publisher

License Rule: Always supply a license with every contribution

³ Icons, <http://docs.codehaus.org/display/uDig/Icons>

⁴ *Erich Gamma, Kent Beck - Contributing to Eclipse: Practices, Plug-Ins, Patterns

These rules should gradually be adopted as the project moves towards completion.

Several should be adopted immediately:

- **User Continuity:** *Preserve the user interface state across sessions.*
We have several examples where the user supplied state is not being retained, particularly during the use of Import Wizards.
- **Fair Play:** *All clients play by the same rules, even me.*
We should adopt this guideline to prevent direct dependencies between modules and to ensure our extension-points are useable.

It is too early for the adoption of several rules (such as Stability); others (such as Adapt to IResource) don't apply.

3.4 Icons Guidelines

The Eclipse User Interface Guidelines have relaxed somewhat. This section documents these changes.

3.4.1 Icon Types

TYPE	type	description
local toolbar	lcl	found on the far right of the title area of a view
toolbar	tool	used in cascading menus, and the global toolbar
view	view	found in the top, left corner of a new view
model object	obj	used in the tree, list, properties views, and editor tabs
overlay	ovr	placed on top of model object to indicate a change
wizard banner	wizban	used in wizard dialog windows

3.4.2 Organization of icons directory

DIRECTORY	disabled	Enabled	Other	Banner	Size	Placement
local toolbar	dlcl16/	elcl16/			16x16	left & top clear
toolbar	dtool16/	etool16/			16x16	left & top clear
view	dview16/	eview16/			16x16	left & bottom clear
model object			obj16/		16x16	centered, bottom clear
overlay			ovr16/		7x8	one pixel white outline
wizard banner				wizban/	55x45	bottom left on blue gradient
PALETTE	color	gray scale	color	color		

*perspective & fastview icons require the right and bottom edges to be clear.

3.4.3 Filename conventions



























The original guidelines documented two directories, *clcl* and *ctool*, which allowed a full colour palette; for Eclipse 3.0 *elcl* and *etool* have supplanted this use.

FILENAME SUFFIX	lcl	tool	view	obj	ovr	wizban
invoke a wizard, or graphics in a wizard		_wiz				_wiz
invoke executable file		_exec				
in an editor view		_edit				
in a navigator view	_nav	_nav	_nav			
do not fit into a category	_misc		_misc			
represent tasks that user can do	_tsk		_tsk	_tsk	_tsk	
toggles the working mode of the view	_mode					
found in a menu	_menu					



































































found in a property sheet	_ps	_ps	
used in the tree, list, or property view			_obj
model object icons on object palettes			_pal
commands that engage the system	_co	_co	

3.5 Imagery Guidelines

The GIS Platform makes use of a consistent set of imagery when defining icons and application graphics:

	udig logo		project file
	feature		grid
	datastore		grid coverage exchange
	web feature server		web map server
	feature file		grid file
	map file		pixel
	repository		web registry service
	server		database
	project		project missing
	map		empty map
	map layer		map layer empty
	map folder		map folder missing
	paper		page template

Eclipse Guidelines provides the following representations:

create, new		compare		forward		jar		plugin	
save		debug		backward		WAR		extension	
cut		run, execute		previous		EAR		extension point	
copy		import		next		window		thread	
paste		export		project		perspective		process	
add		play, resume		open project		property sheet		mapping	
remove		suspend		folder		table		error	
delete		terminate		open folder		database		warning	
erase, clear		stop		file		repository		alert	
search		undo		library		class		conflict	
find		redo		package		interface		public	
help		refresh		session bean		attribute		protected	
edit		filter		server		element		private	
								default	

This use of common imagery provides a unified look and feel to the GIS Platform.

3.6 Project Model

This section discusses the uDig data model and is primarily a way to provide context for the reader to understand the following sections.

At the top of the data model is the **project** class. The project contains maps and pages, which in turn contain viewport models, context models and layers - the rest of the data model. Because projects contain maps, pages and layers, they are responsible for creation and deletion. Projects in uDig are organized as single files; all of the maps and pages contained by a project are saved into the file. As a result projects are responsible for managing, saving and loading maps and pages as well as themselves.

Map objects contain one context model and one viewport model and have little functionality other than as a way for accessing the models and being related to the concept of a map. The **context model** contains the layers of a map and notifies interested listeners when the layer state somehow changes. A **viewport model** encapsulates the view of the data. In other words, it specifies the area of the data being viewed and what coordinate reference system the view, not the data, is in.

The **layer** provides access to the “real” data and models map related information such as the style used for the layer, the z-order of the layer, the display name of the layer, the visibility of the layer and the selectability/editability of the layer.

The **page** is the last piece of the data model and is discussed in more detail later. It is sufficient here to say that a page represents how a map should be printed on paper. A page contains boxes that contain decorators and maps.

The model API is satisfactory as is, however the viewport model interface needs to be separated into different interfaces: viewport model and viewport operations, because the current API has some convenience methods that have proved confusing to third party developers. For example, there is a method for obtaining the display size. This leads a developer to believe that the viewport model is responsible for controlling the display area, which is not the case.

3.7 Rendering

The rendering architecture is organized as a set of renderers, a timer that polls the renderers for updates and a Java Advanced Imaging (JAI) tree that combines all the rendered images from the renderers into one image that is displayed. Figure 4 illustrates the current behaviour of the rendering system.

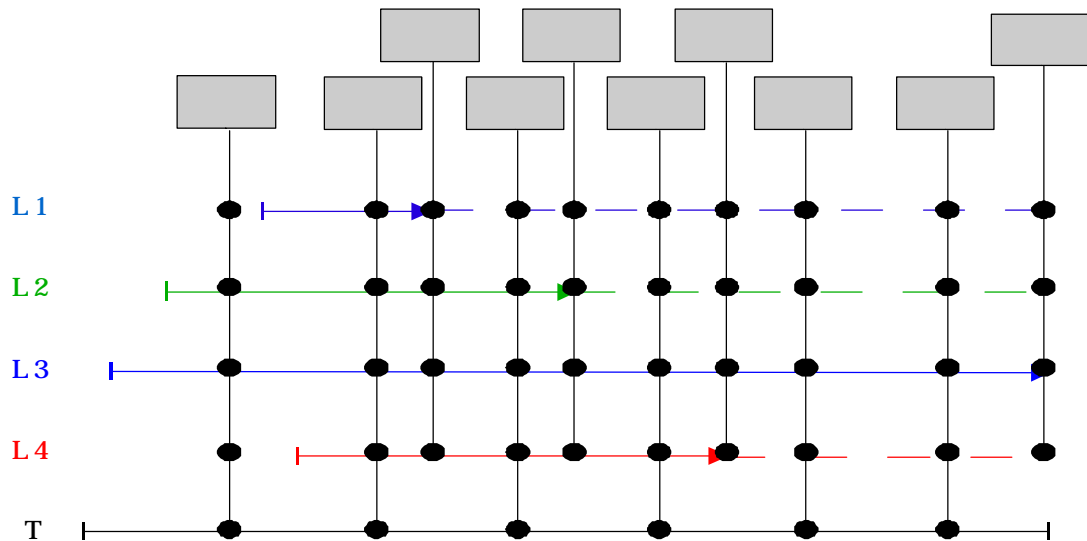


Figure 4 - Current JAI Rendering System

L1 through **L4** are four renderers, each rendering a different layer. The x-axis shows time increasing from left to right. The solid lines beside **L1-L4**, beginning with a bar and ending with an arrow, show the execution of the renderers. The bar indicates the time when the first data arrives from the data source, and the arrow indicates the end of execution. The dotted lines indicate that the rendered image is complete. The vertical lines indicate the rasters being merged and displayed; the black dots indicate which layers are being merged. The line beside **T** is the timer thread. As Figure 4 illustrates, the timer currently wakes up at regular intervals and obtains the image raster from each layer, regardless of whether the image has begun rendering. In addition, because the timer thread wakes every second, each renderer can initiate a display update. The last point to observe is that all layers are merged each time an update occurs.

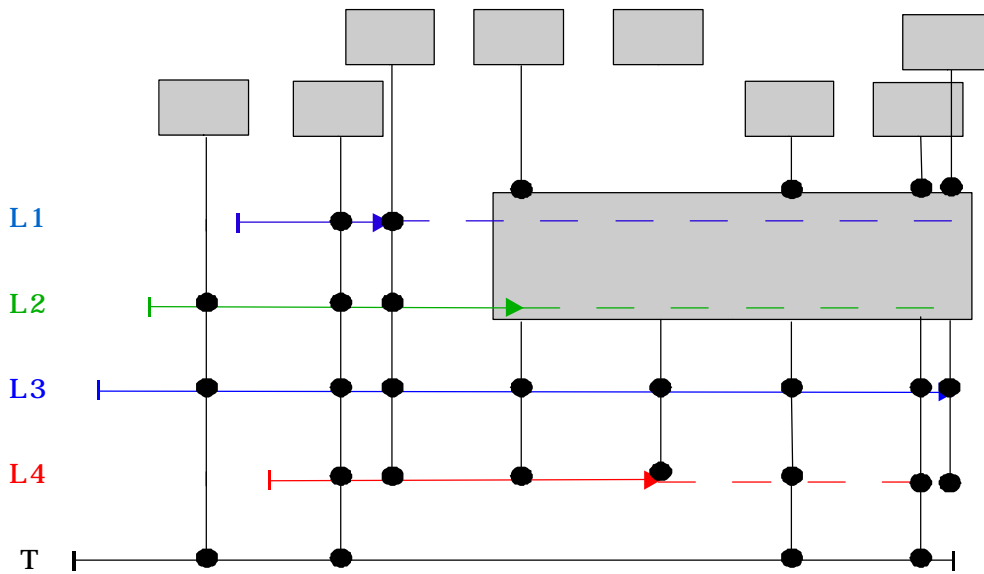


Figure 5 - Recommended JAI Rendering System

Figure 5 illustrates some improvements on the current implementation.

The first improvement is illustrated in the left-most vertical line. The update thread only merges rasters that have begun rendering. The second improvement is that the timer thread is reset each time a renderer initiates an update. Finally, adjacent rasters that have been completed are kept in a buffer so they can be merged as one raster. These improvements have yet to be implemented.

3.8 Printing

The printing framework needs significant improvement, especially with regard to printing Web Map Server maps at higher DPIs. The Layout system has been renamed to Template and a basic implementation has been created and needs to be extended.

The current implementation is capable of simply printing out a Map with a title. An extension point has been created, but is not completely usable as the printing system uses the first Template it encounters.

Ideally we would like to implement a system that allows users to create their own Templates. They could then be used to define a Page, which is sent to the printer.

A Template is flexible and adapts to various page sizes. Upon creation, the Template is given a page size and orientation and provides a properly formatted Page. It should also be possible to create Pages by deriving them from other Pages.

3.9 Tool Kit

Currently we are using the Geotools data access libraries. These are categorized into Feature providers and Image providers. Both groups take in a set of parameters to produce the requested data. One of our recommendations is to make all data producers accept a single parameter with all others being filled using sensible defaults. This will allow a File, URL or IP to be interpreted automatically, creating a new data source.

While Image data providers currently have a sense of a layer provider, and a collection of layers, Image data providers do not provide easy access to data through a single parameter constructor. We would encourage the Image data libraries to include such a constructor.

Feature providers need considerably more effort to complete an automated constructor. This is mainly due to the lack of singular layer support within the current API. We would encourage the Geotools community to provide two forms of data access, one as a singular layer, the other as a collection of layers. Of course both access methods would be capable of being queried and written to.

Upon the division of the Feature providers' current API, we feel it would be relatively simple to add default constructors for most singular sources, including ESRI Shape files and GML files. We would also like to see sensible defaults with a singular constructor for multi-layer Feature providers, including WFS, PostGIS, and Oracle Spatial.

3.10 Web Map Server

The Web Map Server implementation is a part of the Geotools project. Currently it is capable of reading maps from all versions of the WMS specification. The next implementation will see more key functionality added.

One of the most important things is the implementation of the GetFeatureInfo request, allowing information to be retrieved regarding features at a point in the map.

The ability to make requests using POST rather than GET is also quite important. This allows us to make larger requests and utilize SLD-enabled Web Map Servers.

There are also four requests to be implemented to allow communication with SLD-enabled WMSs.

3.11 Web Feature Server

The Web Feature Server data access library resides within the Geotools project. Currently this extension has been implemented and tested to read data from OGC compliant WFS instances. The creation of this client library spawned the creation of a sophisticated XML schema and document parser.

The parser successfully parses and validates XML documents using their instance schemas. The inheritance within the schemas is used in conjunction with well known schemas to provide direct parsing of the XML document into user defined Java objects. This XML parsing framework is also capable of streaming objects, or publishing the objects to the remainder of the system as they become available. Some of the recommended improvements reside within the parser framework, while other improvements are directly related to the WFS client. For a more detailed explanation of the parsing framework see Open Source GML Parsing (<http://udig.refractions.net/docs/osgp.pdf>).

Many of the recommended improvements will also benefit the GML file reader, which streams GML data from disk. This was built as a proof of concept, as parsing large GML documents was a pre-requisite to a WFS client library.

A large portion of the functionality required to write transactions to a WFS server instance has also been completed, but as of yet is only partially tested. To provide reliable write access to a WFS extension, more comprehensive testing should be completed.

The current WFS implementation has not been thoroughly performance tested, but at first glance there are several directions that ought to be pursued for this WFS client library to perform well. Some speed increases may be realized through multiple concurrent requests, optional data validation, and improved client side transaction caching.

Although in most cases concurrent requests would not greatly affect the performance of a WFS instance, some vendors' WFS servers would benefit. In cases where it appears to be vendor specific, metrics should be included to split requests appropriately.

While most major XML parsers perform some form of validation, many of these parsers also optionally disable validation. The WFS client should also include this functionality, reducing the overhead for parsing complex data components.

Lastly, operations cached against a particular uncommitted transaction may be better stored to improve the performance of reading the results of a query. This would enable improved feature modification on the client side with respect to pending transactions.

Although the WFS client library does not currently support locking, we do not recommend supporting server-side locking until the user interface requires locking functionality.

3.12 Extension Points

Extension points provide programmers with a way to write extensions for uDig. In order to be useful, an extension point must be well defined and very stable. This section discusses the current extension points and makes recommendations with regards to how the current extension point definitions can be improved.

3.12.1 Tools

The current tool extension point allows third-party developers to develop new tools for uDig and is one of the most used points of extension. The current implementation specifies three different types of tools and two ways of grouping tools. The three types of tools are as follows:

- **Action Tool** - A single fire tool that performs a single action and is not modal; a button that sets the viewport so it frames the current selection is an example of an action tool.
- **Modal Tool** - A tool that has on and off modes. When a modal tool is "on" it waits for user input and reacts on it. An example of a modal tool is the zoom tool.
- **Background Tool** - A tool that is always active in the background. A typical background tool would be limited to providing user feedback. An example is the cursor position tool that displays the current mouse location in world coordinates.

The two grouping mechanisms are:

- **Category** - A collection of tools that are always available but are logically similar and therefore grouped together.
- **Mode** - A collection of tools that are similar and are grouped together but are only available when a mode is active. An example would be editing tools.

Tools can be assigned to appear in the tool bar, the menu, and the status line.

Most of the tool extension point is satisfactory; all the current tools are created using the current definition, however, the current definition is not scalable enough. If a large number of tools were added, then it would be useful to have the concept of a tool palette. To address this need the current extension point definition should be modified so developers can add tools to views (the Eclipse concept of a view). In addition, standard tool views should be defined so tool developers are not required to create views in addition to tools. View creation would be optional.

3.12.2 Layer Operation

A layer operation ("op") extension is an extension that performs an operation on one or more layers in a map. Since the layer op extension point is not yet defined this section is a recommendation. The layer op extension point will allow programmatic access to a layer. Layer operations would be listed under the layer menu and in context menus in the layer view when a layer is right-clicked. In order to provide a scalable solution the layer operation extension point will

require an extension to declare a filter that will allow the menu managers to determine whether the layer operation is interested in the layer. If it is, then the operation will be added to the menu. Otherwise the operation will be left out of the menu.

3.12.3 Style

The style extension point is defined so that special purpose renderers are tied to the SLD styles. For example, it is possible for a renderer to render from CAD formats that might have special requirements for styling. Styles also reduce the amount of refactoring that is required to port renderer implementations into uDig. The current specification requires that a style extension provide a style editor - a style that can create a style memento. A style memento is a flexible uDig construct that is used to save the style. Since renderers create styles uDig does not need to know about the concrete style class; only the renderer responsible for the style needs to know the concrete style class. Thus far the style extension point has successfully satisfied its requirements.

3.12.4 Renderer

A Renderer interprets spatial data and represents the data in a visual manner. In uDig there are different types of renderers that can render different types of data. For example, a feature renderer can render feature data. A WMS renderer can communicate with and render images from a web map server.

The current API requires that a Renderer extension must create: a RenderMetrics class, which can provide metrics about how fast a renderer can provide its service; and, a RenderMetricsFactory class, which can determine if a data source can be rendered by the renderer and can create RenderMetrics that provide metrics with regard to a particular data source. Furthermore, a renderer extension must provide an implementation of the Renderer Interface, normally by extending the abstract superclass that handles threading and event notification for the renderer.

The renderer extension point has had three different extensions and has satisfied all their requirements well. However, for the sake of simplicity a toolkit object should be provided that the renderer can use. This would provide the user with a buffered image that the renderer could write on and provide a single interface for obtaining all information that the renderer may need. In the current implementation the Abstract superclass that each renderer is intended to subclass provides such an API. However, because the AbstractRenderer performs many different functions it is difficult to tell what methods and fields the extension should use. Separating the functionality and data access methods into a separate class would make the distinction more explicit and reduce programming errors.

3.13 Catalog

As mentioned at the start of this section the biggest recommendation for Catalog has been a name change (from Local Registry).

3.13.1 Catalog Review

Catalog services have been well imagined and require very little in the way of functional modification.

In practice we have found using the existing RCP Import Wizard extension point sufficient for managing the addition of data sources.

User Interface and Data Discovery requirements will be pushing several changes. We will need to add the following:

- Simplified registration allowing for Drag and Drop support
- Catalog Services version 2.0

3.13.2 Catalog Recommendations

In order to support flexibility as the toolkit APIs change, we recommend the use of IAdaptable to support using a variety of data access and discovery services while maintaining a single registry.

- Registry.getAdapter(Class class)

Toolkit Access:

- GeoAPI Catalog (GeoAPI) – captures OGC Catalog Services version 1.0
- Discovery (pending) – captures OGC Catalog Services version 2.0
- Repository (Geotools) – allows for cross data source operations

Catalog will continue to support the Registry, and RegistryEntry natively.

3.13.3 Catalog API Recommendations

To simplify the registration of data sources the following API is proposed:

- Object CatalogPlugin.open(File file)
- Object CatalogPlugin.open(URL url)
- Object CatalogPlugin.connect(ParameterValueGroup params)