

Participatory Interface Design for Expert Users

Development of a design for the administrator interface of Hydra – a tool for managing document processing pipelines in enterprise search

Master of Science Thesis in Interaction Design

MANDIS SÖDERSTRÖM JOHANSSON

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.

The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Participatory Interface Design for Expert Users

Development of a design for the administrator interface of Hydra – a tool for managing document processing pipelines in enterprise search

MANDIS SÖDERSTRÖM JOHANSSON

© MANDIS SÖDERSTRÖM JOHANSSON, February 2012.

Examiner: FANG CHEN

Chalmers University of Technology
University of Gothenburg
Department of Applied Information Technology
SE-412 96 Gothenburg
Sweden
Telephone + 46 (0)31-772 1000

[Cover:]

A wireframe showing the developed design for the Hydra administrator interface. For a detailed description of the interface see chapter 6, Result.

Department of Applied Information Technology
Gothenburg, Sweden, February 2012

Abstract

The purpose of this project was to investigate if participatory design is suitable when designing for expert users. The project was executed at Findwise, a company that works in the field of enterprise search. The design for an administrator interface for Hydra, a tool that manages document processing pipelines intended for in-house usage, was developed.

The future users of Hydra were highly involved in the design process to take advantage of their knowledge and experience. At first their current work situation was analyzed by interviews and observations. Then ideas for the new interface were generated during participatory workshops. The best ideas were then translated into sketches and prototypes that were tested and evaluated through paper prototyping and discussions in iterations. The work resulted in a list of requirements for the interface and a design proposal in the shape of wireframes that was intended to meet these requirements. It turned out that the design ideas the users liked the most were transparent and very close to the implementation model of Hydra, which follows the guidelines for design for experts.

During the development the users expressed valuable ideas and gave input that a person without their experience would not have been able to provide. Most of the users found it hard to discuss innovative ideas since they tended to focus on implementation matters, which sometimes inhibited the creativity. This indicates that a participatory design approach did contribute to a good design, but the structure of the participatory activities must encourage the creativity if more innovative solutions should be developed by expert users.

Based on the feedback from the users and stakeholders regarding the final design, they are satisfied with the result. To establish this, the interface has to be further developed into a prototype that handles real data and then be tested in an actual work situation.

Acknowledgements

I would like to thank Findwise for giving me the opportunity to execute this project, and the employees at the company who participated in the design process for their valuable input. I would also like to thank my supervisor at Findwise Björn Klockljug Johansson for his help and feedback both regarding the design work and this report.

Table of Contents

- 1 Introduction..... 1
 - 1.1 Aim..... 1
 - 1.2 Research Question 1
 - 1.3 Limitations 1
 - 1.3.1 Focus on the Design Process 1
 - 1.3.2 Solid Concept Rather Than Details 1
 - 1.3.3 Innovation versus a Realizable Solution..... 1
- 2 Background..... 2
 - 2.1 Enterprise Search 2
 - 2.2 Document Processing Pipelines 2
 - 2.3 Hydra 3
 - 2.4 Interfaces of Existing Pipeline Administrator Tools 4
 - 2.4.1 OpenPipeline 4
 - 2.4.2 FAST ESP 6
 - 2.5 Tools 9
 - 2.5.1 Pencil Project..... 9
 - 2.5.2 Microsoft Lync 9
- 3 Literature Review 10
 - 3.1 Cognitive Frameworks..... 10
 - 3.1.1 Mental Models 10
 - 3.1.2 Transparency 10
 - 3.2 Design Methodologies..... 11
 - 3.2.1 User Centered Design..... 11
 - 3.2.2 Participatory Design 11
 - 3.3 Design Methods 12
 - 3.3.1 User Interviews..... 12
 - 3.3.2 Stakeholder Interviews..... 12
 - 3.3.3 Unstructured Interviews 12
 - 3.3.4 Contextual Inquiries 12
 - 3.3.5 Workshops..... 13
 - 3.3.6 Paper Prototyping 13
 - 3.3.7 Evaluation with Users..... 13

3.4 Related Work.....	14
3.4.1 Problems of User Centeredness.....	14
3.4.2 Perceptions of User Involvement.....	14
4 Method.....	15
4.1 The Users of Hydra	15
4.1 Information Collection	15
4.1.1 Interviews.....	15
4.1.2 Stakeholder Interview	15
4.1.3 Contextual Inquiry.....	15
4.2 Idea Generation.....	16
4.2.1 Workshop 1: Concept and Workflow	16
4.2.2 Workshop 2: Visualization.....	16
4.2.3 Workshop 3: Functionality for Troubleshooting.....	16
4.3 Usage Tests.....	16
4.3.1 Discussions around Sketches and Wireframes	16
4.3.2 Paper Prototyping	17
5 Design Development	18
5.1 Concept and Workflow.....	18
5.1.1 Required Core Functionality.....	18
5.1.2 Problems with Existing Interfaces	18
5.1.3 Expectations from the Users	18
5.1.4 Structure.....	19
5.1.5 Helpfulness	19
5.1.6 Interaction	19
5.2 Pipeline Visualization	20
5.2.1 The Non-Linear Pipeline of Hydra	20
5.2.2 Visualization of Stages.....	20
5.2.3 Conditions and Dependencies.....	20
5.3 Functionality for Testing.....	24
5.3.1 Test Driven Pipeline Development.....	24
5.3.2 The Users Needs.....	24
5.3.3 Troubleshooting	24
6 Result.....	26
6.1 Requirements	26

6.1.1 Functional Requirements	26
6.1.2 Usability Goals	26
6.1.3 User Experience Goals.....	26
6.1.4 Environmental Requirements	27
6.1.5 Data Requirements from Hydra	27
6.2 Design Solution.....	28
6.2.1 Structure.....	28
6.2.2 Pipeline Visualization	29
6.2.3 Tools	30
6.2.4 Interaction	34
7 Discussion	36
7.1 Participatory Design with Expert Users.....	36
7.1.1 Experts are Providing Knowledge	36
7.1.2 Expertise versus Innovation	36
7.2 The Interface	37
7.2.1 Reception	37
7.2.2 Adaption for Experts	37
7.2.3 New Ideas.....	37
7.3 Future Work	38
8 Conclusion	39
9 References.....	40

1 Introduction

This chapter describes the goals and focus areas of this project.

1.1 Aim

The goal of this project was to design an administrator interface for a pipeline administrator tool named Hydra, using user centered and participatory design methods.

In order to create a design suitable for the users, pipeline administrators at the company Findwise, their needs and desires were captured. The interface was then designed with the goal to satisfy the users by fulfilling these requirements.

1.2 Research Question

- Is participatory design a suitable design method when working with expert users?

1.3 Limitations

1.3.1 Focus on the Design Process

This work did not result in any functional implementation of the interface. Instead the focus was set on the design process and exploration of how it can be adapted to design for expert users. The reason for this was that the actual users of the system were available to participate in the design process, which created an opportunity to examine if participatory design was a suitable method when working with expert users.

1.3.2 Solid Concept Rather Than Details

The goal of this work was not to create a detailed description of the interface design but instead work out a solid concept in cooperation with the future users. The design was supposed to include the main layout, a good way to visualize the non-linear pipelines, an efficient way to interact with the interface as well as new functionality to help the users perform their work in a more effective way.

Since the result of this work from Findwise's point of view will be seen as a suggestion for how to solve problems discovered, a well-defined concept with motivations would be of higher value than detailed descriptions of every element of the interface and every possible interaction put together without any deeper reflection.

1.3.3 Innovation versus a Realizable Solution

At the beginning of this project the Hydra system was not entirely implemented. The foundations of the system, possibility to set up a pipeline with stages and modify their configuration, was there but other functionality such as how to set up the domain model (see section 2.3) was not yet set. A problem with this was that it could have been the case that the parts of the design were going to be incompatible with the final version of Hydra. On the other hand there was an opening for exploring different design ideas without being restricted by the underlying technology. Therefore the choice was made not to think about implementation related problems and design an interface that followed the foundational concept of Hydra, but without further restrictions.

2 Background

This chapter describes the company where this project was executed, their field of work, technical information about Hydra and its functionality and the administrator interfaces of tools used today by the future users of Hydra.

2.1 Enterprise Search

This project has been executed at Findwise, a company that works in the field of enterprise search. Their aim is to make their customer’s information easily accessible for both employees and customers (Findwise.com, 2012). The practice enterprise search can be described as making different resources within a company or organization searchable (Aim.org, 2012). Resources can here be databases, intranets, webpages or other digital repositories. The following section contains a brief description of the process of making data searchable within enterprise search.

First a *content index* is created by crawling directories and webpages or extracting information from databases and other repositories. The index then contains copies of all the *documents* in the source. Documents can here be e.g. web pages, data base entries, persons, events in a calendar or text documents. The content index must be updated when changes are made to the resources. When a search engine is connected the content index the information within it becomes searchable. A user can then enter a search query into the graphical search interface, and the matching results will be presented (see figure 1).

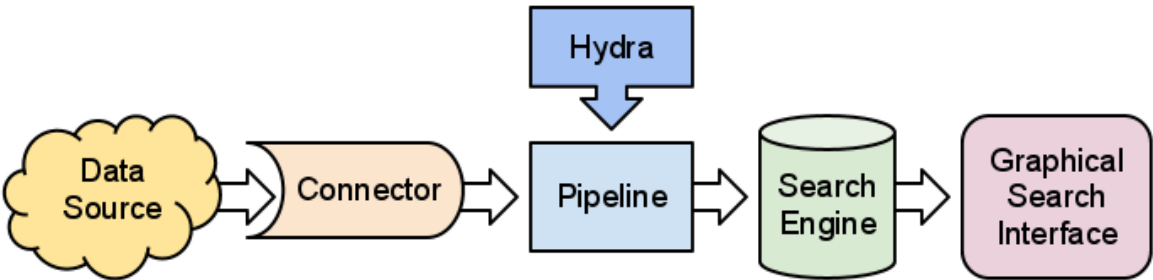


Figure 1: The structure of enterprise search solutions.

2.2 Document Processing Pipelines

One way to increase the quality of a search solution is to make it possible for the users to narrow their search and use facets for filtering the result. In order to do this *metadata* has to be extracted from the documents of the content index. Metadata can for example be the author of a document, the date when it was produced, where it was fetched from or entities mentioned in texts like locations, persons or phone numbers.

The metadata is extracted by running the documents in a content index through something called a *document processing pipeline*. A pipeline consists of a set of *stages*. Every stage has a specific task, for example this can be to extract which section of a web page that a document belongs to or to transform the formatting of a text.

The metadata is written to *fields* connected to a document. These fields are then used by the search engine that matches them to the user’s entered search query. After a document has passed through the pipeline the content of the fields are written to the content index.

2.3 Hydra

Findwise is developing a new administrator tool for document processing pipelines called Hydra, in order to providing their employees with a customized and more powerful system. A fundamental idea behind Hydra, which differs from other tools, is that the development of pipelines should be test driven. The users will set up something called a domain model, which is a set of rules of how they want their output data to look. Then stages can be added to the pipeline and configured in order to make the output match the model. The rules of the domain model have to be fulfilled in order to send the documents to the search index. If there are documents that don't fit the model, they will stay in the pipeline until the user has fixed the problems. The idea is that this will help to ensure the quality of the indexed data.

Another thing that is different from existing tools is that the stages of the pipelines don't have to be placed linearly. Instead all stages are running independently and relations between stages can be created by adding conditions of what previous actions that are required in order for a step to start processing (see figure 2). A document can be processed by different stages simultaneously if they are editing different fields. This will make the system more powerful since stages can be running on different servers.

Two types of conditions can be set up to determine if a document should be processed by a stage. One is which other stages that must have been processing the document before. The other type of condition checks if the document is suitable for the stage. This can be if some field in the document exists or not, or if a field contains a specific value or not.

The stages in a pipeline are independent and have no knowledge of each other. When a document has been processed by a stage, that document will be flagged that it has been processed by that specific stage. Other stages can then look at the flags of documents to determine if they can process them. This means that later stages can have conditions that say that another stage must have processed the document before it will run, but a stage cannot send a document to another stage.

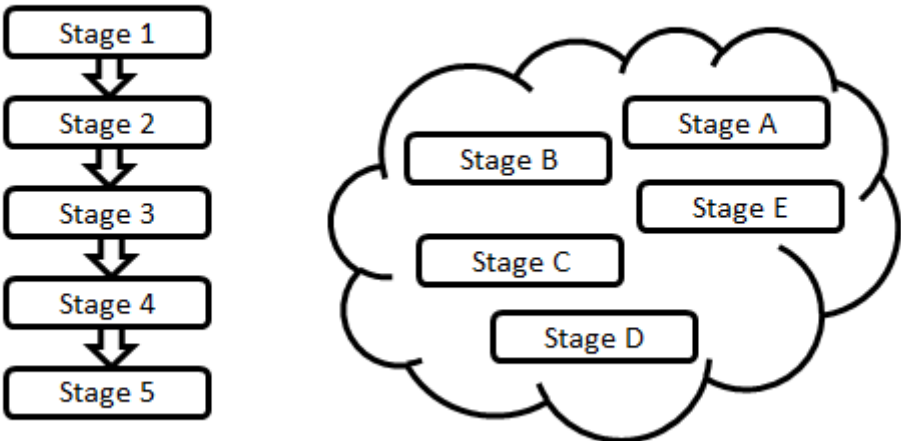


Figure 2: A linearly structured pipeline to the left, and the pipeline structure of Hydra to the right.

2.4 Interfaces of Existing Pipeline Administrator Tools

This section contains descriptions of the interfaces of two pipeline administrator tools, which are the most frequently used ones for building and maintaining pipelines at Findwise today.

2.4.1 OpenPipeline

OpenPipeline is an open source tool developed by Dieselpoint Inc (Dieselpoint Inc, 2010). It provides functionality for crawling, parsing and analyzing documents for solutions in enterprise search. The document processing pipelines are in this tool administrated through a web based interface.

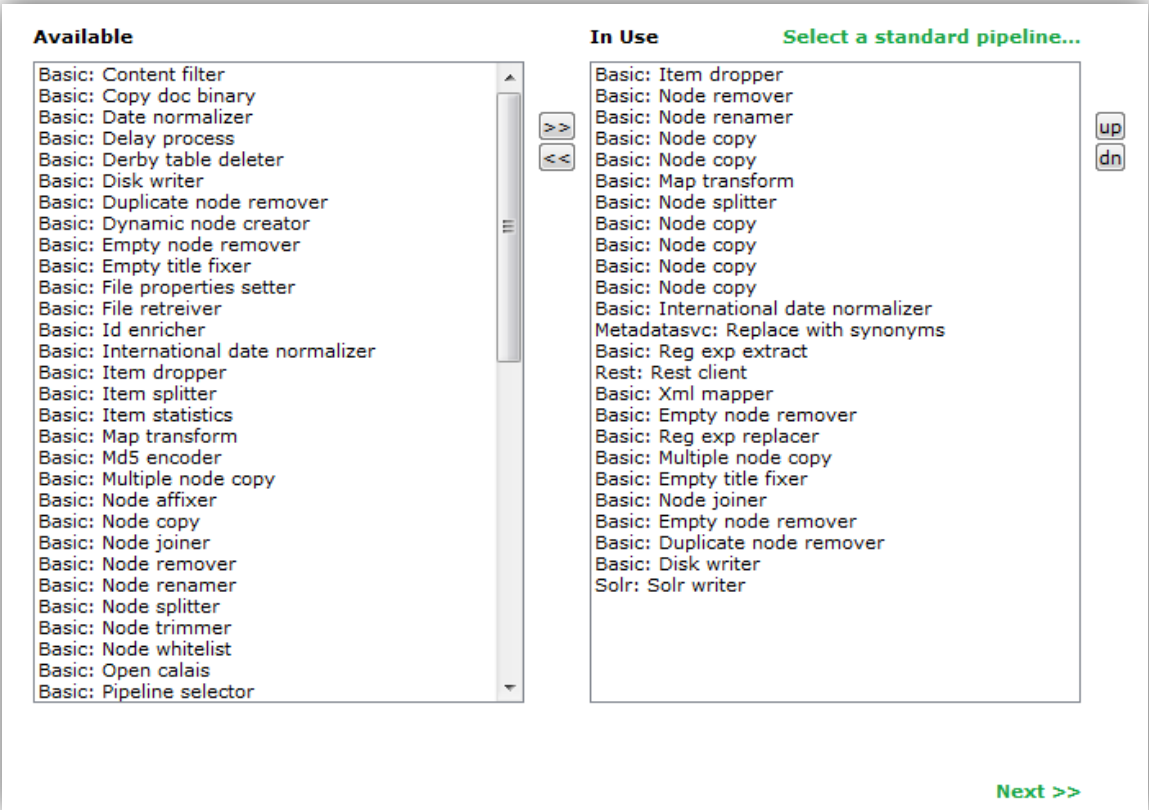


Figure 3: The setup view of the OpenPipeline interface.

When the user wants to create a pipeline he opens the *setup view* where all available stages are shown in the “available” list to the left, and the stages included in the current are shown in the “in use” list to the right (see figure 3). Stages are included in or excluded from the pipeline by selecting the stage to include or remove and then clicking the arrows in the middle of the screen. The stages in the pipeline can be reordered by selecting a stage and then press “up” or “dn”. If the users wants to configure the stages he clicks “next” in the bottom right corner of the screen and another view appears (see figure 4), this also saves changes made to the pipeline.

Configure Stages

Configure stages here. Stages that need to be configured have a "config" link next to the name.

Stage #	Name	Date normalizer
0	Basic: Item dropper (config)	<p>Normalizes dates to the format as accepted by Apache Solr: <code>yyyy-MM-dd'T'hh:mm:ss'Z'</code> or some other format if specified. Assumes the locale is UK</p> <p>Date configuration</p> <p>Field(s) to normalize:</p> <input type="text" value="creationdate revisiondate"/> <p>The values of these nodes will be tested against all the date formats defined below</p> <p>Outputformat</p> <input type="text"/> <p>The dateformat to normalize to, the default is the solr standard <code>yyyy-MM-dd'T'hh:mm:ss'Z'</code></p> <p>Date format(s):</p> <input type="text" value="yyyy-MM-dd
yy-MM-dd
yyyy-MM-dd HH:mm
yyyy-MM-dd HH:mm:ss
EEE MMM dd HH:mm:ss zzz yyyy"/> <p>Date format patterns to normalize</p> <p>Purge illegal dates: <input type="checkbox"/></p> <p>Empties the content of fields holding invalid dates.</p>
1	Basic: Node remover (config)	
2	Basic: Node renamer (config)	
3	Basic: Node copy (config)	
4	Basic: Node copy (config)	
5	Basic: Map transform (config)	
6	Basic: Node splitter (config)	
7	Basic: Node copy (config)	
8	Basic: Node copy (config)	
9	Basic: Node copy (config)	
10	Basic: Node copy (config)	
11	Basic: International date normalizer (config)	
12	Metadatasvc: Replace with synonyms (config)	
13	Basic: Reg exp extract (config)	
14	Rest: Rest client (config)	
15	Basic: Xml mapper (config)	
16	Basic: Empty node remover	
17	Basic: Reg exp replacer (config)	
18	Basic: Multiple node copy (config)	
19	Basic: Empty title fixer (config)	
20	Basic: Node joiner (config)	
21	Basic: Empty node remover	
22	Basic: Duplicate node remover (config)	
23	Basic: Disk writer (config)	
24	Solr: Solr writer (config)	

Figure 4: The *configuration* view of the OpenPipeline interface.

Now the stages in the pipeline are shown in a numbered list to the left, which represents the order in which they will process the incoming documents (here also called “items”). The stages are represented by the name of the class they are instances of, which means that stages have the same name if they are the same type of objects.

The configuration of a stage is opened by pressing “(config)” to the right of the stage in the list. It then opens up to the right on the screen and the configuration can be typed into the textboxes. To the right of the boxes there are descriptions of what to write in them. Here the configuration of “International date normalizer” is open. To save the configuration the user clicks “next” in the lower left corner (not visible in this screenshot).

2.4.2 FAST ESP

FAST ESP (Enterprise Search Platform) is developed by FAST, a Microsoft Subsidiary (Microsoft, 2011). ESP is a development platform for creating indexes of searchable content. It contains a web based interface for administrating document processing pipelines.

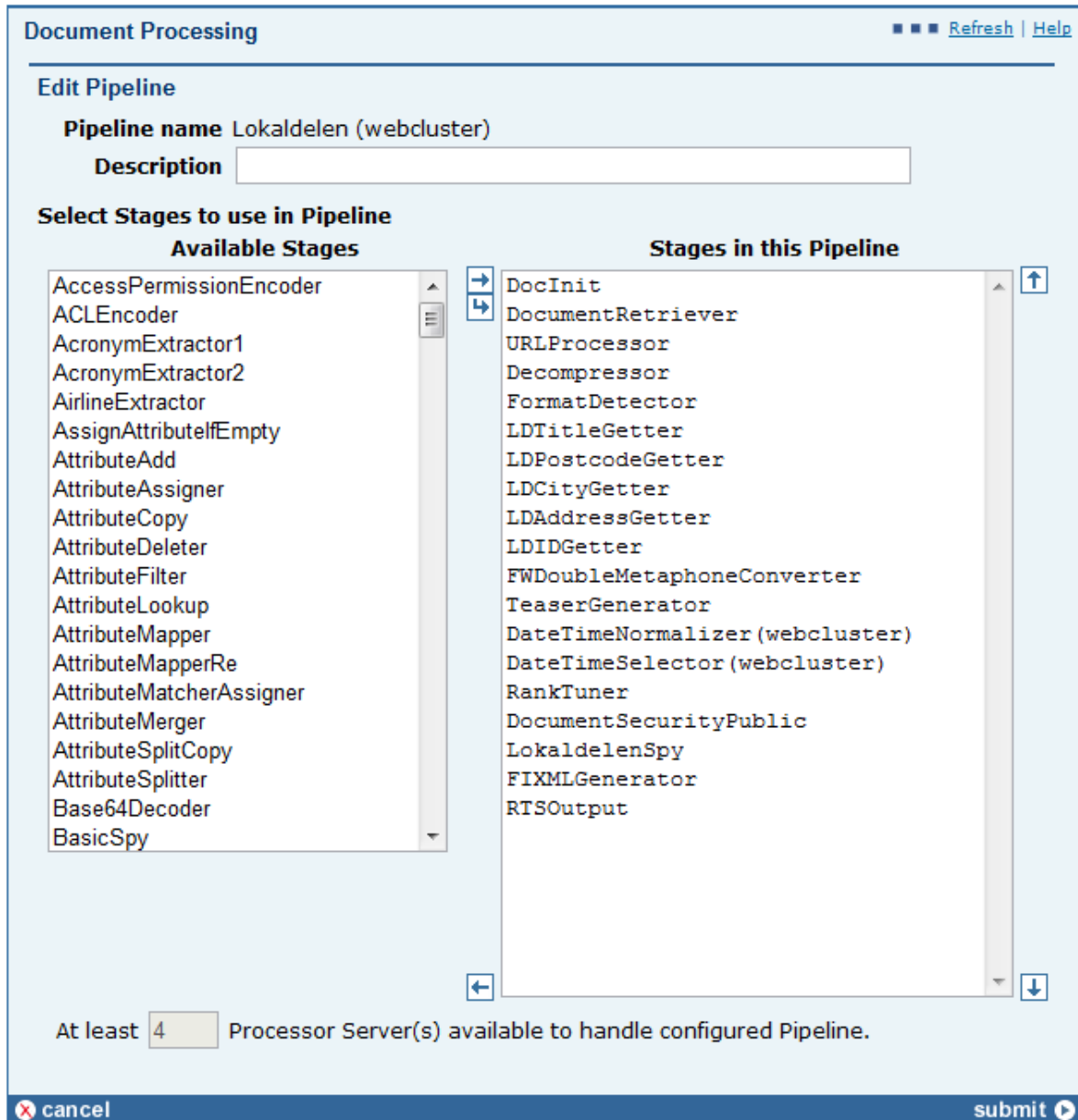


Figure 5: The setup view of the FAST ESP interface.

The setup view of ESP is very similar to the one of OpenPipeline (see figure 5). In the column to the left all available stages are listed and the right list contains the stages included in the current pipeline. The user can add or remove stages from the pipeline by pressing the arrow buttons between the columns. The stages in the pipeline can be reordered by selecting a stage and move it up or down by using the arrows to the right on the screen. Changes are saved by pressing "submit" in the lower right corner of the screen.













































Custom Stages			
Stage Name	Description	Type	
AcronymExtractor1	Detects acronym definitions.	general	   
AcronymExtractor2	Extracts acronyms, based on the base form of previously detected acronyms.	general	   
AirlineExtractor	Extracts names of airlines.	general	   
BasicSpyAppendThesis	Example that shows how a pipeline stage can obtain and act upon configuration data given to it by the user via the admin GUI.	general	   
BodySizeFilter	This filter discards documents with empty bodies.	general	   
CarExtractor	Extracts names of cars.	general	   
CompanyExtractor1	Extracts names of companies, based on general patterns and dictionaries.	general	   
Default Stages			
Stage Name	Description	Type	
AccessPermissionEncoder	Parse docadxml field to generate docacl, docaclsystemid and docsiggroups fields	general	 
ACLEncoder	Encodes entries in document ACLs	general	 
AssignAttributeIfEmpty	Assigns a constant value to a field if it is empty.	general	 
AttributeAdd	Add document attributes	general	 
AttributeAssigner	Assign a constant value to a document attribute The attribute name and value are defined in configuration parameters.	general	 
AttributeCopy	Copy document attributes	general	 
AttributeDeleter	Delete attributes from a document	general	 
AttributeFilter	Drop documents based on attribute values	general	 

Figure 6: Lists of stages in the FAST ESP interface.

In ESP the stages cannot be configured inside the pipeline. Instead the stages are configured in a different view containing all available stages. This means that if more than one pipeline contains an instance of the same stage, all of them will be affected to changes made to the configuration of that stage.

The list of stages are divided into two sections (see figure 6). The one on top displays all preconfigured stages (custom stages) and the lower one contains stages that have not been configured (default stages). By pressing the “+” sign to the right of a stage name the user can create a new instance of that stage. If an instance of a default stage is created, it will be placed among the custom stages. By pressing the icon with a document to the right of the names of the custom stages, next to the one with the magnifying glass, the configuration of that stage can be changed.

Create Stage
Refresh | Help

Stage Info

Attribute	Value
Class	Matcher
Name	<input style="width: 90%;" type="text"/>
Description	<div style="border: 1px solid #ccc; padding: 5px; min-height: 100px;"> Extracts dates. </div>

Configuration

Parameter	Value	Type
rename	<input style="width: 90%;" type="text"/>	string
matcher	en:linguistics/extractors/configuration.dateextractor.e	string
phrases	<input style="width: 90%;" type="text" value="0"/>	string
dispatch	<input style="width: 90%;" type="text" value="language"/>	string
separator2	<input style="width: 90%;" type="text" value="/"/>	string
filter	<input style="width: 90%;" type="text" value="*:illegal"/>	string
guard	<input style="width: 90%;" type="text"/>	string
meta	<input style="width: 90%;" type="text" value="type:semantic"/>	string
separator	<input style="width: 90%;" type="text" value=";"/>	string
output	<input style="width: 90%;" type="text"/>	string
byteguard	<input style="width: 90%;" type="text"/>	string
input	<input style="width: 90%;" type="text" value="title body xml"/>	string
type	<input style="width: 90%;" type="text" value="date"/>	string

cancel
submit

Figure 7: Configuration view of the FAST ESP interface.

The configuration will be displayed in another view (see figure 7). Here the configuration of an instance of the class “Matcher” is displayed. The user can fill in the desired configuration in the text boxes, and also give the instance a specific name. Changes are saved by pressing “submit”, or discarded by pressing “cancel”.

2.5 Tools

This section describes the tool used for sketching and prototype development in this project.

2.5.1 Pencil Project

Pencil Project is an open source product for creating GUI prototypes (Evolus, 2010). The tool makes it possible to create sketches of interfaces quickly, by providing common interface elements that can be included and transformed. It is also possible to create custom shapes and save them in a special library, which saves a lot of time. Elements can be linked to other sketches and exported as html, which provides a very simple way of creating low-fidelity prototypes. Figure 8 shows a screenshot of the program.

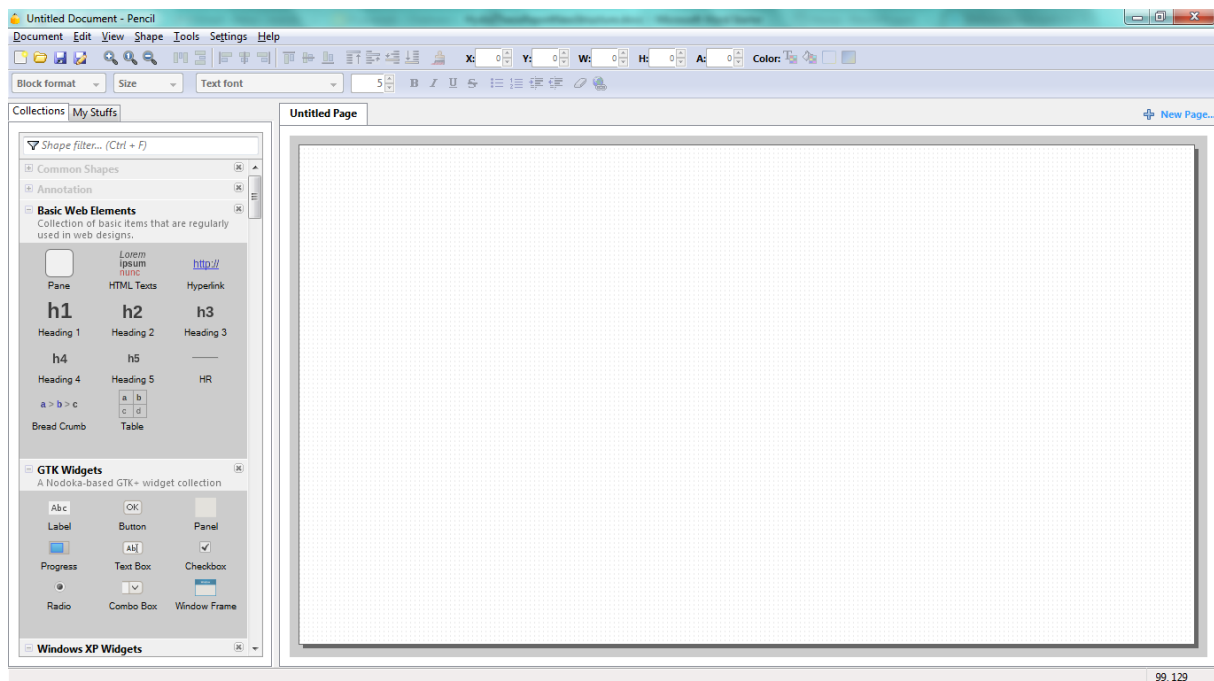


Figure 8: The interface of the Pencil GUI prototyping tool.

2.5.2 Microsoft Lync

Microsoft Lync is a tool for instant messaging adapted for usage in corporate environments (Microsoft, 2010). The tool provides functionality for chats, audio and video conferences for a pair or groups of people. Lync is used for internal communication at Findwise.

3 Literature Review

This chapter explains design theories and methodologies as well as related research performed on user participation in development processes.

3.1 Cognitive Frameworks

When designing an interface it is important to adapt it to the users' level of experience and knowledge. This section describes theoretical frameworks related to this topic.

3.1.1 Mental Models

The functionality of a system and how it is interpreted by a user can be described with different types of models. The *implementation* (or *target*) *model* of a system describes how it actually works, while a *mental model* is the user's internal representation of how a system works based on previous experience and knowledge (Staggers & Norcio, 1993). A *conceptual model* can also be developed as a tool for helping the user to understand a system.

In the field of interaction design, one of the tasks assigned to the designer is to create a conceptual model that the user can understand (Interaction-Design.org, 2009). If it is well executed it should support the development of a correct mental model when the user interacts with a system.

Research has shown that the correctness of a user's mental model and performance have a correlation, especially when it comes to more complex tasks (Staggers & Norcio, 1993). Users with a well-developed mental model are more likely to be better problem solvers than users with a weaker model, since they can associate symptoms with causes according to their model.

3.1.2 Transparency

In the field of software design the concept *transparency* can be described as in what extent the user can see, understand and make use of the underlying mechanisms of a system (Löwgren & Stolteman, 1998). A system with a low level of transparency can be described as a black box, where the user gives the system some kind of input and receives an output without knowing what functionality that is hidden within the box. The user must then create an own mental model of the relation between input and output. Systems on the other end of the transparency scale can instead be seen as boxes of glass, which allows the user to see the underlying functionality. The user can then see the actual relationship between input and output, and can therefore understand the underlying process.

For a designer it is important to set a correct level of transparency. A non-transparent design is secure and controllable but limits the user's freedom, while a transparent solution provides flexibility but limits the designer's power of controlling the usage of the design. In general a less transparent design is suitable for novice users, who want to learn and get quick results, while a more transparent design is better for expert users who demand a higher level of precision and level of freedom.

3.2 Design Methodologies

3.2.1 User Centered Design

In design processes with a user centered approach the users and their goals should be the driving force instead of just technology (Sharp, Rogers & Preece, 2007): p 425-429). A design should support the users in their activities instead of putting up constraints.

During the beginning of a process it is important for the designer to develop an understanding of the users by exploring their characteristics and how they perform their tasks. Usability and user experience goals should be documented and the design should then be built to meet the needs of the users. During the development process the design ideas should be tested with users in order to measure the goals and to get important feedback.

The design is often developed in iterations which mean that the result of an evaluation of a prototype and new problems discovered during the evaluation are taken into the next version of the design. In this way the quality of the design is improved in every iteration, and it is assured that it meets the needs of the users.

3.2.2 Participatory Design

Design processes can have different levels of user involvement (Sharp, Rogers & Preece, 2007: p 419-424). In some projects users are included in the design team to work full time as a representative for the users, in other processes the user participates through workshops and continuous exchange of information. Participatory design is an approach where the users are highly involved in the design process. The main idea of the methodology is to let the people that are going to use a system to take part of the process of designing it (Schuler & Namioka 1993).

Participatory design has its roots in the late 1960's Scandinavia (Sharp, Rogers & Preece, 2007). There were two main factors that lead to development of the methodology. First, there was a need for communicating complex system information. Second, the labor union movement fought for the worker's right to democracy and control over their own work situation.

An advantage of including the users in the design development is that the users can provide valuable knowledge and experience in a field that can be unfamiliar to the designer (Faulkner, 2000). Another benefit is that the users are more likely to feel involved and realize that a system is designed for them they are therefore more likely to care about the end result (Sharp, Rogers & Preece, 2007). Participation creates a sense of ownership so that the users are more likely to accept a new design when it is released. It also keeps the users' expectations at a realistic level, since they can follow the development from an early stage.

An objection against user involvement, especially in projects with shorter time-scales, is that it takes time to organize and manage it (Sharp, Rogers & Preece, 2007). This time could instead have been spent on development. Another risk is that a user that has been involved in the process for too long is being accustomed to the situation and loses the ability to be representative for the other users. This can be avoided by alternating between different user representatives.

3.3 Design Methods

These design methods were used in order to collect information about users, their activities, similar existing systems and the implementation of Hydra as well as during the development and evaluation of the interface.

3.3.1 User Interviews

An interview is a form of qualitative research that can help the designer to understand otherwise complex situations (Cooper et al, 2007). Interviewing users can provide valuable information about their attitudes, behavior and skills as well as information about their relation to existing systems. In a development process it can help the designer to identify the users' goals and problems encountered in their current way of solving the tasks as well as understanding their mental model of the activities. This will help the designer to make informed decisions and develop a design based on research results instead of personal preferences.

3.3.2 Stakeholder Interviews

When developing a design solution it is important to understand the material there is to work with, its limitations and opportunities (Cooper et al, 2007). The research for a new product should start by defining the goal for why it is developed. A stakeholder is here someone with authority who is responsible for the product.

Interviews with stakeholders are held to capture the vision of a product, which must be incorporated with the vision of users and customers. The budget and schedule must be explained in order to choose a feasible design method and to set the bar at a reasonable level. An understanding of the technical constraints and opportunities helps to steer the design in the right direction. It is also important to specify the business goals, why the product is being developed.

These interviews helps the designer to create a product that is not only adapted for the users, but also meets the expectations and demands of the company. It is important that all people involved believes in a developed solution. They should preferably be asked for their opinions early in the process, otherwise there is a risk that the input arrives as critique later on in the process.

3.3.3 Unstructured Interviews

In unstructured interviews the questions asked are open ended, which means that the interviewer cannot expect answers on a certain form (Sharp, Rogers & Preece, 2007). The benefit of this method is that the interview is more likely to go into the depth of a topic, since the interviewee have more room for explanatory answers. The interviewer can choose to ask follow up questions on topics that are especially interesting. A drawback is that it can be hard to compare results from different interviews, and keep the focus straight within one.

3.3.4 Contextual Inquiries

The users of a system sometimes find it hard to describe an activity they perform, especially when it is taken out of context (Cooper et al, 2007). Therefore it is good to observe the users when they are in a normal work situation. A contextual inquiry is a form of user observation with a more collaborative approach. The designer and the user then works in a more

collaborative way and are encouraged to discuss features and problems in the system. It is important that the designer understands what the user is saying and does not make assumptions without verifying them with the user. The designer must also keep the focus on areas of interest without interrupting the user or be too controlling.

3.3.5 Workshops

A workshop can take many different shapes. In participatory design workshops are often used to help different parties in a process to communicate and commit to the same goals (Muller & Druin, 2010). Participants in a workshop are often introduced to activities that are normally not part of their work assignments. It is therefore important to give them a way to express their opinions and thoughts to be able to share their knowledge.

There are many methods for how to perform a workshop session. Some of them are focusing on defining problems and other on generating ideas and finding solutions. A benefit of holding a workshop is that it often produces some kind of artifact. This will serve as part of the documentation and can be a stepping stone for the next iteration.

A workshop can bring people from different backgrounds and with different levels of experience together, which will help to create well thought-out ideas that have been seen and evaluated from different points of view. When new ideas and concepts are participatory developed by the users they will probably be valuable and practically useful since the people that have an interest in the result has been given a chance to provide their input.

3.3.6 Paper Prototyping

When designing a product it is important to define a framework for the design before going into detail (Cooper et al, 2007). This will help to create a stable foundation that a design that needs of the users can be built upon. Paper prototyping is a suitable method for testing early design concepts of user interfaces. A prototype here consists of a set of cut out pieces of paper or post-it notes that represents the elements of the interface. During a test session the user typically receives a task to perform and the moderator's assignment is to act as the computer and control the interface.

The benefit of this kind of low-fidelity prototypes is that they can be made quite quickly; different ideas can be tested and thrown away without losing very time consuming work. It is also easier to get honest feedback from the users if they see that the prototype is simple, with a more refined prototype they may think that it is too late to make major changes to the design and feel obligated to say good things in order to make the designer happy.

3.3.7 Evaluation with Users

It is important to consider the feedback from all users regarding a design, but every suggested change cannot be realized at the same time. The designer must determine which ideas that are suitable and which are not. It is important that the designer then communicates back to the users and explains why certain changes are unfeasible (Gulliksen & Göransson, 2002). Otherwise there is a risk that the users feel that their opinions do not matter and becomes less motivated to continue to participate in the design development.

3.4 Related Work

This section describes articles on research performed on user involvement in software design and development processes.

3.4.1 Problems of User Centeredness

The authors of the article *Don't Underestimate the Problems of User Centeredness in Software Development Projects – There are Many!* have done research on how user centeredness affects software development processes, in different domains and for different user groups (Heinbokel et al, 1996). They focus on two types of processes; some with *user participation* and some with *user orientation*. User participation here means that at least one user have been part of the development team, while user orientation means that the developers had the user in focus and produced software adapted for the needs of the user. 29 software development projects in Germany and Switzerland participated in the study.

The results of the study show that projects with a high level of user centeredness were less successful both regarding the process and the quality of the developed product. Projects with user participation showed problems with innovation, flexibility and effectiveness. The user oriented ones led to low effectiveness and had problems with team interaction. Problems with participation affected the whole outcome of a project, while problems with user orientation were more centered on ongoing activities. These results were based on data collected from the developers, not the users.

The authors discuss the reason for these results and suppose that the problem can be in the relationship between developer and user. If the users express new ideas late in the design process it is hard for the developers to realize their desires. The developers could also have found it stressful to incorporate the users' demands, especially if they had limited knowledge of how to put the user centeredness into action.

3.4.2 Perceptions of User Involvement

In the article *User Involvement During Information Systems Development: a Comparison of Analyst and user Perceptions of System Acceptance* the authors investigate if there is a difference between what users and system analysts think about how user involvement affects the outcome of information system development processes (Foster & Franz, 1999).

They studied 87 cases, development of software systems that included both system analysts and users. The analysts answered a questionnaire about their perceptions of the user's involvement and the system acceptance. The users evaluated their own contributions to both the design and implementation phases as well as the usefulness of the system.

The results of the study show that there was a strong connection between the users' rating of their own involvement and their perceived usefulness of the systems. The same connection was there between the analysts' rating of the users' involvement and the analysts' perceived usefulness of the system. However there was no relationship between the two groups. If the participation of the users was ranked high by one group, the usefulness of the system was not considered to be that high by the other group. The authors state that the reason for this probably is that there is a difference in how user perception is perceived by analysts and by the users themselves. They also stress this does not matter for the overall success, user involvement still helps to create more usable systems.

4 Method

4.1 The Users of Hydra

The end users of the Hydra administrator user interface will be people working with pipeline configuration at Findwise. By the definitions of user groups in (Faulkner, 2000) they are direct users, since they sit in front of the screen and apply transformations to the pipeline directly. They are mandatory users because using the system is necessary in order to carry out their job assignments. They will also soon become expert users since, when it is implemented; they will use the software in their everyday work.

Most of the users have a computer science or software engineering background, which also makes them experts in computer oriented areas. By some means they are not only expert users, but also experts in the medium in which the design will be built. The statement “Users don’t understand Boolean Logic” (Faulkner, 2000) is certainly not true in this case.

During the design process a group of users were highly involved to take advantage of their experience and knowledge of pipeline development. Every employee working with pipelines at the Findwise Gothenburg was invited to participate. At first their needs and opinions were captured and later they participated in the development and evaluation of the design.

4.1 Information Collection

4.1.1 Interviews

Unstructured interviews were held with twelve of the future users of Hydra. Ten were interviewed at their worksite at the Findwise office in Gothenburg and two were interviewed from the office in Stockholm through the video conference tool Lync.

The users were asked questions about their work process and if they encountered any problems when carrying out these tasks in existing pipeline administrator interfaces. They were also asked to describe what functionality that could make their work easier or more pleasant. The open ended questions made it possible to steer the interviews in different directions and focus on areas where the user gave extra interesting input, since the users were more or less interested in different areas.

Some of the users were very new to the concept behind Hydra so it had to be explained to them during the interviews. These users found it hard to express their desires of the new interface. Others were more engaged and had many ideas of how the new interface could make it easier for the users to accomplish their tasks.

4.1.2 Stakeholder Interview

The lead developer of Hydra was also interviewed. He was asked to explain the architectural structure of the system and the functionality it provided.

4.1.3 Contextual Inquiry

A user was asked to explain the process of setting up a pipeline while sitting by his desk in front of his computer screen. He showed the structure of the interface and talked about his own opinions of the functionality. He also showed problems with the interface and pointed

out things that were complicated and hard to accomplish. The sound was captured for later listening and analysis.

4.2 Idea Generation

To generate ideas to the new design participatory workshops were held in conference rooms at the Findwise office in Gothenburg. They were each focusing of one of the following topics; *concept and workflow*, *pipeline visualization* and *functionality for testing*. The number of participants differed and depended on how many developers that was available at the time.

4.2.1 Workshop 1: Concept and Workflow

This workshop focused on defining a concept for the interface to create a framework for the rest of the design development. Areas covered were the structure of the interface and how to interact with objects on the screen. The workshop also had the purpose to start up the design work and develop a contact with the users.

The workshop was held on two occasions with three participants each time. A sketch of a design where the whole interface was represented within one view was brought to the workshop, together with colored pens and papers. The idea was presented to the participants and they were encouraged to give their opinions and concerns. They were also given the material to show their own ideas of how the interface could look, without going into detail.

4.2.2 Workshop 2: Pipeline Visualization

This workshop focused on visualization of the non-linear pipeline and how to present conditions and relations between stages. Five users participated in this workshop.

The users were handed papers and asked to draw their own ideas of how the pipeline could be visualized. The idea was that they then should give the sketch to the person next to them to improve the design. When the next person received it there was a problem since he did not understand the idea behind the drawing. Instead the creator had to explain the idea. All ideas were discussed in the group, to capture their strengths and weaknesses. The users were also encouraged to draw and write down new ideas that arose.

4.2.3 Workshop 3: Functionality for Testing

The last workshop focused on functionality for testing the pipeline and ways of displaying its performance. Five users attended this workshop.

At first the users got some time to think of functionality that would help to make their work easier. The participants were then divided into two groups that each developed one design suggestion. They then presented the suggestion for the other group followed by a discussion around the advantages and drawbacks of the ideas.

4.3 Usage Tests

4.3.1 Discussions around Sketches and Wireframes

The ideas that the users developed during the workshop sessions were translated into sketches. The users then had opportunities to leave their comments on the ideas during discussions with one participant at the time. They were also encouraged to present their

own ideas of how the design could be made better. The feedback and ideas that the users provided were taken into consideration and the sketches were refined and evaluated again. These discussions started after the first workshop when the sketches were very simple, and continued until the end of the project when the users were shown the final design in the shape of wireframes.

4.3.2 Paper Prototyping

Paper prototyping was used to test the workflow and interactions. Some of the sketches were printed out and graphical elements in different states were cut out. The user received a smaller task to accomplish. He then tried to execute the task in the way he thought was the right way. Sometimes the user was corrected if he went in a totally wrong direction. Then it was explained to the user how the assignment was intended to be carried out. He was asked if he thought the design idea was a good, and had then opportunity to suggest how he would have wanted the interface to behave.

5 Design Development

In this chapter the development of the interface design is described, in terms of how the findings from the interviews and observation have been combined with the ideas developed during the workshops.

5.1 Concept and Workflow

This section focuses on the overall concept for the interface and how it was developed from requirements from Hydra and the users' expectations.

5.1.1 Required Core Functionality

In order to design an interface that is adapted for Hydra, the concept must support its functionality. Even though this work focuses more on innovation than creating a realizable solution, a framework that was compatible with Hydra had to be developed.

The main goal of building document processing pipelines is to extract the necessary metadata from the documents in the content index. To do this the interface must provide two basic functions. First, the user must be able to create a pipeline and include stages to it. In order to do this he must be able to find the desired stages and add them to the pipeline. Second, in order to make the stages behave correctly the interface must also provide a way to configure them.

5.1.2 Problems with Existing Interfaces

During the interview and observation the users expressed their concerns of other interfaces regarding the workflow and navigation. They managed to carry out their work assignments, but in a sometimes slow and unnecessarily complex way. (For descriptions of the interfaces of the tools used today, see section 2.4).

When the user wants to include a new stage to a pipeline there are no functions for searching for a stage, either by its name or certain functionality. Instead the user has to scroll through a list to find the desired stage, and sometimes also read their descriptions, which takes a lot of time.

To reorder a stage in the setup views the user has to select a stage and then press a button causing it to move one step up or down, this becomes very repetitive if the user wants to make big changes to the structure of the pipeline.

In both tools the setup view where the user can manage the pipeline by reordering, adding or removing stages is separated from the configuration view. This means that the users have to navigate back and forth to reach fundamental functionality, which they find tiresome.

5.1.3 Expectations from the Users

The users agreed that they wanted to spend their time making the document processing work properly, instead of extensive clicking or unnecessarily complex navigating between views that breaks the workflow. They would appreciate a more flexible solution that could be adapted to fit their current task better.

The users agreed that innovation is cool but made it clear that new solutions must provide something good in terms of functionality; they should not be included just because it looks

good. They also stated that the interface must be clear and reliable, since they have to have control over what they are doing.

5.1.4 Structure

The evaluated interfaces are very static, which can be a reason for why the users find the navigation and interaction tiresome and time consuming. To solve this problem an idea with a more dynamic interface was presented during the first workshop. The idea was to divide the interface into two areas; a workspace and a toolbox. In this way the interface would feel more like a coherent stand-alone application than of a series of different views. This would help to create a better workflow since the user can choose what information and functionality that will be visible on the screen depending on the situation.

It turned out that the users were positive to the idea of presenting the interface within one flexible view. They agreed that a space for presenting the pipeline in a more visual way was necessary to be able to understand how the stages were related to each other, but emphasized that it had to be clear and understandable.

A concern was if there would be enough space to show both the representation of the pipeline and the tools in one view. An idea of having tabs for different types of functionality was expressed, which would make it possible to use the same space for many purposes.

5.1.5 Helpfulness

One of the desires from the users was that the interface should be efficient and help them to accomplish their assignments in a quicker and smoother way. Many of them thought that the problems with finding stages could be solved easily just by adding a search function. Some of them would be happy if it would be possible to select a stage in the list of available ones and start typing which would cause the list to scroll to the desired position. Others had more complex ideas and wanted search functions with filtering for finding stages.

The solution that everybody agreed on was one with a free text search and possibility to filter the search results by the categories of stages. It was important that the search results were updated immediately so that it was quick to work with. Some of the users wanted to search the description of the stages if they were unsure of their names. They also wanted a possibility to search and clone configured stages in both the current and other pipelines.

5.1.6 Interaction

With a structure where the interface has a workspace with the pipeline represented by graphical elements it was not feasible to relocate the stages with buttons or arrows since they are not placed linearly. Drag and drop is appropriate for rearranging objects when the relations can be shown visually (Scott & Neil, 2009). This would provide a quick way to rearrange objects that would have been very hard to accomplish in another way when the objects are not structured in a list. Therefore it was explored if this could be a good way to handle the interaction within the workspace.

Some of the users liked the idea, while others would have preferred the possibility to control the interface only by using the keyboard. No solution to how this could be carried out with keyboard only was found. Instead it was decided that the drag and drop actions should be simple and quick to execute since the users wanted to avoid interaction that required fine motor skills that would take a lot of time.

5.2 Pipeline Visualization

This section describes the development of the model for visualizing the pipeline with the stages and relations between stages in form of conditions.

5.2.1 The Non-Linear Pipeline of Hydra

One thing that separates Hydra from other interfaces is that the stages are not arranged in a linear list but instead placed in a cloud with no knowledge of each other. To design a visual representation of the pipeline that will support the user in his work he must be able to understand how the stages are related to each other and in which order the stages are processing the documents.

There are two types of conditions; stage related and document related conditions. The stage conditions describes which stages that have to have processed a document before, and the document conditions describes what properties a document must have for a stage to process it.

5.2.2 Visualization of Stages

A problem that the users expressed regarding the interface of OpenPipeline was that there is no way to give custom names to the stage instances. This becomes a problem when the user has more than one stage of the same type in a pipeline. In order to separate the stages the user has to enter another view and read the configurations of the stages, which takes extra time and breaks the workflow. In Hydra all stages will have unique names which will solve that problem. The names of the stages are important to the users, and they agreed that this information should be visible in the representation of a stage. The users also agreed that the visualization should display the name of the stage type, since this would help the user to remember the functionality of the stage.

5.2.3 Conditions and Dependencies

During the second workshop the users discussed how the pipeline with stages and dependencies could be visualized. Most of the users imagined the visualization like a graph or flowchart, where the stages were represented by some sort of boxes connected by lines representing their relations.

In some ideas the conditions were separated from the stages and represented by graphical elements (See figure 9). A problem with this representation is that it is based on the documents way through the pipeline. It works well for that purpose, but since many different types of documents are processed simultaneously through the same pipeline it can be confusing. The only conditions that exist are included in the configuration of a stage and determine whether a document is ready for processing or not based on if it has been processed by some other stage or contains fields with specific values. If this condition is not fulfilled, that stage cannot tell another stage to process the document instead, since stages are unaware of each other. Because of this, many users found this type of stand-alone conditions to be a bad idea since it would make them feel less in control.

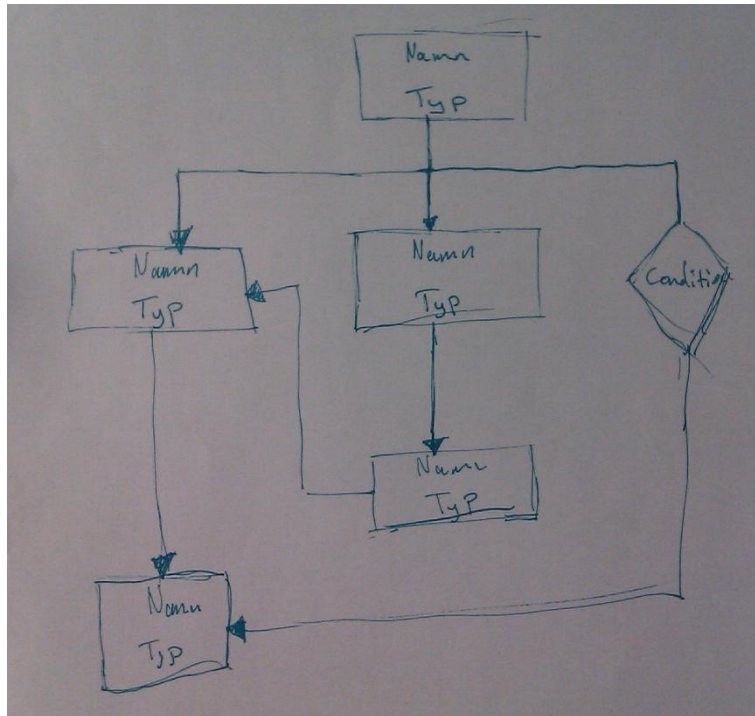


Figure 9: A sketch of a pipeline from the second workshop.

It was hard for the users to determine how big and complex the pipelines in Hydra would be, and they were afraid that very big graphs would be hard to overview. Therefore they expressed a desire to be able to group the stages in some way. They drew sketches of different idea of how to accomplish this with different kinds of barriers clusters and sub-pipelines.

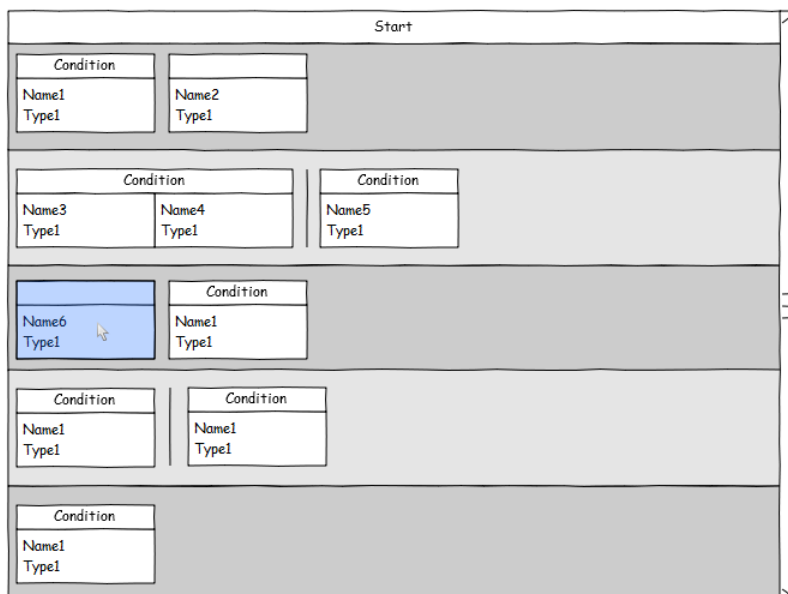


Figure 10: A sketch of a pipeline visualization with groups of stages.

One idea was to divide the stages of the pipeline into groups that had a common condition to start processing documents (see figure 10). Stages in a group (row) could either be required or not, the required ones had to do their processing before stages in the next group could start. A problem was that there was no way to create branches; the pipeline could just grow in one direction. There would be cases where the users wanted to process documents

by different stages depending on their properties and then continue the processing differently depending on what stage that got the document. This was impossible to solve with this type of visualization.

The next idea was similar to a flowchart but focused on the relations between stages rather than the documents route through the pipeline. Here a pipeline is built starting at the top of the screen and grows downwards, the same direction as in the lists of the other tools (see figure 11). The stages are connected with lines representing the order in which they could process a document. Stages with their top connected to a line must wait for the stage in connected to the upper end of the line to run before they can start processing.

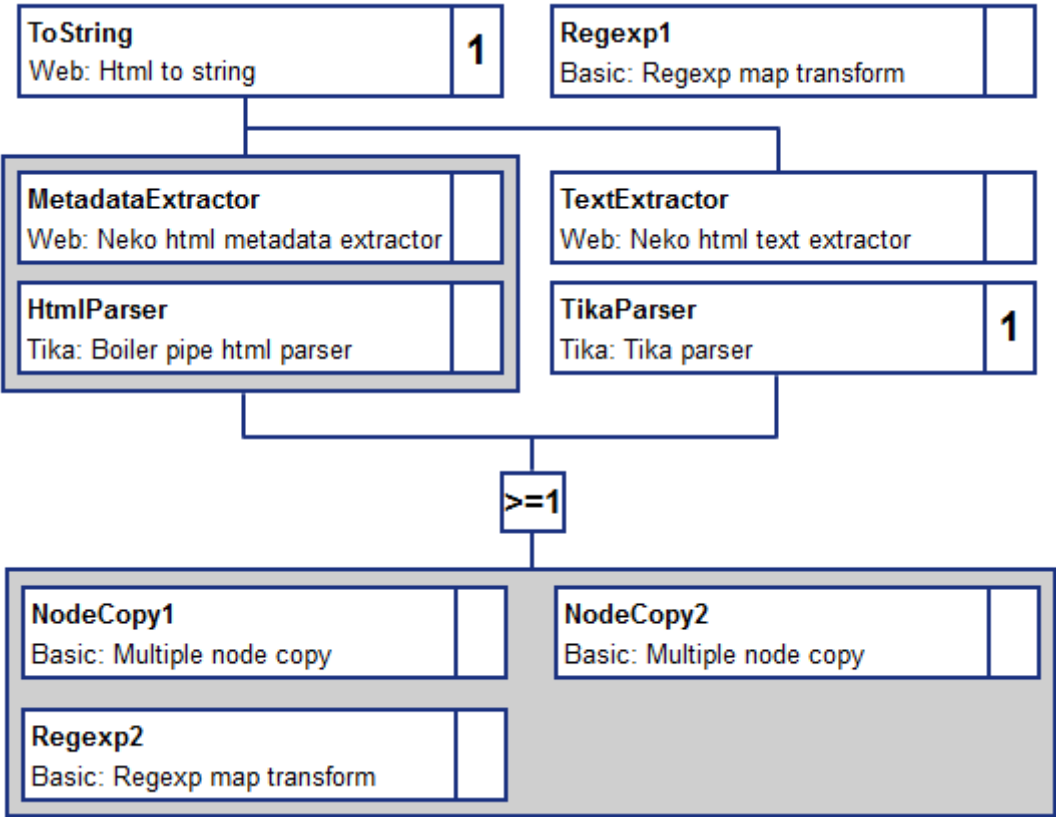


Figure 11: A pipeline visualization where conditions are represented by lines.

Document conditions are set up in the configuration of the stages, and the number of them is displayed on the boxes representing the stages. Stages that can process a document concurrently are placed inside a surrounding grey box. This limits the number of connections and makes it easier to overview the pipeline.

The design allows the user to create branches of stages for different types of documents and provides possibility to set up logical expressions of what should happen if two branches are connected; either the next stage after the merge will have to wait for all branches or just one of them.

The users appreciated this solution since it was clear and easy to follow; no major problems were discovered regarding the logic and conditions. The only concern was that some users found it hard to understand where the documents entered the pipeline since there is no visual element representing the input.

Pipelines with few parallel branches could be fit to the workspace with this representation, but a pipeline with many branches would force the user to horizontal scrolling which they wanted to avoid. A real pipeline with around forty stages was constructed using this model, and it showed that it did not scale well. It was discovered that it had many parallel branches and that most of them were only two stages deep (see figure 12). Each branch had the purpose of setting one or a few specific fields of the documents, and two branches were never merged together once separated.

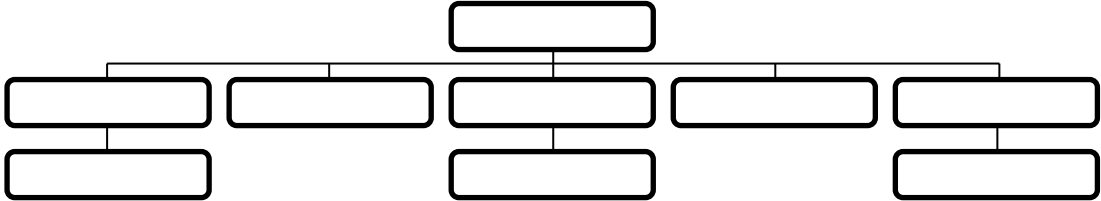


Figure 12: Structure of the test pipeline.

Since the pipeline turned out to be wide and quite shallow, it was tested to turn the structure it 90 degrees (see figure 13). The dependencies are still displayed using lines, but here they are connected to the left and right side of the stages instead of their top and bottom. A bar representing the input was introduced together with the possibility to break a line and continue the pipeline construction on the next row in order to fit the whole width of the pipeline on the workspace and avoid horizontal scrolling. Document conditions are marked out with a box attached to the left side of the stages that have any.

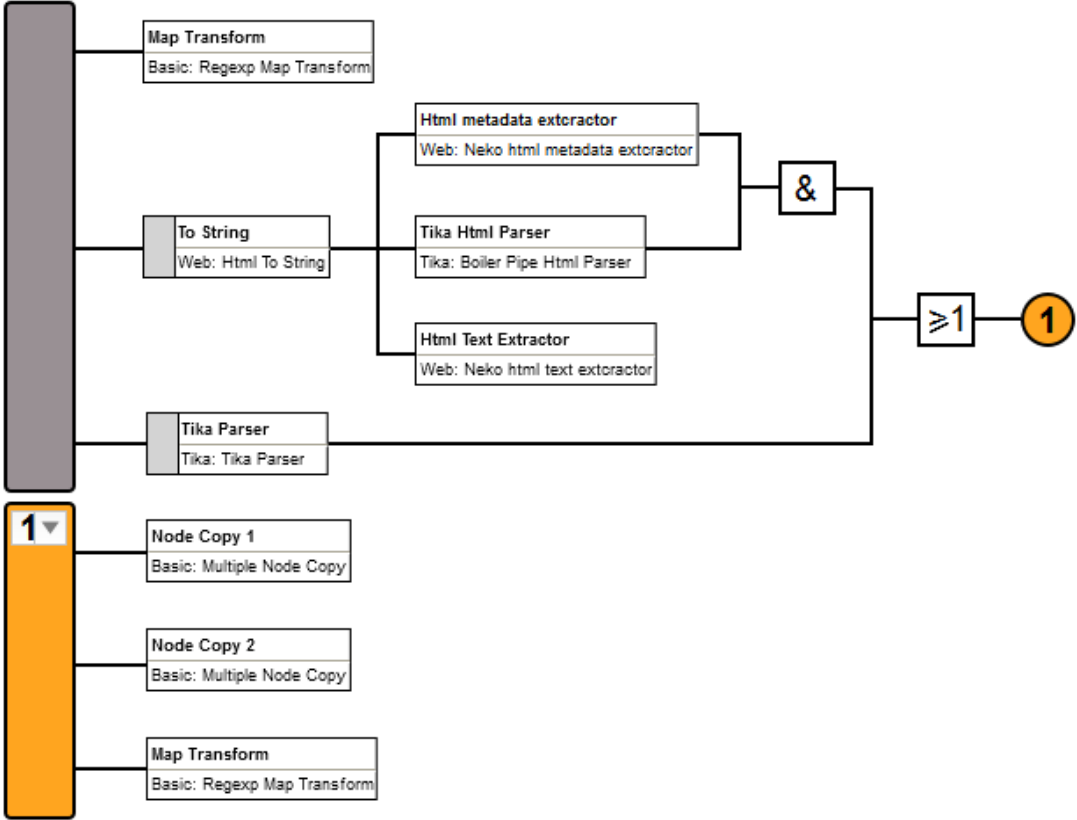


Figure 13: The final representation of the pipeline.

The users said that they often focused on a few stages at once in order to develop a specific function. In this representation it becomes easier to focus on one group since they are placed next to each other. The users were satisfied with this design and said that it was clearer than the previous ones.

5.3 Functionality for Testing

5.3.1 Test Driven Pipeline Development

When the users are about to create a pipeline they usually start by examining the data that should be indexed to see what metadata and information that is provided and should be searchable. The idea in Hydra is that the user then should configure a *Domain Model* which is a model of what properties the processed documents should have. The documents will then be matched against this model after they have passed through the pipeline to determine if the processing has been successful. In this way the user defines the desired result before he starts to extract and adapt the data. The domain model works like a filter that only lets the high quality documents through, so that the user can configure the pipeline until every document have passed.

How this would be implemented and used in the interface was not decided by the developers. Therefore the users were asked about their ideas and opinions of how the test driven development could be supported and used to make their work easier.

5.3.2 The Users Needs

The users explained that they missed functionality for helping them making the pipelines behave as intended in existing interfaces. They thought it was circumstantial to find documents that were processed incorrectly as well as the reason for the problems. Typically the user first tries to find errors then he locates documents that are affected by that error. He then examines the configuration of the stages in the pipeline to find the reason why the problem occurs so that it can be solved. He then re-indexes the document previously affected by the error and looks in its fields if the problem was solved correctly.

The users expressed that they wanted to have an overview of the documents in the pipeline to be able to see how many that had been processed and how many of them that had been accepted by the domain model in Hydra. This would help them to see what errors that are common so that they can focus on solving the problems that have the biggest effect on the output. They also wanted functionality for displaying the contents of a document's fields directly in the interface, without having to leave for another system.

5.3.3 Troubleshooting

During the third workshop the users had opportunity to develop ideas for how to provide an overview of the pipeline's performance and for functionality for troubleshooting. Some of the users had doubts regarding the idea with a domain model. They found it unlikely that they would know exactly how the output was supposed to look in the beginning of the development process of a pipeline, and therefore thought it would be hard for them to set up the model before configuring the pipeline. Everybody though agreed that the concept of test driven development was a good thing, if it could make their work easier. They were encouraged to think of the test driven development while developing their ideas.

In the beginning of the project the only way that Hydra could tell that there was an error was if a document had been in the pipeline for too long. The users agreed that this was too little information to help them with their troubleshooting. A notation for errors was discussed and it was decided that there were two kinds of errors; documents that failed to fulfill the conditions of the domain model and exceptions where a stage for some reason had crashed.

The users were very interested in being able to see what part of the domain model a document failed to fulfill. This would help them to find the reason for the failure by locating the stage that affected the specific field. They also wanted a possibility to follow a document through the pipeline in order to see what changes every stage did to the document.

During the workshop the users developed ideas and solution regarding these issues. Some of them showed different forms of tables displaying the contents of the fields of a document that changed when the user was stepping through the pipeline (see figure 14). The most important thing was to find errors and the reason for them in a quick and simple way.

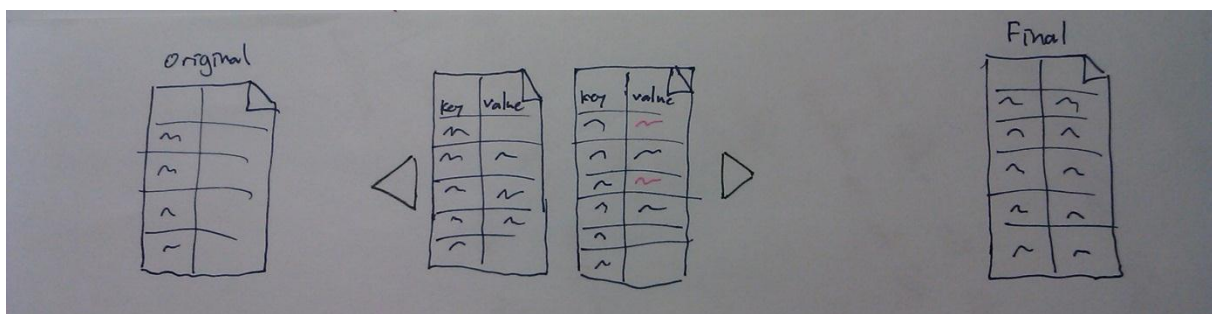


Figure 14: A sketch displaying different versions of a document and its fields.

It was also discussed whether there should be different modes for constructing and testing the pipeline. At last it was decided that there should only be one mode. The reason for this was that different modes would break the workflow. It was impossible to tell what the user's next action would be after inspecting a document. Either he has then found the solution for the problem and wants to enter the configuration of a stage to solve the problem, or he did not find the reason and wants to examine another document.

Hydra does not save every version of a document when it is processed by a pipeline, only the initial and the final version. To save every version of every document would have required a huge amount of data storage. The ideas of working with a set of documents instead of everyone started to emerge. To save every version of a few documents would not take up that much space. Then the user could add interesting or representative documents to the set and use them for locating and solving errors.

6 Result

6.1 Requirements

The following requirements are based on the analysis of existing pipeline administrator interfaces and the opinions of the users regarding sketches and prototypes.

6.1.1 Functional Requirements

The user must be able to set up a pipeline.

The interface must provide functionality to add stages to the pipeline, remove stages from the pipeline and to reorder the stages of the pipeline.

The user must be able to configure the contents of the pipeline.

The interface must provide functionality for giving names to the stages, configure the behavior of the stages and to set up conditions for when a stage can process a document, both regarding properties of the document and which stages that have processed the document before.

6.1.2 Usability Goals

The interface should be efficient to work with.

Time consuming activities that do not add to the performance of the pipeline, such as interaction that requires fine motor skills or extensive clicking should be avoided. There must for example be a smooth way of constructing the pipeline and setting up relations between stages.

Navigation must be easy.

The interface must be well structured so that it is easy to navigate between its different parts. Different modes and completely different views that break the workflow should be avoided. The organization of the functionality should support the workflow so that the user easily can reach the desired functionality, which can be different depending on the situation.

The visualization of the non-linear pipeline must be clear.

The user must be able to see in which order the stages will process a document. It is important that both the relations between stages and the direction of the pipeline are clear and consistent. Stage and document conditions should preferably be separated in order to avoid confusion. It is desirable that the user can affect the organization of the stages in the visual representation of the pipeline in some kind of clusters to make it easier for him to get an overview. The user should be encouraged to keep the relations between stages simple.

6.1.3 User Experience Goals

The interface should be transparent.

The way that the user sets up conditions should follow the model of the implementation. It is important that the interface does not fool the user by using another kind of logic. The relations should be built up by simple conditions; stages that has to process before and conditions regarding fields of the document, in combination with simple Boolean logic.

The interface should be helpful.

It should provide functionality to help the user in his work and provide the right kind of information on the right time. Search functions and other tools that contribute to a better workflow are desirable. The interface should provide visual hints of the outcome of drag and drop actions so the users can avoid errors. The users would also like to have modelless feedback of how well the pipeline performs.

The interface should be discrete.

Even though the users want a helpful interface, the help should be provided in a discrete way. The interface should “trust” that the user knows what he is doing and only break the workflow by notifications when a very critical action, that will cause work to be deleted, is to be executed.

The interface should support test driven pipeline development.

The interface should provide an overview of the performance of the pipeline so that the user can see what kind of errors that appears frequently. It should also be possible to see which stages and documents that causes or holds the error, which makes it easier for the user to figure out the reason for it. A function for tracking documents through the pipeline to see what changes every stage has made to it is desirable. For documents that do not match the domain model, it should be possible to see on which part of the model that they fail. The users would also like to have a possibility to run custom made test documents through the pipeline.

6.1.4 Environmental Requirements

The interface should work on both large and small screens.

The interface should be optimized for large ($\approx 24''$) desktop screens, but it should still be possible to work with on smaller laptop screens ($\approx 13-15''$). Therefore the visualization of the pipeline must be relatively simple, clean and clear so it can be displayed on a small screen.

6.1.5 Data Requirements from Hydra

Model errors and exceptions must be separated.

There must be a way to separate errors that occur because a document does not match the domain error and errors that are caused by failing stages. It should also be possible to get more detailed information about errors from the two categories.

It should be possible to determine if a stage has made changes to a document.

It should be possible to see if a stage has written any changes to a document during the processing of it. There should also be a possibility to save every state of a document, after every stage, so that the user can trace it through the pipeline and see exactly what changes every stage has made to it.

6.2 Design Solution

In this section the final design of the interface is presented in the form of wireframes, describing its elements, functionality and how to interact with it.

6.2.1 Structure

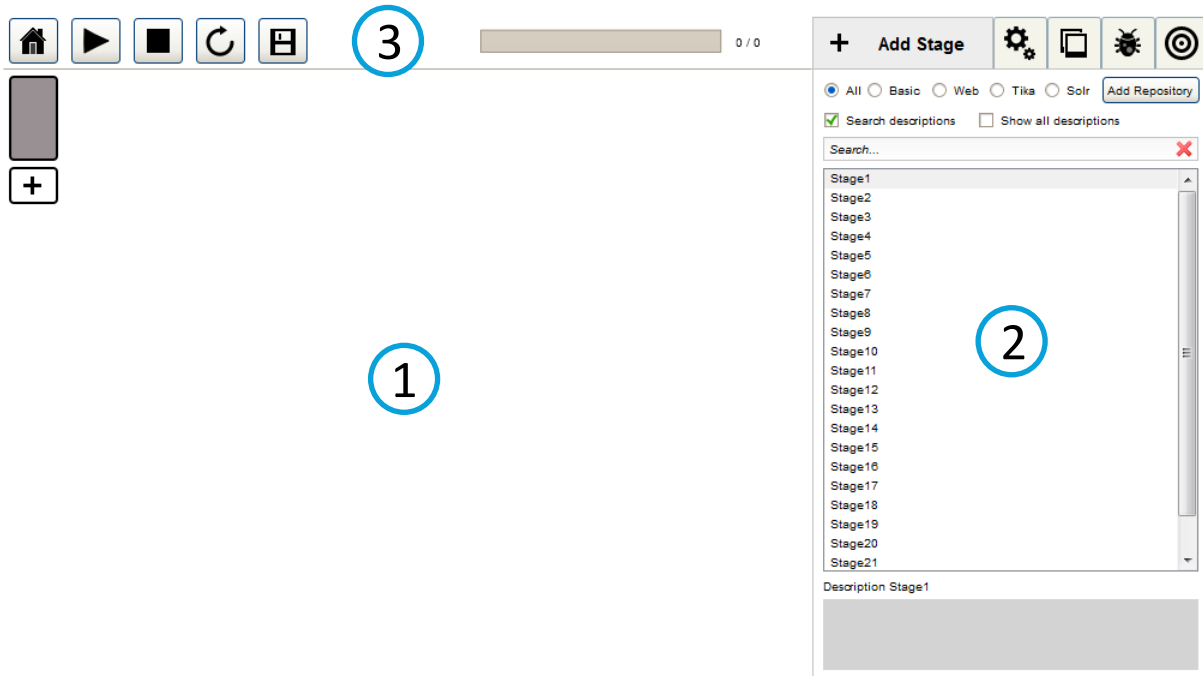


Figure 15: Structure of the interface.

Workspace

The space to the left is the workspace where the stages in the pipeline and their relations to each other are displayed (see figure 15:1). In this figure no stages have been included to the pipeline yet. The grey bar in the upper left corner represents the input node.

Toolbox Tabs

All functionality can be reached from the tabs to the right (see figure 15:2). The size of the tab panel stays the same regardless of what functionality that is chosen. The open tab, here *Add Stage*, displays the name and a representative icon. Closed tabs only display an icon, due to the limited amount of space. The functionality that can be reached from the tabs is from left to right *Add stage*, *Configuration*, *Documents*, *Debug Set* and *Domain Model*.

Controls

Above the workspace there is place for buttons for controlling the pipeline (see figure 15:3). A suggestion is that there should be functionality to (from left to right) *leave the pipeline view*, *start the document processing*, *stop the processing*, *re-index all documents* and *save the pipeline configuration in its current state*. The progress bar to the right of the buttons shows how many of the incoming documents that have been processed by the pipeline and passed the domain model.

6.2.2 Pipeline Visualization

Stages

Stages are represented by boxes displaying the name of the stage, here “Map Transform”, on the upper line and the name of its type after which collection it belongs to, here “Basic: Regexp Map Transform”, on the lower line (see figure 16). To save space and avoid scrolling sideways, the width of the box is adapted to the length of the text.



Figure 16: Visual representation of a pipeline stage.

Document Conditions

Stages that have a condition regarding which documents they should process have a marking on their left side (see figure 17). This just reminds the user that a stage has a document condition, not of what type it is or if there is more than one.

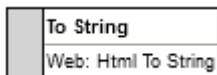


Figure 17: Stage with document condition(s).

Stage Conditions

Stages connected directly to the input node do not have any stage conditions (a condition that says that another stage must have processed a document before that document can be processed by the current stage) (see figure 18).

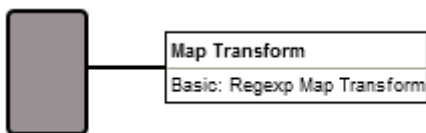


Figure 18: Stage with no document or stage conditions.

Stages that have a connection to their left side can only process documents that have passed the stage connected to the left end of that connection (see figure 19).

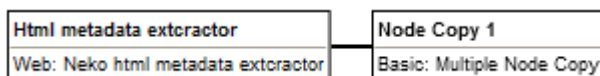


Figure 19: Stage with a condition that the stage to the left must process first.

Stages that are attached to a connection from a box with an “&” (and) sign must wait for all stages that are connected to the box from the left (see figure 20). The same thing goes for boxes with an “≥1” (or) sign, but in this case the stage only have to wait for one of the stages connected to the left side of the box (see figure 21).

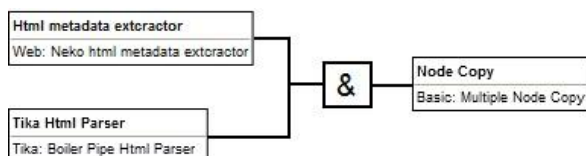


Figure 20: Stage connected to an & condition.

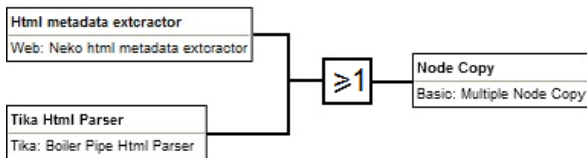


Figure 21: Stage connected to an ≥ 1 condition.

Connection Nodes

Stages that are connected to another bar than the one representing the input must wait for stages connected to the numbered circle, with the same number as the bar, before they can start processing a document (see figure 22).

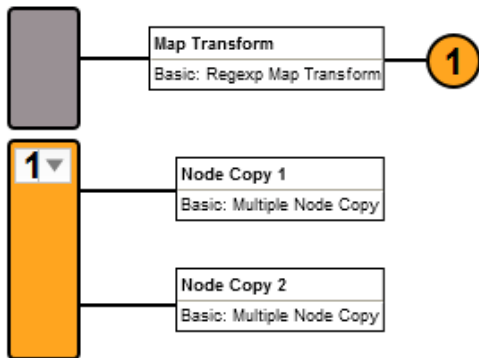


Figure 22: A node connected to a stage and two stages connected to a bar.

6.2.3 Tools

Add Stage

Under the tab *Add Stage* the user can choose a new stage to include in the pipeline (see figure 23). All stages that are available are displayed in a scrollable list. Here the checkbox “show all descriptions” is unchecked. The list displays the name of the stages available and if the user clicks a stage, the description of that stage is displayed on the area under the list. If the checkbox is checked both the names and descriptions will be shown in the list. The list will then be longer since the description area under it will disappear.

The user can click the list and start writing the first letters in the name of the desired stage, which will cause the list to scroll to the position of the stages beginning with that letters.

At the top there are radiobuttons that works like a facet. The different choices represents the collections of stages that are available. If a collection is chosen the list will only display the stages of that collection. The user can add more collections by pressing “Add Repository”.

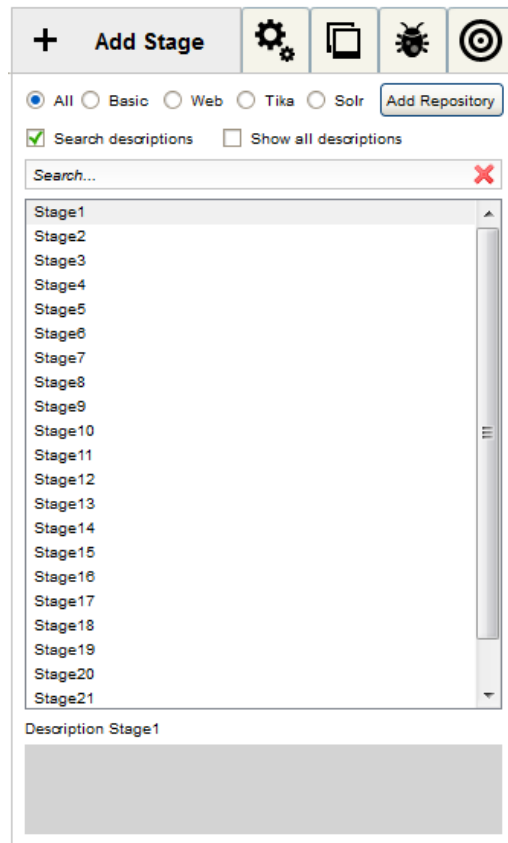


Figure 23: The *Add Stage* tab panel.

The user can also search for a stage by using the free search text box. If the checkbox “Search descriptions” is selected the query will be matched against both the names and descriptions of the stages. If it is not, the search will only match against the names. The query can be deleted by pressing the red cross.

All filtering will cause the list to be updated in real time. If the user uses both the facets and the free text search, only stages matching both will be displayed.

To include a stage in the pipeline, the user selects the desired stage and drags it out to the workspace area.

Configuration

When the user clicks on the visual representation of a stage, the configuration of that stage will be opened under the configuration tab (see figure 24). If the user clicks the configuration tab when another tab panel is open, the configuration that was last opened will be shown.

At the top of the tab panel, the user can give the stage a name by entering it in the textbox. Under that is the name of the type of the stage, which cannot be edited.

Under that there are three expandable text boxes: *Custom description*, *Stage description* and *Dependencies*.

Custom description holds a textbox where the user can enter a specific description of the behavior of the stage instance.

The *Stage description* is the built in description of a stage of the specific type, which will be shown when the user expands it. This description cannot be edited by the user. If it is too long it will be scrollable, so it does not cover the whole tab panel.

Under *Dependencies* the user can add conditions regarding what documents the stage should process. These will preferably be written on the same form as the conditions in the *domain model*.

The lowest section contains the configuration of the behavior of the stage. How this will look depends on the implementation of stages in Hydra. Probably the size of it will differ a lot between different types of stages. The behavioral configuration has a separate scroll bar to make it possible for the user to look at part of it at the same time as the description or dependencies.

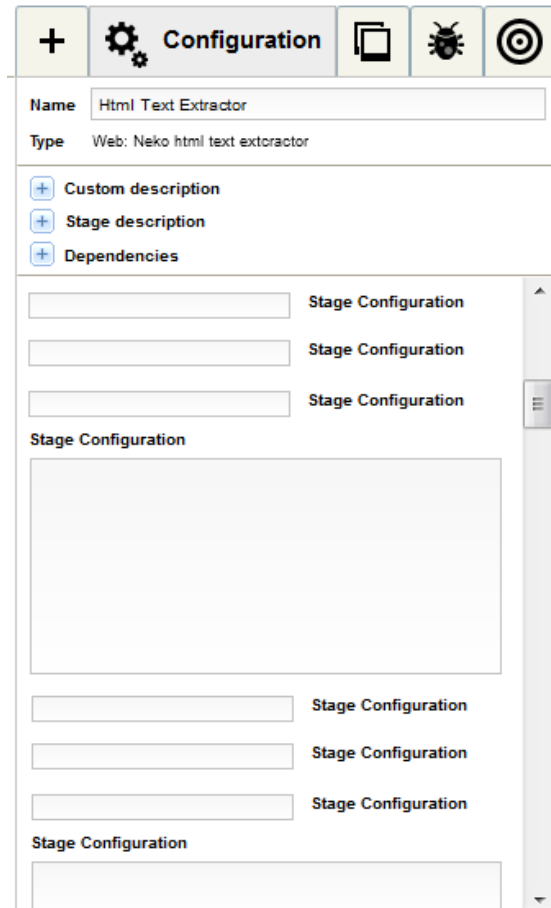


Figure 24: The Configuration tab panel.

Documents

Under the *Documents* tab the user can get information about the performance of the pipeline (see figure 25).

A set of the documents that have entered the pipeline will be shown in the list. The intention is not that the user should look for documents in this list but instead using the filters to find documents with certain properties.

There are different filters that the user can choose from, some of them can be combined which will narrow down the search results. The filters are divided into collapsible sections containing different choices. The numbers next to the labels tells the user how many documents there are in the category.

Filters under *Model Errors* is used for filtering out documents that do not match the domain model. The radio buttons represents different types of errors, which should be specified in the domain model.

Filters under *Exceptions* are used for filtering out documents that have been processed by a stage which then has caused an exception. Further filtering in this category will be names of the stages that caused the exceptions or more detailed error information, if it can be provided by the stage classes.

Under *Field Values* the user can filter out documents depending on field properties. The combobox in middle should contain different choices such as “has value”, “is empty” or “exists”. These choices should preferably follow the syntax user for constructing the domain model in order to make the interface consistent. In the left textbox the user enters the name of the field and in the right one the desired value, if applicable.

If a filtering category is minimized and a choice other than “All” have been made, that choice will be displayed beside the label so that it is remembered. The filtering can be removed by clicking the red cross and “All” will be selected instead.

By selecting the checkbox “show indexed only” only documents that have passed the domain model will be displayed. This will cause any filtering regarding errors to be inactivated, since these to categories are each other’s opposites.

There is also a free text search where the user can enter a name or id of a document.

In the bottom of the panel there are three buttons where the user can choose what to do with a selected document in the list. The first two “Show original” and “Show final” will display the document with its original and final values in the fields in a pop up window that the user can move around.

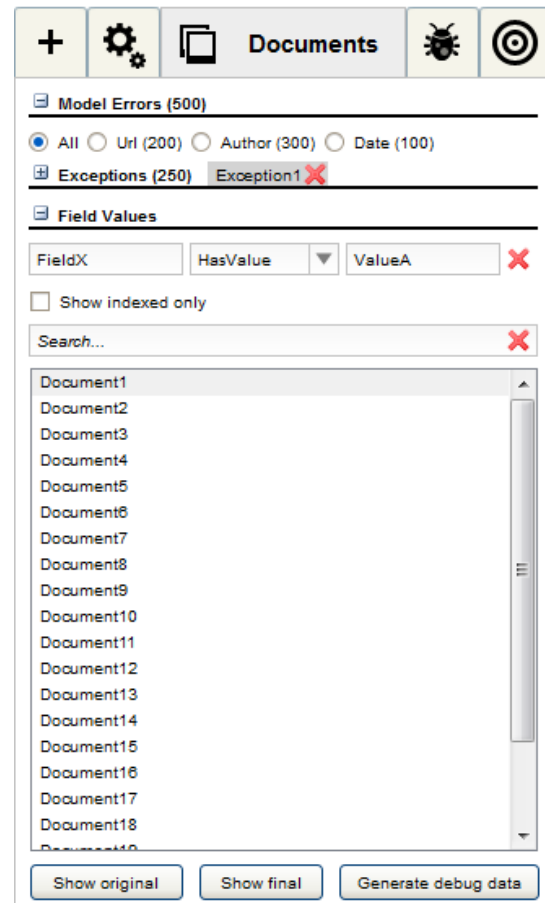


Figure 25: The *Documents* tab panel.

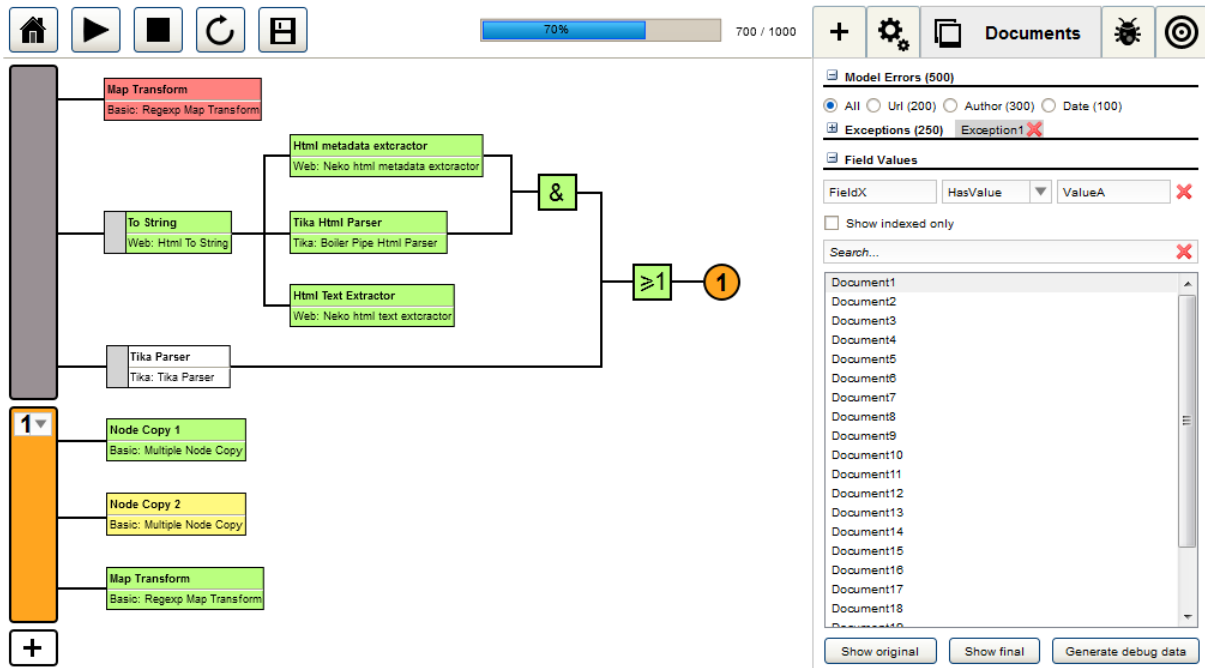


Figure 26: Information about a selected document shown by coloring the pipeline stages.

If a document in the list is selected information of what stages that document has been processed by will be displayed in the visual representation of the pipeline. Stages that have processed the documents and written to it will be in one color, stages that has just processed a document and that has caused exceptions will be other colors (see figure 26).

The button “Generate debug data” will place the selected document in a debug set. The document will then be processed by the pipeline again and this time it will be recorded what changes every stage does to the fields of that document.

Debug Set

Under the tab panel “Debug Set” the documents that user has added to the set are shown in a list (see figure 27). To the left of the name of the document there is a green checkmark or a red cross depending on if that document has passed the pipeline or not.

In the bottom of the panel there are buttons with different choices that the user can choose for selected documents. The three right ones acts like it sounds “Remove selected” removes the selected documents from the list, “Re-index selected” reruns the selected documents through the pipeline and “Re-index all” reruns all documents in the list.

If the button “Inspect” is pressed the user gets an opportunity to see what changes every stage has done to the selected document. The document will be opened in a pop up window displaying the original version. The user can then step through the pipeline (in the order that the document was processed) and all changes will be highlighted.

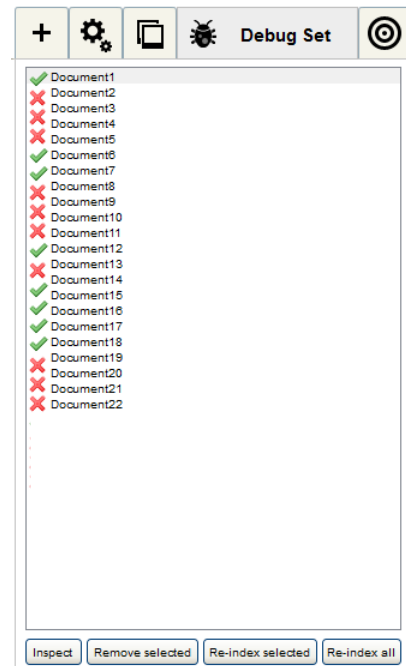


Figure 27: The *Debug Set* tab panel.

Domain Model

Under the tab panel “Domain Model” the user imports or builds up the logic for the domain model of the pipeline, that is, the conditions that the fields of the documents have to pass in order to be indexed (see figure 28).

A condition is added by pressing the “+”-sign in the upper left corner. It can then be given a name and the field conditions are built up by selecting fields and logical operations. The “+” sign next to the name adds another row for field logic within the condition.

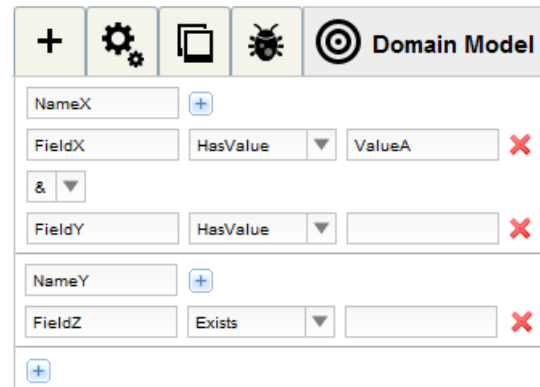


Figure 28: The *Domain Model* tab panel.

6.2.4 Interaction

Drag and Drop

Stages are placed on, and relocated within, the workspace by drag and drop actions. If the user clicks a stage and holds the button the stage will change its representation to a smaller box displaying its name so that it only covers a smaller part of the screen. If the user drags the stage close to an area where it can be placed, visual feedback will be provided to display the outcome if the stage is released.

If a stage is held on an empty area to the right of the input node or another stage the interface will highlight the area where it will be placed if dropped and what connections that will be created (see figure 29). If the stage is held to the right of a stage that already is connected to another stage on its right side, the new stage will be placed in between the stages and the previous connection will be moved to the right side of the new stage.



Figure 29: Visual feedback for drag and drop actions.

If the user holds the stage above or below a stage the interface also provides feedback (see figure 30). If the stage is dropped the connection to the old stage is divided and the new stage will share the connection.

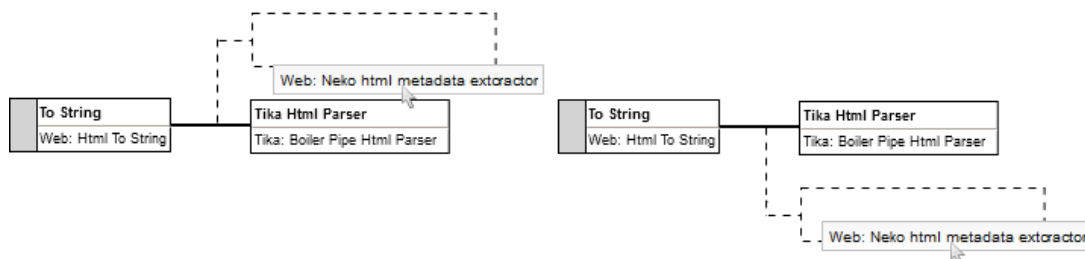


Figure 30: Visual feedback for drag and drop actions.

Context Menus

When the user clicks the visual representation of a stage the stage will be selected and colored. If he holds a key (preferably Ctrl or Shift) he can select more than one stage. If the user then right clicks a context menu will open. There are four options to choose from; “Connect with OR”, “Connect with AND”, “Create node” and “Remove dependencies”. The first two are used to create Boolean conditions. If the user chooses one of these when two or more stages are selected, they will be connected with a gate or the selected type (see figure 31).

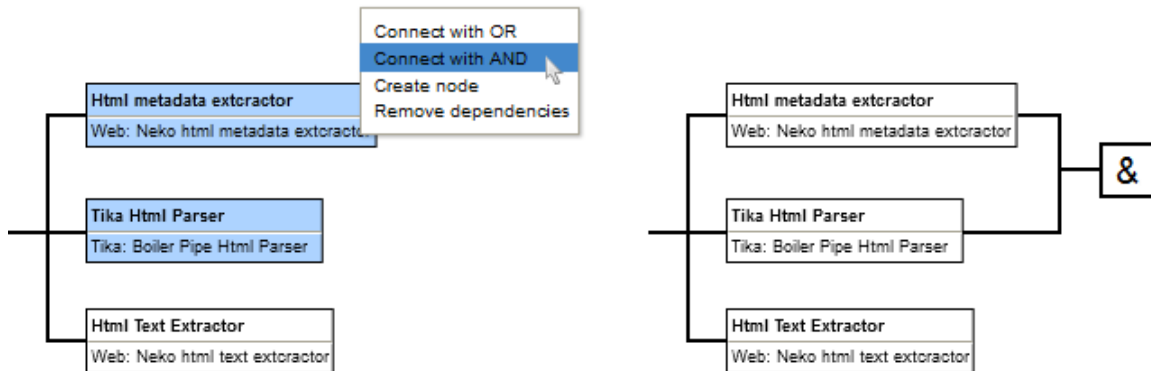


Figure 31: How to create an “&” condition.

The third option is for creating connection nodes so that the pipeline can be continued on the next horizontal row. This option will work when one or more stages are selected. When the user selects this option a node with a number will be connected to the right end of the stage(s) (see figure 32). Another bar will also show up under the input node displaying the same number as the node in a combo box. If more than node has been created, all their numbers will be selectable in the box. The user can also choose to create a new bar before the node by pressing the “+” sign to the left.

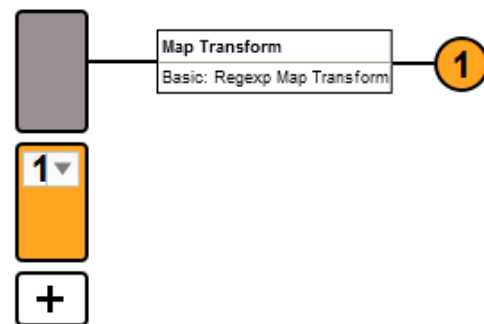


Figure 32: Connection node and bar.

The last option “Remove dependencies” simply removes all connections to the selected stage(s). If a stage in the pipeline is dragged to another position the old connections will also disappear. The old position will still be marked out as long as the user has not dropped the stage, so it is possible for the user to drop it back to its old position.

7 Discussion

7.1 Participatory Design with Expert Users

7.1.1 Experts are Providing Knowledge

In a design project like this, defining the problem is part of the process since the specification of demands develops together with the solution (Löwgren & Stolteman, 1998). Pipeline development is something rather complex for a person without experience in the field. The only persons that know exactly how the process of developing pipelines works at Findwise are in fact the developers at Findwise. If the users had been left out during the development process, it would have been very hard both to define and solve the problems that were encountered.

From the information collection during the initial discussions and interviews it was possible to say that the users had problems with existing interfaces, but it was first during the development of new ideas that it was possible to fully understand what the reasons for users' problems were. During the workshops the users presented ideas of solutions that someone without their experience probably would not have thought of, which indicates that participatory design activities has made a contribution to the solution.

As discussed in the article (Heinbokel et al, 1996) there are disadvantages of involving the users in development processes in terms of innovation, flexibility and effectiveness. The authors suggest that the problems arose because of problems in the communication between the users and developers, and because the users tended to come up with new ideas late in the process. In this case the users have the same technical knowledge as the developers, which mean that they probably managed to understand the system faster than they would have been able to without it. This means that they were able to express their ideas early in the process which has limited this kind of problems.

7.1.2 Expertise versus Innovation

Since Hydra did not have any graphical interface before this work was done, it all had to be designed from scratch. By the start of this project the task was very open; the only thing that was specified was that the administrator interface should be adapted for people working with pipelines at Findwise. This means that there was no way of observing people working with the system. To find inspiration the interfaces of similar tools were analyzed and evaluated together with the users. This somewhat limited the framework since it focused on solving problems that occurred in other systems, but it also gave an insight in how the interface solutions affected the work situation for the users.

During the workshop the users found it hard to overlook technical constraints and tended to focus on how if it was feasible to implement. Here the expertise of the users was standing in the way for innovative solutions. It is possible that the users could have been more encouraged to think outside the box by using a different format for the workshops. Maybe it would have been a good idea to hold it outside of their workplace or to bring some kind of artifacts that were not related to their work in order to stimulate their creativity. On the other hand, the ideas that were developed are more likely to be realizable, which is a good result from the company's point of view.

The user's knowledge about Hydra's functionality varied a lot. Some of them had been involved in the development themselves while other had just heard about it. Some of the involved persons have probably gained knowledge about Hydra by participating in this project. Sometimes the less experienced users expressed more innovative ideas than the more involved ones, which also indicated that too much knowledge can be an obstacle for creativity when it comes to highly educated persons.

7.2 The Interface

7.2.1 Reception

Both the user and stakeholders of Hydra have expressed their satisfaction over the developed design, which indicates that a good solution has been produced using a participatory approach.

A reason for the good reception could be that the users feel a sense of ownership of the design as described in literature (Sharp, Rogers & Preece, 2007). This could mean they are less likely to give critique since the design is based on their own ideas. They might also be more receptive of the design since they have been involved in the process and know why certain design decisions have been made. If the developed solution would have been presented to a group of people that have not been involved in the development process it is possible that would be more skeptical, since they do not know the reason for the design decisions. It is also possible they will find it harder to use the interface.

7.2.2 Adaption for Experts

Since the users are satisfied with the result and the design is based on their needs and ideas, the design is adapted for this group of experts that are also experts in computer oriented areas. The model for visualization of the pipeline and the troubleshooting functionality are very transparent and close to the implementation model, which follows the guidelines for design for expert users (Staggers & Norcio, 1993). In the beginning of the design development the ideas were further from the implementation model, but it turned out that another layer of made up functionality was more confusing than helpful to the users. It could be the case that it is extra important to have a transparent solution when designing an administrator interface, since the user must have control over what is actually happening.

A surprise was that the user accepted the drag and drop solution, since design for expert otherwise promotes keyboard shortcuts and less graphical approaches (Cooper et al 2007). They though had their concerns about the drag and drop interactions, it had to be quick and smooth to work with. The solution where the dependencies are set up automatically when the user drops a stage at a specific location probably helps to speed up the work here. The amount of time the user spends on dragging objects he will save since he do not have to spend time on setting up the dependencies.

7.2.3 New Ideas

In some aspects Hydra is different from other administrator tools which meant that new functionality had to be included that did not exist in other interfaces. The developer explained that it was possible to add functionality to the system if the users expressed needs that were not possible to satisfy with the current functionality.

In some aspects the developed interface is very different from other systems. The biggest difference is the drag and drop interactions and the graphical visualization of the pipeline, but hopefully it also encourages the user to work with a test driven approach. The functionality for troubleshooting requires more data than Hydra provides today, this can be fixed if the developer finds the functionality useful. The fact that there have been a communication between the users and developers with exchange of information and ideas are another good outcome of the participatory approach.

7.3 Future Work

The next step in the design process would be to build a functional high fidelity prototype of the interface. It should test the drag and drop interaction to see if it is possible to set up the conditions by placing stages in different zones or if it requires too much fine motor skills. The error handling and troubleshooting functionality must also be tested with real data to see if the concept works in a real situation. A coherent syntax for describing document conditions and the domain model also needs to be developed.

Preferably the participatory approach should be kept to assure that the users still feel involved in the development. They are also the ones at the company that have the best knowledge of the ideas that the design is based on. The interface should also be tested by users that have not been involved in the process.

8 Conclusion

The goal of this work was to develop a design for an administrator interface for a pipeline development tool named Hydra. The interface should be adapted for expert users in form of pipeline developers at Findwise, who are also highly educated in computer science and engineering. The work was executed using user centered and participatory design methods to examine if this approach was suitable for expert users.

There were both advantages and drawbacks of involving expert users. The users were engaged to participate in the activities and were very helpful. Their technical knowledge made it possible for them to express feasible ideas, but it also restricted the level of innovation in the design because of their focus on implementation matters. During the process they provided valuable information about their thoughts and needs regarding their tasks that would not be available for a designer through any other channel.

The outcome of the work was an interface design in the shape of wireframes. The users and stakeholder were satisfied with the result, which indicates that the method used was suitable. A reason for the good reception could be that the users feel a sense of ownership of the developed interface.

The design of the interface is transparent and very close to the implementation model of Hydra. This follows the guidelines for interface design for expert users, and is probably necessary for the users to be able to administrate the system which means that they must have control over the outcome of their actions.

To establish the positive reception of the design and see if it works in real work situation, a high fidelity prototype must be developed so that the interface can be tested with real data, and by users that did not participate in the development.

9 References

- Aiim.org. (2012). *What is Enterprise Search?* URL: <http://www.aiim.org/What-is-Enterprise-Search>. [Last accessed 2012-01-29].
- Cooper, A., Reimann, R. & Cronin, D. (2007). *About Face 3: The Essentials of Interaction Design*. Indianapolis, Indiana: Wiley Publishing, Inc.
- Dieselpoint Inc. (2010). *Say hello to OpenPipeline*. URL: <http://www.openpipeline.org>. [Last accessed: 2012-02-10].
- Evolus (2010). Pencil Project. Url: <http://pencil.evolus.vn/en-US/Home.aspx>. [Last accessed: 2012-02-10]
- Faulkner, X. (2000). *Usability Engineering*. Basingstoke, Hampshire & New York, New York: Palgrave.
- Findwise.com (2012). *Findability by Findwise*. URL:<http://www.findwise.com/what-we-do/findability-by-findwise>. [Last accessed: 2012-02-10].
- Foster Jr, S. T. & Franz, C. R. (1999). User involvement during information systems development: a comparison of analyst and user perceptions of system acceptance. *Journal of Engineering and Technology Management*. Vol. 16, no. 3-4: 329-348.
- Gulliksen, J. & Göransson, B. (2002). *Användarcentrerad systemdesign*. Lund: Studentlitteratur.
- Heinbokel, T., Sonnentag, S., Frese, M., Stolte, W. & C. Brodbeck, F. (1996). Don't underestimate the problems of user centredness in software development projects - there are many! *Behaviour & Information Technology*. Vol. 15, no. 4: 226-236. DOI: 10.1080/014492996120157
- Interaction-Design.org (2009). *Mental models*. URL: http://www.interaction-design.org/encyclopedia/mental_models_glossary.html. [Last accessed 2012-01-31].
- Kautz, K. (2011) Investigating the design process: participatory design in agile software development. *Information Technology & People*. Vol. 24, no. 3: 217-235. DOI: 10.1108/095938411111158356
- Löwgren, J. & Stolteman, E. (1998). *Design av informationsteknik - materialet utan egenskaper*. Lund: Studentlitteratur.
- Microsoft (2010). *Microsoft Lync*. URL: <http://lync.microsoft.com/en-gb/product/pages/capability-overview.aspx>. [Last accessed: 2012-02-14].
- Microsoft (2011). *FAST Search Server 2010 for SharePoint*. URL: <http://sharepoint.microsoft.com/en-us/product/capabilities/search/Pages/FAST-Search.aspx>. [Last accessed: 2012-02-12].
- Muller, M. J. & Druin, A. (2010). *Participatory Design: The Third Space in HCI*. Cambridge: IBM Watson Research. URL: [http://domino.research.ibm.com/cambridge/research.nsf/0/43d801f234786fe58525777d00723efb/\\$FILE/TR2010.10%20Participatory%20Design%20The%20Third%20Space%20in%20HCI.pdf](http://domino.research.ibm.com/cambridge/research.nsf/0/43d801f234786fe58525777d00723efb/$FILE/TR2010.10%20Participatory%20Design%20The%20Third%20Space%20in%20HCI.pdf). [Last accessed: 2012-02-08].

Schuler, D. & Namioka, A. (1993). *Participatory design: Principles and practices*. Hillsdale, New Jersey: Lawrence Earlbaum Associates, Inc.

Scott, B. & Neil, T. (2009). *Designing Web Interfaces*. Sebastopol, California: O'Reilly Media Inc.

Sharp, H., Rogers, Y. & Preece, J. (2007). *Interaction Design: Beyond human computer interaction, 2nd Edition*. Chichester, West Sussex: John Wiley & Sons, Ltd.

Staggers, N. & Norcio, A. F. (1993). Mental models: concepts for human-computer interaction research. *International Journal of Man-Machine Studies*. Vol. 38, no. 4: 587-605. DOI: <http://dx.doi.org.proxy.lib.chalmers.se/10.1006/imms.1993.1028>