OPAALS PROJECT

Contract n° IST-034824

WP10: Sustainable Community Building

Del10.18 - Wille visualisation toolkit for developers with a concise OKS Visualisation Application catalogue for end-users

Project funded by the European Community under the "Information Society Technology" Programme

Contract Number: IST-034824

Project Acronym: OPAALS

Deliverable N°: 10.18

Due date: May 2010

Delivery Date: May 2010

Short Description:
In this deliverable we present the final version of the visualisation system that concludes the line of research from the early visualisation system prototype to component-based design with applications.

Authors: Ossi Nykänen (Senior Researcher, Team Leader), Jaakko Salonen (Researcher), Jukka Huhtamäki (Researcher)

Partners contributed: IPTI, SUAS, UniKassel

Made available to: OPAALS Consortium

Versioning

Version	Date	Name, organization
v1	03/10	TUT internal draft
v2	04/10	Comment draft
v3	05/10	Final version

Quality check

Internal Reviewers:

Lia Carrari (IPTI), Frauke Zeller (UniKassel)

Dependences:

Achievements*	Final version of the visualisation architecture, developer toolkit with end-user visualisation examples, and sustainable website for the visualisation system.
Work Packages	• WP5 • WP6 • WP9 • WP10
Partners	IITK, IPTI, SUAS, LSE, UniKassel
Domains	• Computer science • Social science
Targets	• Visualisation developers, designers and end-users • OKS developers
Publications*	Nykänen, O. (2009). Semantic Web for Evolutionary Peer-to-Peer Knowledge Space. In Birkenbihl, K., Quesada-Ruiz, E., & Priesca-Balbin, P. (Eds.) Monograph: Universal, Ubiquitous and Intelligent Web, UPGRADE, The European Journal for the Informatics Professional, Vol. X, Issue No. 1, February 2009, ISSN 1684-5285, CEPIS & Novática. Available at http://www.upgrade-cepis.org/issues/2009/1/upgrade-vol-X-1.html Nykänen, O. (2009). Understanding Data via an RRS in RDF/XML. In IADIS International Conference Applied Computing 2009, Rome (Italy), Nov. 19-21, 2009, Vol. I, ISBN 978-972-8924-97-3. Salonen, J., Huhtamäki, J. (2010). Launching Context-Aware Visualisations. Proceedings of the 3rd International OPAALS Conference on Digital Ecosystems: OPAALS 2010. 22.-23. March 2010, Aracaju, Brazil. To appear.
PhD Students*	Jukka Huhtamäki: Researcher Jaakko Salonen: Researcher
Outstanding features*	• Deployed visualisation system that natively supports P2P configuration • Sustainable developer resources
Disciplinary domains of authors*	Computer Science: Jukka Huhtamäki, Ossi Nykänen, Jaakko Salonen

The information marked with an asterisk (*) is provided in order to address Recommendation n. 4 from the Year 2 review report

Table of Contents

1. Introduction

In this deliverable we present the final version of the visualisation system that concludes the line of research from the early visualisation system prototype in Deliverable 10.6 (Haapaniemi, Huhtamäki, Kortemaa, Mannio, Nykänen, Salonen, 2007) to component-based design in Deliverable 10.12 (Huhtamäki, Nykänen, Salonen, 2009a). In brief, the main contribution of this final deliverable is threefold:

1. It finalises the technical main visualisation tasks of the reading, transforming, and viewing data by deploying the profile-enabled Wille 2 visualisation framework. The framework has been finalised taking the use case requirements, prototype, and application feedback into account. In particular, the final version abstracts many of the tedious tasks of data processing, enabling the implementation of quite complex visualisation pipelines in terms of very short scripts.
2. It introduces several visualisation components and end-user applications in the OPAALS context, providing a concise visualisation catalogue sufficient in demonstrating the capabilities and roles of the visualisation system. In brief, this catalogue provides insight for visualisation developers, for modifying the examples to suit the needs of particular applications, and for developing new applications within the framework.
3. It deploys a sustainable Web site for the visualisation system. In brief, this website provides public access to the visualisation system, documents and tools. This sets up a framework for developing and deploying the visualisation system further and utilising it in the future research and application development.

Note that since the application development has taken place in parallel with other OPAALS tasks, the main applications have been developed in the collaborative context of other work packages, namely WP5 and WP9 (Guigoh and Flypeer integrations and demonstration use case), WP10 (social networks), and WP6 (evolutionary framework for language). This also supports sustainability by introducing the visualisation concepts, system, and catalogue applications in new research activities.

As a consequence, the results of the deliverable are distributed between three entities:

1. This narrative deliverable document currently at hand outlines the work topics and research behind the visualisation system and the demonstrative applications.
2. The Wille visualisation system wiki website (http://wiki.tut.fi/Wille/) provides main access to the documentation, software releases, and tutorial examples of the visualisation system.
3. The runnable application examples provided alongside this narrative deliverable document demonstrate the key applications not downloadable from the web site. (Please note that some of these applications may include sensitive data, specific to the OPAALS research community.)

The rest of this document is organised as follows: In chapter 2 background of this work is considered, in chapter 3 we present and outline the toolkit that was developed as part of the work, in chapter 4, the catalogue of end-user visualisations based on the toolkit are presented, finally we conclude the presentation in chapter 5 with discussion and final conclusions.

2. Background

To understand the contribution of this deliverable, it is instructional to consider its immediate research background and the state-of-the-art context of (Web-based) visualisation systems.

2.1. Development Rationale

The main architecture of the visualisation system and suggested process models were established in Deliverable 10.12. In brief, the visualisation system comes with three architectural configurations: Processor, Hosted Component, and Adapted Component (Figure 2.1.1).

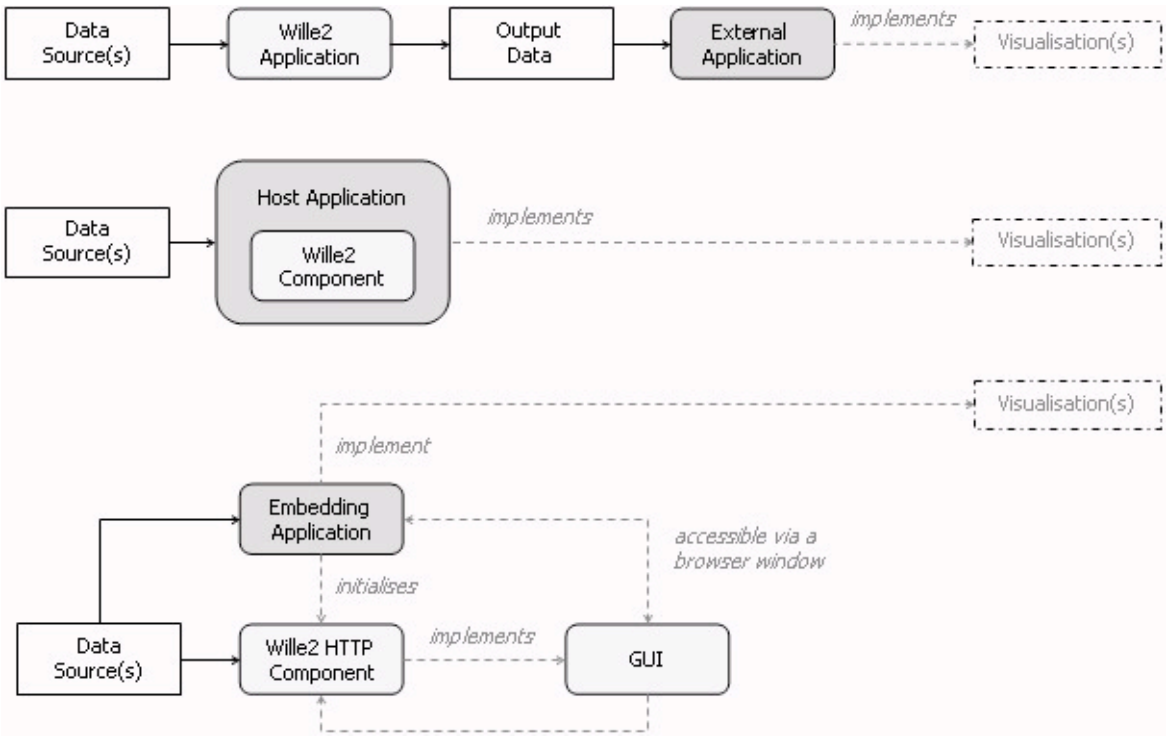


Figure 2.1.1. *Three architectural configurations of Wille2 (Huhtamäki, Nykänen, Salonen 2009a)*

Further, because developing non-trivial visualisations requires scripting and adapting new visualisation components, the visualisation application development and usage process model is fundamentally iterative (Figure 2.1.2). Also, the process fundamentally includes the common scenario of finding and/or developing new visualisation applications and components in cases where the provided visualisation applications are not sufficient. For details, please see Deliverable D10.12 (Huhtamäki, Nykänen, Salonen 2009a).

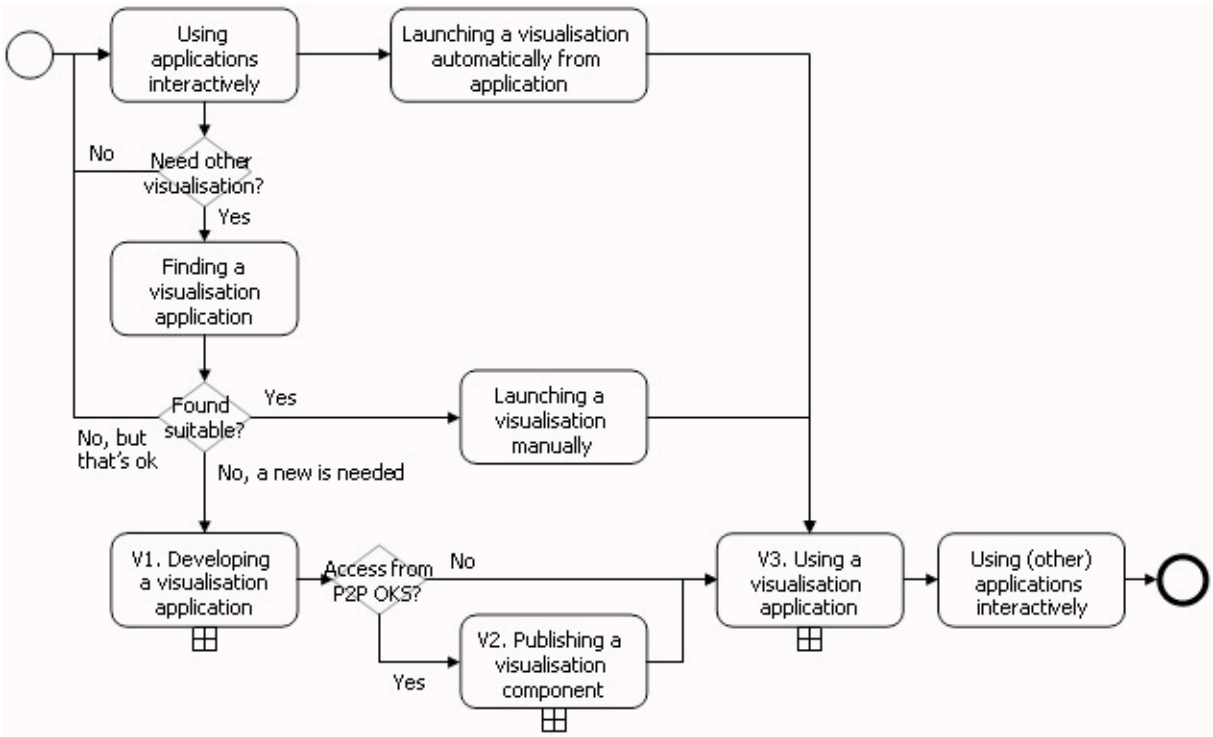


Figure 2.1.2. *Visualisation application development and usage process model (Huhtamäki, Nykänen, Salonen 2009a)*

Alongside with the visualisation system role definitions (Author, Developer, Designer, End-User), this explains the main research and development choices, leading from the early preliminary studies to the final system. Since each practical visualisation application is different either by data source, data pre-processing and conceptualisation, transformation and visual mapping, and viewer functionality, the resulting complexity can only be managed by iterative development where visualisation designers introduce and integrate visualisation components to the benefit of particular end-users.

As a consequence, end-users have to know relatively little about the development process but on the other hand, depending on the application, may have only limited options for configuring the applications. The abstract nature of the visualisation system as a framework also implies that while the end-user applications share certain technical and conceptual characteristics (access to private and 3rd party author data, pipeline architecture, itemisation of free-form data, etc.), the Wille visualisation system can typically be encapsulated behind the scenes.

2.2. Towards a Component-based Visualisation System

In Deliverable 10.6 we reviewed the applicable open source components. This provided a foundation of abstracting the services and applications of complex visualisation pipelines. In turn, it also highlighted the sporadic nature of existing stand-alone applications, most of which have been designed for a highly specialised purpose. On the other hand the publicly available, "webised" visualisation systems provide a high-level integration and look and feel, but with the price of monolithic design. In the worst case scenario this means systems whose components are not exposed to, nor can be reasonably configured by end-users. Please note that due to nature of this project, commercial applications were not considered.

Due to the significance of this application domain, several different approaches to information visualisation exist. Mainstream Web applications such as Yahoo Pipelines[1] and IBM Many Eyes [2] demonstrate the concept of collaborative, distributed data-processing and data-driven visualisations. Moreover, several visualisation tools and visualisation frameworks including Prefuse[3] and Processing[4] exist to be used for creating tailor-made high-end visualisations. Different widget libraries such as Simile Widgets[5], Protovis[6] and Highcharts[7] present a lightweight approach to creating simple information visualisations.

In addition, a large number of Web APIs exist, providing data for mash-ups and visualisations in explicit formats. Examples include Google Maps API[8] for geographical data and Last.fm API[9] for data representing the listening habits of Last.fm users. Interestingly, major companies have also partly opened infrastructure for developers. For instance, Google App Engine[10] and Amazon Web Services[11] can be used to build RESTful applications (Richardson & Ruby, 2007) for processing and visualising data.) Furthermore, visualisation gadgets play an important role in Google Docs that present an example of a simple information management and visualisation environment for end-users.

However, from visualisation designer's point-of-view, there are still some issues in building visualisations. For example, many of the existing collaborative data-processing and visualisation services insist, by design, that both the visualisation data and the produced visualisations are public to the users of the service. Consider, for instance, IBM Many Eyes, where the user first uploads and publishes the data set into the service before creating a visualisation. The developers of Many Eyes report that many of the users tend to anonymise their data before publishing them to the service for visualisation (Danis, Viegas, Wattenberg & Kriss, 2008). Further, users often create individual visualisations and place them into their own blogs or other online social systems, instead of discussing the visualisation directly in Many Eyes. Effectively, this means using the service as a "community component" instead of an online community as such.

Further, while these kinds of application-based visualisations usually provide users the possibility to copy or clone existing data and pipeline structures, adding new types of components may not be possible. The creation and management of both visualisations and data sets in existing applications is often done manually, rather than by providing access to programmatic use. This is reasonable for securing resources, but limits the repertoire of applications. We see that there is a need of building general-purpose component-based visualisation pipelines. Ideally, these pipelines should be capable of authenticating to various data sources, handling sensitive data, using existing chains of trust and other information in defining the visual range of data and visualisations. The component design should further enable sharing computing capacity and other resources among trusted parties, and allow creating visualisations to specific groups of people.

In principle, many of the challenges in visualisation data processing and pipeline design, can be achieved by utilisation of a P2P infrastructure. In a successful P2P network, existing structures can be leveraged to provide a socio-technical framework for scaling the visualisation architecture as well. Once the network is established, data and services can be efficiently shared between trusted

1 http://pipes.yahoo.com
2 http://many-eyes.com
3 http://prefuse.org
4 http://processing.org
5 http://www.simile-widgets.org/
6 http://vis.stanford.edu/protovis/
7 http://www.highcharts.com/
8 http://code.google.com/apis/maps/
9 http://last.fm/api
10 http://code.google.com/appengine/
11 http://aws.amazon.com/

parties. With a sophisticated P2P infrastructure, the network can be scaled from local communities to a global, scale-free network.

We believe that a suitable mixture of community practices, data mining methods, and component techniques provides a sufficient basis for information visualisation in a large and heterogeneous network (see e.g. Nykänen, Salonen, Haapaniemi, Huhtamäki, 2008). Further, we claim that the component-based approach presented in this article is natural in a P2P setting, highlighting the balance between globally specified top-down application interfaces and the ease of quickly implementing locally motivated applications with relatively little global perspective.

In its default configuration, Wille runs on a local machine. This is a powerful feature from visualisation data management point of view because it enables the use of sensitive data as a source for visualisation applications without the need to publish it. Examples of possible sensitive sources include system log data (used e.g. in Social Network Analysis, SNA), personal browsing history, contact networks, emails and others. Moreover, Wille can be used to crawl many of the data sources automatically and in cases where automation is not possible, visualisation user can log into source systems to collect the data and save it locally for Wille to access. The specification of Wille's P2P configuration was also considered in Deliverable D10.11 (Huhtamäki, Nykänen, Salonen, 2009c).

3. Wille Developer Toolkit

Wille Developer Toolkit is the collection of resources that allows visualisation developers to create new applications and services on top of Wille. On an abstract level, Wille Developer Toolkit consists of 1) Wille2 Framework, 2) related documentation and tutorials and 3) a catalogue of reusable components.

In this chapter, we will provide a brief overview of the contents and functionality of this toolkit. Specifically we do not cover practical usage of Wille, or details on any specific components. For more practically oriented documentation on visualisation development with Wille, see the website[12].

3.1. Architecture

The objective of Wille2 is to function as a component-based data pre-processing and visualisation system for distributed and peer-to-peer (P2P) environments. Wille provides a light-weight core that can be extended by creating and integrating components. By scripting pipelines of component execution, interactive views to data can be created.

In a layered architecture, Wille2 can be modelled as follows (Figure 3.1.1):

- Instances of Wille including Wille Framework itself are executed with Python (either CPython or Jython) on top of a local run-time platform. Each instance of Wille include containers for Wille Services and Wille Apps.
- Wille Services can be deployed to and executed from Wille Service Container. Similarly Wille Apps can be deployed to and executed from Wille Applications Container. Wille Apps can access services from local and remote Service Containers, as well as with external web APIs and other end-user (GUI) applications.
- Wille has been designed to run both on Web Based Systems and P2P runtime environments.

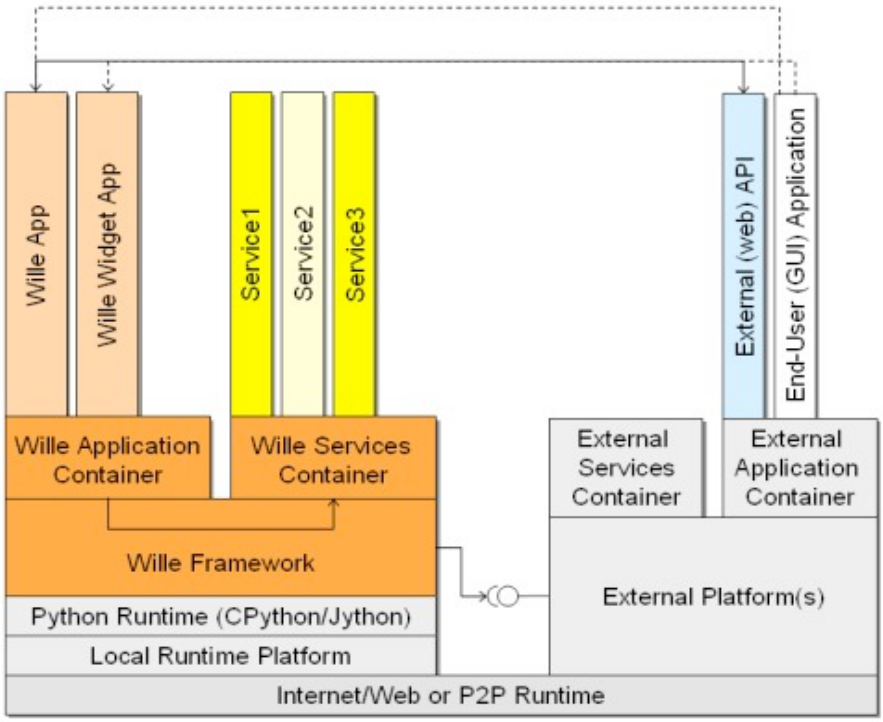


Figure 3.1.1. *Overview of Wille 2 Architecture*

12 http://wiki.tut.fi/Wille/

Several key concepts and components within the architecture can be identified:

- *Wille Services* are stateless services that can be used for specific data (pre-)processing tasks. When executed, services transform given input parameters into a single output result. A service itself may invoke other services and APIs to perform the required data processing.
- *Wille Client* is part of Wille Framework that allows search and execution of Wille Services. It can also carry user- and session-specific information including user's authentication and authorisation credentials in a keyring. By composing several service executions with Wille Client, *dynamic data processing pipelines* are created
- *Wille Apps* are web-based (visualisation) applications that can use Wille Client for various tasks
- *Wille Widget* is a reusable Wille App that can be integrated with other Wille Widgets to create a dashboard of visualisations
- *Wille Server* is an application that allows deployment and sharing of Wille Services and Apps, and therefore Wille Widgets as well
- *Wille Scripts* are scripts that use parts of Wille, especially Client, but have not been integrated either as Apps or as Services.

In a basic usage pattern, visualisation development would start by creating simple Wille Scripts. Later on, when the visualisation matures, individual Wille Services and Apps could be created based on the initial script. Eventually, a visualisation would consist of several Wille Apps and Services.

Different data transformations could be implemented as Wille Services. A pipelined services execution could be then orchestrated either from an aggregating Wille Service or from a Wille App. In order to integrate existing visualisation components and to provide a graphical user interface, Wille Apps could be created. At the point in which several Apps need to share same visualisation component, Wille Widgets could be then created and reused between multiple Apps.

More information on Wille's architecture, core process descriptions and OKS integration, please see Deliverable 10.12 (Huhtamäki, Nykänen, Salonen, 2009a).

3.2. Wille2 Framework and Core

Wille2 Core is a release of Wille2 framework that has been bundled with a minimalistic set of components that aid visualisation and Wille component development. The latest version of the Wille2 Core can be obtained from the Download page[13].

The key component in the core is the Wille2 Python Library. The Python library has been actively developed since 2007, and several versions of Wille2 were released during the project:

- First version, 2.0b was released 26th of June in 2009.
- Wille 2.1 was released in October 2009, as a minor version upgrade from 2.0b to 2.1.
- The latest feature adding release was version 2.2, released in 9th of April in 2010.
- At the time of the writing, the latest stable version of the framework is 2.2.1 (April 26th).

For more comprehensive information on key changes, refer to Wille's ChangeLog[14].

13 http://wiki.tut.fi/Wille/Download
14 http://wiki.tut.fi/Wille/ChangeLog

Note that in earlier versions of Wille2, tutorials on guides on Wille's usage were provided as part of OPAALS Deliverable 10.12 (Huhtamäki, Nykänen, Salonen, 2009a). Especially for reasons of sustainability and the possibility to updates, this content has now been extended and delivered as part of Wille Wiki:

- For a brief usage guide on installing and getting started with Wille2, see GettingStarted[15].
- A tutorial on using different parts of Wille2 including Wille Client and Wille Service, is available at the Tutorial page[16]
- Working and documented examples are provided in the Examples page[17]
- Information on Wille2 troubleshooting[18] is also available in the wiki

3.3. Outline of Components

A catalogue of different components (Apps, Services, and scripts) for Wille exists. The following table contains descriptions of the components, that are up-to-date and available for the current version of Wille.

For each component the following information have been catalogued:

- Type of the component (App, Service, Script), reflecting Wille's package structure
- Component name
- Component's description
- Component's profiles
- Package in which the component is located at

The latest and the most up to date version of the component catalogue is available online at the Components page[19] in the Wille Wiki. A snapshot of this catalogue is provided in Table 3.3.1.

Table 3.3.1. *A snapshot of Wille component catalogue*

Type	Name	Description	Profiles	Package
app	fileserver	A simple file server App. Serves all files placed under "shared" folder		Core
app	requestsniffer	App for inspecting ("sniffing") information that gets passed to Wille Apps		Core
service	Tidy	HTML Tidy fixes mistakes in HTML and assists in HTML formatting in many ways.	- cleaner - transformer	Core
service	requestsniffer	Service for inspecting ("sniffing") information that gets passed to Wille services	- examples	Core
service	weather	Weather report example. Demonstrates how a Wille Service can invoke an HTTP web service, and a SOAP/WSDL WS service	- examples	Core
service	xml2json	XML to JSON converter example	- examples	Core
app	timeline	Integration of SIMILE Timeline as a Wille Widget	- rss - widgets	RSSExplorer

15 http://wiki.tut.fi/Wille/GettingStarted
16 http://wiki.tut.fi/Wille/Tutorial
17 http://wiki.tut.fi/Wille/Examples
18 http://wiki.tut.fi/Wille/Troubleshooting
19 http://wiki.tut.fi/Wille/Components

Type	Name	Description	Profiles	Package
app	tableviewer	HTML Data Table Viewer Widget	- rss - widgets	RSSExplorer
app	dataselector	Widget for RSS data aggregation and filtering	- rss - widgets	RSSExplorer
app	rssexplorer	An application for visual exploration of web feeds	- rss - visualise	RSSExplorer
service	rss2json	Extracts data from RSS or Atom feeds and converts it to JSON format	- rss - transformer	RSSExplorer
app	sna-opaalswiki	Facilitates SN analysis and visualisation in OPAALS Wiki	- opaals - sna - wiki - visualise	SNAToolkit
app	sna-guigoh	Facilitates SN analysis and visualisation in OPAALS Guigoh	- opaals - sna - visualise	SNAToolkit
app	sna-innovation ecosystems	Facilitates SN analysis and visualisation on basis of Innovation Ecosystems Consortium data.	- sna - example - visualise	SNAToolkit
app	crawler-opaalswiki	Creates a local copy of the wiki.opaals.eu content in valid XHTML format. The files are listed in *index.xml* file. Note that the update process may take an hour to run on a modern workstation.		SNAToolkit
app	sna-example	To be used as a template if one wants to create an SNA service of one's own .	- sna - example - visualise	SNAToolkit
service	reader.guigoh	Fetches a copy of data on social network within guigoh.opaals.org. The service is a wrapper for a Java component originally developed by Lia Carrari from IPTI.	- sna - example - visualise	SNAToolkit
service	resourcefetcher-1file	Logs into a MoinMoin wiki and fetches a resource (a wiki page or attachment).	- sna - reader - wiki	SNAToolkit
service	resourcefetcher	Logs into a MoinMoin wiki and fetches a set of resources (wiki pages or attachments).	- sna - reader - wiki	SNAToolkit
service	wiki.moinmoin.search	Lists resources from a MoinMoin wiki (now only wiki.opaals.eu) on basis of specified criteria.	- sna - reader - wiki	SNAToolkit
service	wiki.moinmoin.pagehistory	Scrapes the history information from an HTML-formatted MoinMoin wiki history page.	- sna - reader - wiki	SNAToolkit
service	wiki.pagehistory.collector	Collects the contribution history of a set of wiki.opaals.org pages selected by the given search criteria.	- sna - aggregator - wiki	SNAToolkit

Type	Name	Description	Profiles	Package
service	wiki.pagehistory.slicer	Extracts a slice from the wiki history. For now, supports only time-based slicing. Can be generalised to support any kind of contributions.	- sna - wiki - filter	SNAToolkit
service	sna.metrics	Calculates key figures from given a XML file. Uses Wille graph format as input.	- sna - analysis	SNAToolkit
service	sna.wiki.onemode.dynamic.matrix	Creates a matrix representation on basis of an event log.	- sna - analysis	SNAToolkit
service	sna.graph2graphml	Creates a representation of a socio-graph in GraphML format. Please note that the component supports Vizster compliant dialect of GraphML. Takes toolkit's internal graph representation as input.	- sna - serialiser	SNAToolkit
service	sna.graph2pajek	Creates a representation of a socio-graph in Pajek format. Uses toolkit's internal graph representation as input.	- sna - serialiser	SNAToolkit
service	xslt-saxon[20]	XSL (Extensible Stylesheet Language) 2.0 Transformer running as a Wille service. Uses Saxon 8.8.	- sna - transformer	SNAToolkit
service	snametrics[21]	A general-purpose service for computing social network characteristics from socio-matrix data.	- sna	SNAToolkit

Note that a new feature, introduced in Wille version 2.2, is profiles. Profiles are arbitrary tags that can be associated to Wille Services and Apps. Multiple profiles may be assigned to each individual component. The primary use case for profiles is to use them to tag and eventually package components into individual release packages. For instance, Wille SNA Toolkit delivers all components assigned to *sna* profile.

The wrapper design in Wille allows components written virtually in any technology to be adapted to Wille. Ideally, this design allows rapid integration of a component based on any language or tool. Also the components listed in the table have been implemented with various technologies. In a typical integration use case a Java-based component is integrated by invoking it from command-line in a service wrapper.

Currently all the components listed in the table are up to date with Wille. In addition, components integrated to earlier versions of Wille2, can be still used as well. A list of these components is provided in Deliverable D10.12 in chapter 1.1 (Huhtamäki, Nykänen, Salonen 2009a). Open source components from Wille1 are usable as well. For instance SOMA[22], an implementation of self-organising maps algorithm in Java (Salonen, 2007), was implemented and integrated to Wille1. SOMA can be integrated into Wille2 as well, either as a standalone application or as a Wille Widget via Java Applet.

A comprehensive list of these components can be found at Appendix G of Deliverable D10.6 (Haapaniemi, Huhtamäki, Kortemaa, Mannio, Nykänen, Salonen, 2007). In Wille2, some of these components have become obsolete; it readily provides easier and more efficient ways of performing similar tasks.

20 http://wiki.tut.fi/Wille/XSLTransformer
21 http://wiki.tut.fi/Wille/SNAMetrics
22 http://wiki.tut.fi/Wille/SOMA

4. Wille Visualisation Catalogue for OKS end-users

In this chapter, we will present a catalogue of Wille visualisations that have been designed for OKS end-users. For each of the visualisations, we will present an introduction and a technical overview, as well as a case study of the tool within the context of OPAALS.

4.1. RSS Explorer

In this section, we will introduce *RSS Explorer:* a general purpose application for aggregating and visually exploring web feeds, written on top of Wille2. In addition, the implementation and overall architecture of the application is described. We also describe and discuss the design of a Wille Widget and Dashboard system that emerged from the development of RSS Explorer. The section is concluded with general discussion and remarks of the topic. For a more comprehensive description of the tool and a usage guide, see its wiki page[23].

4.1.1. Introduction

An important part of our work in OPAALS involved support community building by creating visualisations from OPAALS and OKS data. Within these tasks, we often found out that the most time-consuming part of the visualisation developed was automated extraction of data from source systems and their conversion to more general-purpose data formats. We often ended up writing components that were specialised to dealing with data extraction in very specialised settings.

While the resulting visualisations may be very useful for the specific situations they have been designed to work in, in order to visualise data from new or emerging systems, new components would need to be written. This leaves a very high overhead between adoption of new systems and their visualisation. However, as new systems are being developed and adopted, it still might be useful to provide some - even rudimentary - visualisations with some insight to data in these systems, while more sophisticated visualisations are being developed. We faced this very situation in OPAALS on many occasions. For instance a distributed storage application with semantic search was developed by IITK in parallel to our visualisations for it.

As a solution to this bootstrapping issue, we developed RSSVis, a prototype visualisation tool that provides rudimentary visualisations to data in as generic (web) data format as possible (Salonen & Huhtamäki, 2010). As the data format we chose RSS and web feeds. Rationale for choosing these formats include the following:

- Web feed formats have been standardised
- Data in RSS and web feeds is readily available in many existing (and upcoming) systems
- Mature components for processing web feeds are easily available
- Web feed formats provide a minimal, but easily extendible set of attributes for semantically rich descriptions of items

RSS Explorer was created and published by further refining and refactoring our prototype tool. Several new features in comparison to the prototype tool have been added. Firstly, a built-in feed aggregator and filtered was added supporting aggregation of both RSS and Atom feeds. Visualisation widgets from the prototype (Simile Timeline and Dataviewer) were refactored as Wille Widgets. Finally, support for *Visualise with Wille* browser extension[24] was added for automated feed data collection.

23 http://wiki.tut.fi/Wille/RSSExplorer
24 http://wiki.tut.fi/Wille/VisualiseWithWille

As a brief description, RSS Explorer is a general purpose application for aggregating and visually exploring web feeds. As a visualisation, RSS Explorer seems to work best for perceiving small-scale features on smaller datasets (less then 500 items) that can be later generalised to larger datasets and analysed with other visualisations.

In the next section, we will give a brief description of the implementation of RSS Explorer on top of Wille2. Secondly, we will present an example, highlighting different features of the tool in the given context. Finally, we will discuss the tool's applicability and features on a wider scope.

4.1.2. Implementation

RSS Explorer is implemented on top of Wille2. It utilises several of the features introduced in Wille 2.2 and 2.2.1 releases, especially related to use of Wille Apps.

In an abstract architecture, RSS Explorer is implemented by inheriting Wille Dashboard and Widgets (Figure 4.1.2.1). RSS Explorer is created by inheriting Dashboard: *init* method is overridden to instantiate the Dashboard with Dataselector, Timeline and Timeviewer Widgets. Widgets, on the other hand, are created as new classes that inherit Widget. Similarly default methods may be overridden to provide widget-specific behaviour.

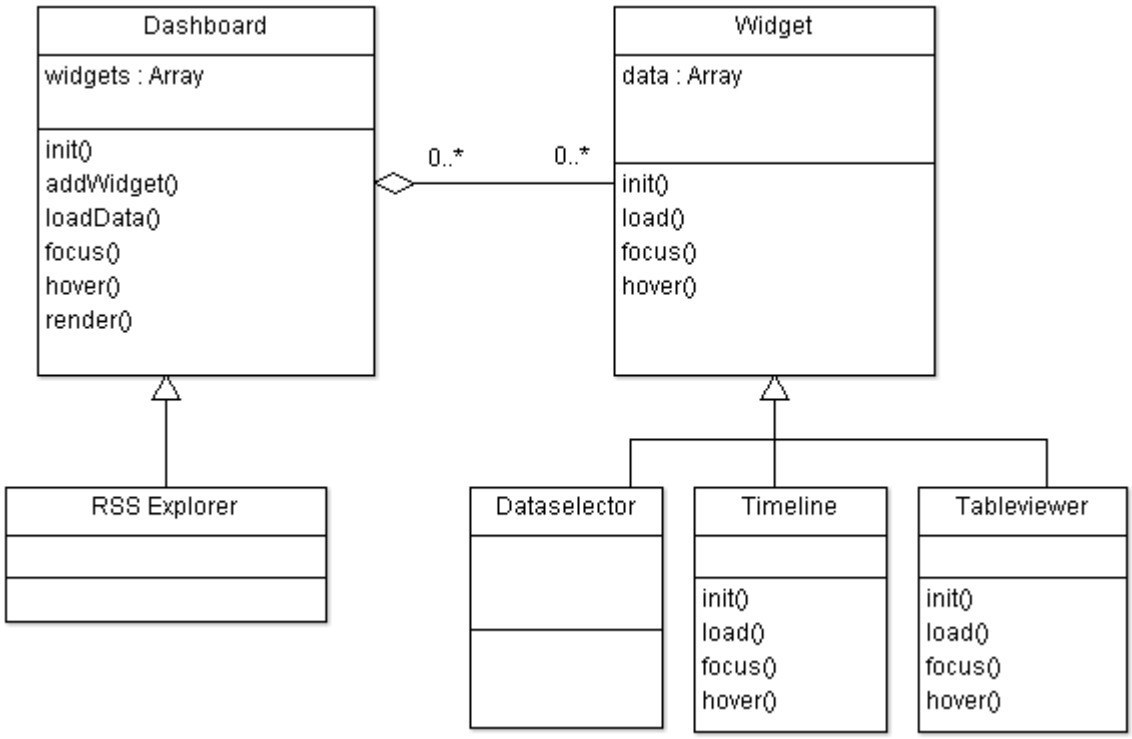


Figure 4.1.2.1. *Class diagram for RSS Explorer*

Both Dashboard and the widgets have several methods that are invoked on specific events. The current implementation supports the following events: *load* (or *loadData* in Dashboard) is invoked when new data is loaded, *focus* is invoked when the user focuses on a specific data item with a mouse click, *hover* is invoked when user moves mouse over or out of a specific item. When a method in Dashboard is called, by default the corresponding call is delegated to all widgets in the Dashboard. For instance, when *loadData* is invoked in Dashboard, load method is invoked for all widgets as well.

Technically, the implementation is split between server-side (Python) code and client-side JavaScript. Methods in Dashboard have been implemented in JavaScript, but partly in Python as well. However, in order to share the widget source files, a simple Wille App needs to bewritten in Python. Widget base class is an abstraction, and thus, is rather a best practise than an interface.

Following Wille's naming and packaging schemes, the different components in RSS Explorer are split between Apps and Services as follows:

Apps:

- *dashboard*: Contains the base implementation of Wille Dashboard
- *rssexplorer*: The main application for RSS Explorer that combines all widgets into a single dashboard.
- *dataselector*: Wille Widget that selects and sends data from given data sources to other widgets for visualisation
- *timeline*: Wille Widget that visualises data with SIMILE Timeline[25]
- *tableviewer*: Wille Widget that creates a sortable HTML table representation of the data

Services:

- *rss2json*: A utility service for converting given RSS or web feed into a specific JSON format for further processing. Used by dataselector.

As an example of how widgets work, let us demonstrate how *tableviewer* widget was built.

Tableviewer is implemented as a single Wille App. First *tableviewer* subdirectory under *apps* was created as a place holder for all code, markup and files related to the implementation of the widget. The name of the subdirectory is also used as the name of the widget App. In addition, *willeapp.properties* was created to describe properties of the App for Wille:

```
profile=rss,widgets
description=HTML Data Table Viewer Widget
```

For widgets, very minimal properties are sufficient. The more widgets become available, the more important the widget description in properties become. In tableviewer, we assigned the widget to several profiles: *rss* profile was added to package the widget together with other components in RSS Explorer, *widgets* profile was added to identify that the App was created as a widget rather than as a full, stand-alone App. A description of the widget was provided as well. The description is visible in Wille's management view[26] as well as in generated online component catalogue[27].

The actual implementation of the widget, is split into several files. In *tableviewer.py* (Listing 4.1.2.1) we implement Wille App that functions the container for the widget.

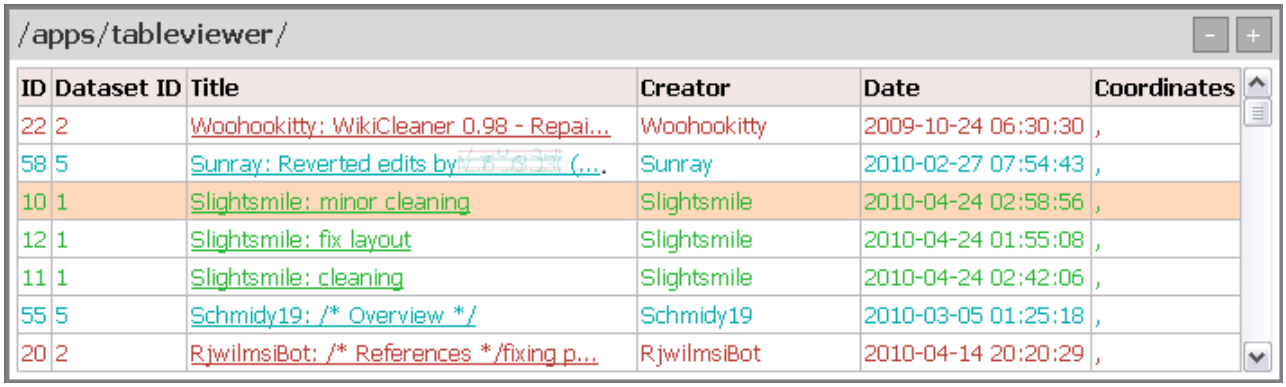
Listing 4.1.2.1 *Implementation of Tableviewer App (tableviewer.py)*

```
 1 import os
 2 import wille
 3 import wille.template
 4 from wille.views.contrib import ServeFiles
 5 class TableViewer:
 6         def GET(self, request):
 7                 return wille.template.render('template.html',
                                                template_dir=request.workdir)
 8 urls = (
 9         ("/", TableViewer, '.'),
10         ("/(.*)", ServeFiles, '.'),
11 )
```

25 http://www.simile-widgets.org/timeline/
26 E.g. http://localhost:8080/management/
27 http://wiki.tut.fi/Wille/Components

Minimally, we need to provide URL[28] mappings (*urls* variable, in lines 10-14) and a default handler for the widget (lines 6-8). In this widget, the handler simply renders *template.html* and returns it. The template includes the widget code and its required JavaScript libraries and CSS[29] styles. We also have files under the folder that need to be served as well. For that purpose, handler for serving the files is required as well (line 13).

The actual widget is implemented in a separate JavaScript file (*tableviewer.widget.js*). In the Widget script file, we implement handlers for widget methods: *init*, *load*, *focus* and *hover*. Note that the widgets always require a Dashboard to call back the widget methods. For example in Figure 4.1.2.2 Tableviewer is running inside a Dashboard implemented by RSS Explorer. The Tableviewer provides a hook for *onmouseover* and *onmouseout* events and calls back Dashboard to invoke *hover* method. As all widgets receive the event, the hover effect can be rendered to all widgets simultaneously.


/apps/tableviewer/

ID	Dataset ID	Title	Creator	Date	Coordinates
22	2	Woohookitty: WikiCleaner 0.98 - Repai...	Woohookitty	2009-10-24 06:30:30	,
58	5	Sunray: Reverted edits by [redacted] (...	Sunray	2010-02-27 07:54:43	,
10	1	Slightsmile: minor cleaning	Slightsmile	2010-04-24 02:58:56	,
12	1	Slightsmile: fix layout	Slightsmile	2010-04-24 01:55:08	,
11	1	Slightsmile: cleaning	Slightsmile	2010-04-24 02:42:06	,
55	5	Schmidy19: /* Overview */	Schmidy19	2010-03-05 01:25:18	,
20	2	RjwilmsiBot: /* References */fixing p...	RjwilmsiBot	2010-04-14 20:20:29	,

Figure 4.1.2.2. *Example view from Tableviewer running inside RSS Explorer Dashboard*

Note that in this section we provided a description of creating a Widget in Wille 2.2.1 with RSS Explorer. Currently, the Wille widget system is developed as part of the RSS Explorer branch. As the system matures, it is planned to be included as part of Wille Core instead, and may be refactored in the way. In future versions of Wille we anticipate that creation of new widgets will be more streamlined, hence, requiring less steps for minimal widget creation.

4.1.3. Case: Wiki Activity Visualisation

As a case example of using RSS Explorer, let us consider its usage for wiki activity visualisation. For full instructions on using RSS Explorer, please see its usage guide within its wiki page[30].

A wiki, in short, is a collaboratively edited website that also allows creation of new content and hyperlinking. While many popular wikis, such as Wikipedia, WikiTravel and WikiHow, are public, in work environments it often makes sense to restrict access to the sites for a specific working group or project. Yet similarly within enterprises, wikis have the potential to gather and make tacit knowledge explicit while satisfying key knowledge management needs (O'Leary D. E., 2008).

In OPAALS, our team's work involved the use of numerous wiki sites. Some of the wikis were public, however access to some of the sites was restricted to smaller working groups within the project. Working with Wille2 development, required staying up to date as well as authoring at multiple wikis. An overview of these wikis is provided in Table 4.1.3.1.

28 Uniform Resource Locator
29 Cascading Style Sheets
30 http://wiki.tut.fi/Wille/RSSExplorer#Brief_Usage_Guide

Table 4.1.3.1. *An overview of wikis used in Wille2 development*

Name	Wiki Engine	Description
OPAALS Wiki[31]	MoinMoin	OPAALS's wiki. Used for various documentation and collaboration tasks in OPAALS.
Wille Wiki[32]	Foswiki	Wille's wiki site. Used for documenting and publishing Wille2 and related tools and visualisations.
Flypeer Wiki[33]	Kenai.com wiki	Flypeer is an implementation of OPAALS P2P in Java. The wiki is used for documenting guides and examples of Flypeer's usage, required for developing integration of Flypeer and Wille
TUT's internal OPAALS wiki	MoinMoin	TUT's internal wiki workspace for OPAALS. Used for instance to keeping track of tasks and documenting meetings.
HLAB's internal wiki	MediaWiki	HLAB's internal wiki. Used for instance to internally document tools and processes in TUT/HLAB.

A challenge in working with several wikis is that it is generally hard to perceive the overall pattern of edits and collaborations that happen either simultaneously or between the wikis. For a user of these wikis, it might be helpful to provide some overview to the daily activities. Let us consider how RSS Explorer could be used to visualise and provide some insight of the activity in these wikis.

Wille Server with RSS Explorer was started with a customised user data folder for data collection:

```
python wille-server.py -u ../data/my-wikis
```

We chose to extract information about recent changes in the above wikis and use it as source data in RSS Explorer. In order to assist the extraction of RSS data from wikis requiring authentication, we installed and used Visualise With Wille browser extension[34]. With the extension, 5 files with recent changes data were stored to the chosen user data folder (see Listing 4.1.3.1).

Listing 4.1.3.1 *Listing of directory content from the chosen Wille user data folder*

```
Directory of C:\work\data\my-wikis

07.05.2010  15:32    <DIR>          .
07.05.2010  15:32    <DIR>          ..
07.05.2010  15:21           126 802 hlab
07.05.2010  15:19            22 602 kenai_com-Flypeer
07.05.2010  15:20            12 235 Opaals_eu
07.05.2010  15:22            13 032 tut-opaals-wiki
07.05.2010  15:24            12 079 wiki_tut_fi-Wille
               5 File(s)        186 750 bytes
               2 Dir(s)   8 001 675 264 bytes free
```

An overview of the visualisation of this dataset with RSS Explorer is provided in Figure 4.1.3.1.

31 http://wiki.opaals.eu
32 http://wiki.tut.fi/Wille/
33 http://kenai.com/projects/flypeer/pages/
34 http://wiki.tut.fi/Wille/VisualiseWithWille

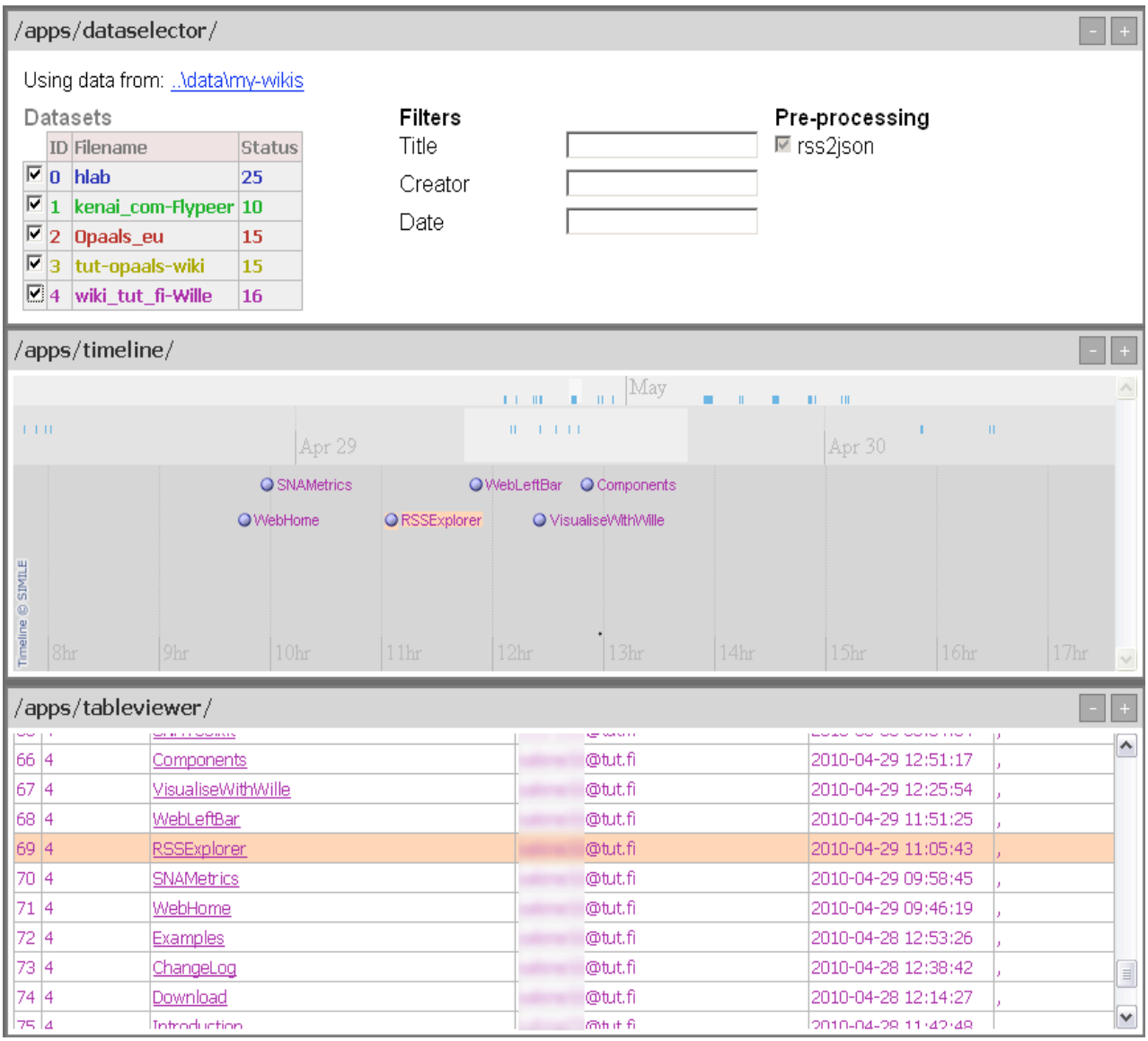


Figure 4.1.3.1. *An overview from RSS Explorer with Wiki Activity Data*

As a large scale trend in Timeline (top of the image), we can see that most of the wiki edits occur daily between 9 am and 2 pm, implying the wiki has been in the most active use during local office hours. In the single day timeline view, we can see that during April 29th only one of the wikis (Wille wiki) had been edited by a single user. Different wiki pages were edited at different times. When we click the links back to the wiki pages, we can confirm that indeed the wiki went through active editing phase.

When we navigate through the timeline to display activity in several other days, we discover that it is very typical that only one or two wikis at the most, undergo active editing in any given day during two weeks period. In many occasions, multiple editors work editing content in the same wikis. However, it is not typical that multiple wikis are active during same day. In addition, the visualisation suggests that various wikis are often used sequentially, with work occurring in single wikis in editing "bursts".

Following the initial analysis, we focused on the activity of a single user in the wikis, by using RSS Explorer's filters to filter out edits from all other users. Edits by a single user were more scattered in the timeline. Hence, we focused on the table representation of the data, as rendered by Tableviewer (Figure 4.1.3.2).

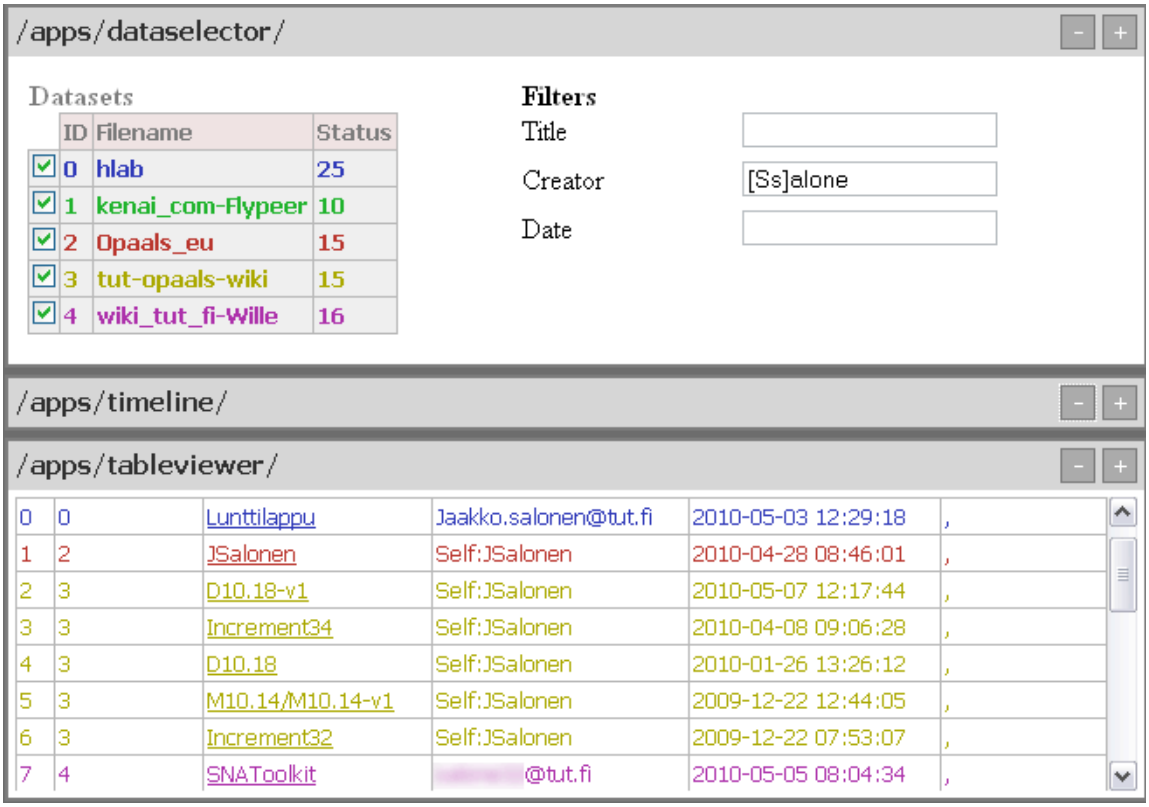


Figure 4.1.3.2. *RSS Explorer with data filtered for only matching user*

Based on the data, the selected user is active in all given wikis. This is of course expectable since the wikis were selected based on what a single user uses for OPAALS related work. Again, similar editing pattern could be detected: individual wikis were edited in bursts. We expected that in some cases, related content would be edited on several wikis simultaneously. However, in the given datasets, this was not the case. One explanation for this is that while similar content may be present in several wikis, the content between wikis is transferred by editing only the target wiki. This would not leave any traces in the editing history of the source wiki, and hence, would not be visible in our edit-only visualisations.

4.1.4. Discussion

All in all, with RSS Explorer some preliminary discoveries from the wiki editing process could be identified. Especially - based on the given data - wiki editing seems to be happening sequentially, rather than in parallel between several wikis: within each wiki the edits happen in "bursts" of work.

A problem with the case is that the visualisation did not provide us with very conclusive insights in terms of larger scale trends of the data. However, it provided us with initial points of exploration and hence, can be depicted as a visualisation that enables us to perceive initial, small-scale features of the data. Based on these insights, we could continue our data collection and analysis with tools more fit to be used with larger datasets.

Note that the dataset we used, was rather small (81 items in total). While we are confident that RSS Explorer could be used for much larger datasets as well (around 500-1000 items), it does not scale very well for large datasets. Since all data in the visualisation is processed with JavaScript, especially rendering may be slow with thousands or more items. For analysis of large datasets, use of visualisations based on more efficient technologies may be desirable. For instance DataBox visualisation (see chapter 4.3) implemented in Java may be a more efficient choice for visualisation of larger datasets.

In terms of extensibility, Wille Dashboard (and RSS Explorer) provides the rudimentary technical framework for adding any kind of web-based visualisation widgets. Wille Dashboard relies on jQuery[35], and therefore target component may need to be compliant with it. Technically, a catalogue of existing JavaScript components can be integrated to Wille Dashboard as widgets. Especially integration of components from Ext JS[36], Protovis[37], and Google Chart Tools[38] have already been considered. An aspect of the Wille Dashboard that may need rethinking in the future is the point of separation between server and client side code: currently, both Python and JavaScript need to be written to implement even the simplest possible Wille Widgets. In more rapid widget integration, it is desirable to provide users with more light-weight means of integration for easier and more rapid Wille migration. It is very likely that this point of development will be taken in future versions of Wille Dashboard.

Note that in this study, the the analysis was merely based on what was visible from the tool with the given data. If prior knowledge of the editing process would be combined with the visualisations, more results and insights could be potentially found. For this analysis, a further study is necessary.

4.2. Wille SNA Toolkit

In this section, we will discuss the social network analysis (SNA) use case and application on the context of the Wille visualisation system. For an overall description of Wille SNA Toolkit including a step-by-step tutorial for using the components delivered with the toolkit and utilising the toolkit to develop your own SNA applications, please refer to its homepage under Wille wiki at http://wiki.tut.fi/Wille/SNAToolkit.

4.2.1. Introduction

Social network analysis is a research field that studies the structure of social actor networks. In practise, the analysis is performed as a combination of formal methods (for instance indicators computed with matrix algebra) and visual analysis of actor networks (for instance qualitative hypotheses based on contextual information). The use of pictorial images in representing social configurations is important because "[i]t allows investigators to gain new insights into the patterning of social connections, and it helps investigators to communicate their results to others" (Freeman, 2009).

From early days of OPAALS Network of Excellence, *data-driven visual social network analysis* has been one of the application domains of Wille visualisation framework. SN analysis has served both (1) as one of the case domains of Wille generating requirements to the Wille toolkit core and (2) a source of useful visualisation tools and data-processing algorithms that we have utilised in building information visualisation solutions to different usage contexts.

In OPAALS, visualisations of social networks are targeted both to the research of social networks and related issues *per se* (cf. Huhtamäki, Salonen and Nykänen, 2009b) as well as other members of OPAALS (Huhtamäki, 2007). Wille social network visualisation tools provide also support for numerical SN analysis. Wille was used, for example, to collect, aggregate and pre-process data for Colugnati and Carrari Lopes (2010) for the analysis of the evolution of OPAALS wiki contributor network.

We see that Wille SNA Toolkit can significantly contribute to the objective set by Linton C. Freeman - a social network visualisation expert with a strong background in the field - according to whom "we have made progress in developing programs for visualisation. We can look forward to

35 http://jquery.com/
36 http://www.extjs.com/
37 http://vis.stanford.edu/protovis/ex/
38 http://code.google.com/apis/charttools/

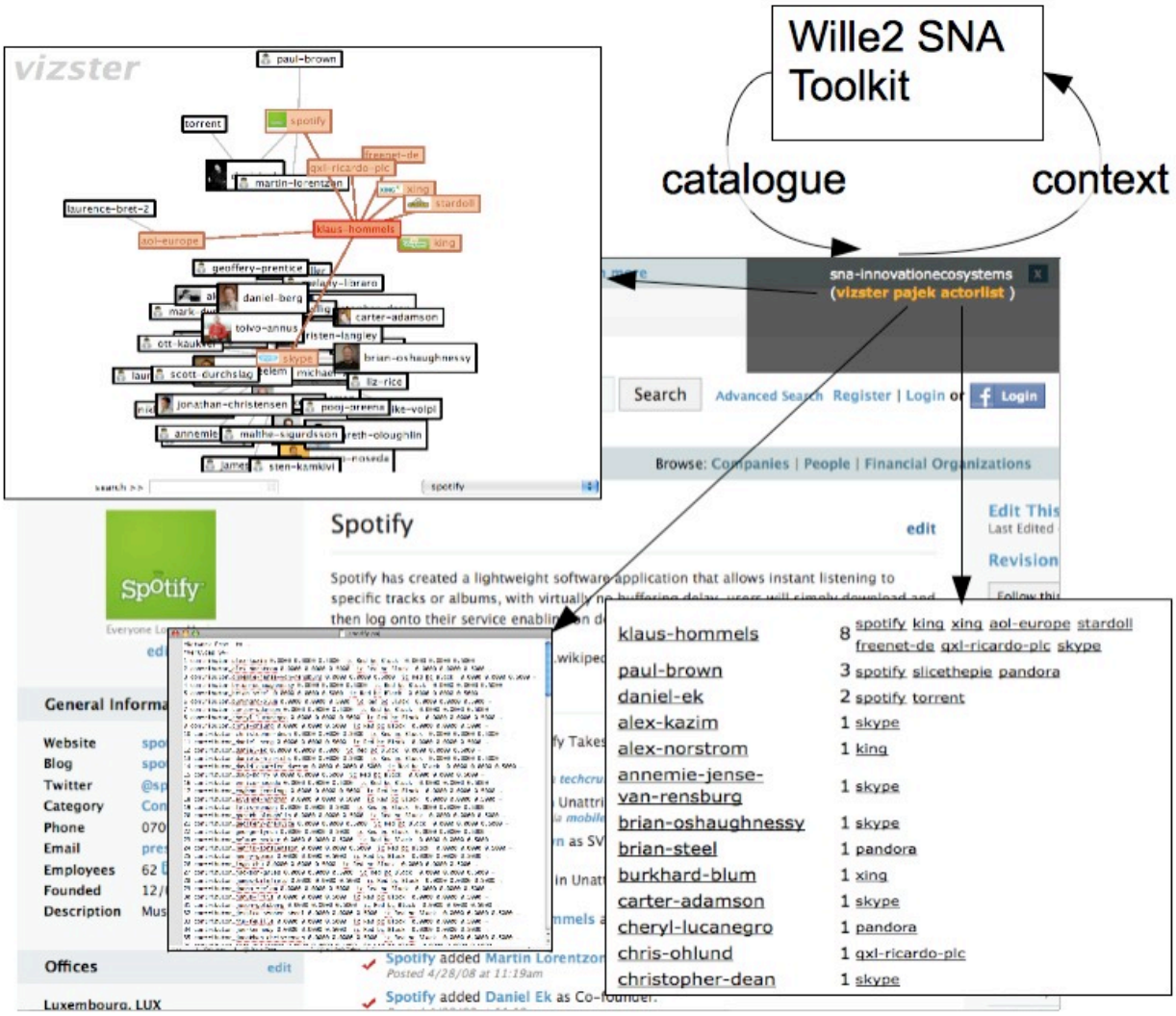


Figure 4.2.1.1. *Diagram of the operation principles of Wille SNA Toolkit*

similar progress in developing database programs designed to facilitate the storage and retrieval of social network data. But the real breakthrough will occur when we develop a single program that can integrate these three kinds of tools into a single program. Only then will we be able to access network data sets and both compute and visualize their structural properties quickly and easily." (Freeman, 2000.) While we acknowledge the fact that there are many existing tools for collecting, processing and analysing social network data, we see that the design of Wille has many benefits in facilitating the SN analysis work (Figure 4.2.1.1).

Wille SNA Toolkit is implemented primarily as a collection of tools belonging to *sna* profile. Its main features include:

1. **Data preprocessing**: Wille is capable of connecting and logging into a number of data sources. Wille components provide means to distil, aggregate, transform and refine the data.
2. **SNA tools as Wille components**: Those SNA tools that are operable via a command-line interface can be wrapped into Wille components to be used in different Wille pipelines as Wille services.
3. **Data export**: SNA data export to social network analysis tools including Pajek[39], NodeXL,

39 http://pajek.imfm.si/

Vizster[40], Orange[41], R[42], and SPSS[43] as well as traditional spreadsheet tools with formats based on text transformations can be implemented straightforwardly.

The fact that Wille is designed to operate on the local machine enables the use local data sources for visualisation. This can be considered as a major benefit, since it allows analysts to work with for instance very sensitive, local-only data. Further, sensitive data can be aggregated (mashed) e.g. with data from public data sources and, again, visualised safely on the analyst's personal computer.

A more unique contribution to SNA work with Wille is enabling *context-sensitive SNA* in the spirit of *augmented browsing*. The concept and key technologies enabling this were introduced in our recent article on context-sensitive launching and light-weight integration of information visualisation applications (For a detailed description, see Salonen and Huhtamäki, 2010). The approach enables the delivery of SN visualisation tools to the context from where their source data is collected from. Our hypothesis is that this is a meaningful feature for social network investigators as it gives them the possibility to concentrate in the analysis process rather than working to collect the data and processing it to meet the requirements posed by the different SN analysis tools.

In addition to introducing the basic natives of social network analysis to developers using Wille and implementing features for streamlining the data harvesting and processing processes related to social network analysis done with Wille, Wille SNA Toolkit comes with a set of ready-made apps, services and scripts for SN analysis and visualisation. For this deliverable, we have collected and refined a set of valuable Wille components developed during the course of OPAALS and used them to assemble a collection of SNA application. Also entirely new components have been developed for the deliverable to meet the specific requirements of context-sensitive SNA.

In this deliverable, we concentrate in describing the means to conduct SN analysis with Wille SNA Toolkit in the context of the OPAALS Network of Excellence. As only OPAALS members have full access to both Guigoh and OPAALS Wiki, we include an additional example application, *sna-innovationecosystems*, to the delivery package. This application provides a a proof-of-concept demonstration of SNA Toolkit's usage in the context of the Innovation Ecosystems Consortium (http://innovation-ecosystems.org). See SNA Toolkit's documentation[44] on the Wille Wiki for more information on this example.

4.2.2. Implementation: Reusable SNA components for OPAALS

Two main Wille SNA applications are implemented for OPAALS, *sna-guigoh* and *sna-opaalswiki*. Further, Wille SNA Toolkit includes a set of components that are useful in building new SNA applications or any other information visualisation solutions using the OPAALS OKS infrastructure.

While Wille SNA Toolkit comes with a set of example applications, the main objective of the toolkit is to enable the development of new SN applications for OPAALS and in general. Related particularly to OPAALS technical infrastructure, the following components are available for accessing, processing and visualising OPAALS OKS data.

40 http://jheer.org/vizster/
41 http://www.ailab.si/orange/
42 http://www.r-project.org/
43 http://www.spss.com/
44 http://wiki.tut.fi/Wille/SNAToolkit

Apps:

- *sna-opaalswiki* facilitates SN analysis and visualisation in wiki.opaals.eu.
- *sna-guigoh* enables SN analysis and visualisation in OPAALS Guigoh environment (opaals.org.br).
- *sna-innovationecosystems* introduces means e.g. to set up a simple data proxies and visualise data with Vizster and other SN analysis and visualisation tools. Data is crawled from the Innovation Ecosystems Consortium dataset.
- *crawler-opaalswiki* creates a local copy of the wiki.opaals.eu content in valid XHTML format. The files are listed in file index.xml. Note that due to the nature of the crawling process, updating the data set may take an hour to run on a modern workstation.
- *sna-example* can be used as a template when you want to create an SNA service of your own.

Services for data access:

- *reader.guigoh* fetches a copy of data on the social network within guigoh.opaals.org. The services is a wrapper for a Java-based component originally developed by Lia Carrari Lopes from IPTI.
- *resourcefetcher-1file* and *resourcefetcher* are capable of logging into a MoinMoin wiki and fetching resources (wiki pages and attachments).
- *wiki.moinmoin.search* lists resources from a MoinMoin wiki (now only wiki.opaals.eu) on basis of specified criteria. Select either ALL for all the pages or a category name, such as *CategoryProject/WorkPackage*.
- *wiki.moinmoin.pagehistory* scrapes the history information from an HTML-formatted MoinMoin wiki history page.

Services for data collection:

- *wiki.pagehistory.collector* collects the contribution history of a set of wiki.opaals.org pages selected by the given search criteria.
- wiki.pagehistory.slicer extracts a slice from the wiki history. For now, it supports only time-based slicing. It can be generalised to support any kind of contributions.

Services for data processing and transformations:

- *sna.metrics* calculates key figures from input data given in an XML network format [45]
- *sna.wiki.onemode.dynamic.matrix* creates a matrix representation on basis of an event log. Produces data that enables the kind of analysis process that Colugnati and Carrari Lopes (2010) use.

Services for converting data to visualisable representations:

- *sna.graph2graphml* creates a representation of a sociograph in GraphML format. Uses toolkit's internal graph representation as the input.
- *sna.graph2pajek* creates a representation of a sociograph in Pajek format. Uses toolkit's internal graph representation as the input.

Additional Wille Services which were used:

- *tidy* HTML Tidy is a tool that was originally written by Dave Raggett of the World Wide Web Consortium (W3C). It is designed to fix mistakes in HTML, tidy up the layout (hence the name), assist with web accessibility, convert HTML to XHTML and many other things.
- *xslt-saxon* XSL (Extensible Stylesheet Language) 2.0 Transformer running as a Wille service. The service was used for applying various XSL transformations.

45 See chapter 4.2.2.1 for a definition

The components related to Guigoh and OPAALS wiki can be easily generalised to support other instances of these systems in addition to opaals.org.br and wiki.opaals.eu. While e.g. the wiki reader components only support instances of MoinMoin wiki, many of the components either already are general or can be generalised in a straightforward manner to support other wiki systems (or other collaborative tools) as well.

4.2.2.1. OPAALS-related data formats

Guigoh social network data is represented in format designed by SUAS and TUT teams. An excerpt of the data follows:

```
<?xml version="1.0" encoding="utf-8"?>
<NetworkDescription xmlns="http://www.fh-salzburg.ac.at/NetworkDescription">
    <node id="27" type="null" uri="jsalonen" name="Jaakko"
          profileuri="http://www.opaals.org.br/ProfileView.do?id=27">
      <name>Jaakko</name>
      <location lat="26.360111" lng="-98.782501" name="null, null"/>
      <connections>
        <connection strength="1" to="frauke" type="contact from Guigoh"/>
        <connection strength="1" to="marco" type="contact from Guigoh"/>
        <connection strength="1" to="reder" type="contact from Guigoh"/>
        <connection strength="1" to="tkurz" type="contact from Guigoh"/>
        <!-- Example truncated -->
      </connections>
    </node>
    <node id="32" type="null" uri="tkurz" name="Thomas"
          profileuri="http://www.opaals.org.br/ProfileView.do?id=32">
      <name>Thomas</name>
      <location lat="26.360111" lng="-98.782501" name="null, null"/>
      <servicePool type="onOffer">
        <service id="Guidelines Email" uri="Guidelines Email">
          <name>Guidelines Email</name>
          <description>guildelines email praxissemester internship</description>
        </service>
        <service id="PIIIKO" uri="PIIIKO">
          <name>POOOKO</name>
          <description>abstract of POOOKO</description>
        </service>
        <!-- Example truncated -->
      </servicePool>
      <connections>
        <connection strength="1" to="jsalonen" type="contact from Guigoh"/>
        <connection strength="1" to="jukka" type="contact from Guigoh"/>
        <!-- Example truncated -->
      </connections>
    </node>
    <node id="6" type="null" uri="jukka" name="Jukka"
          profileuri="http://www.opaals.org.br/ProfileView.do?id=6">
      <name>Jukka</name>
      <location lat="26.360111" lng="-98.782501" name="null, null"/>
      <connections>
        <connection strength="1" to="jsalonen" type="contact from Guigoh"/>
        <connection strength="1" to="tkurz" type="contact from Guigoh"/>
        <!-- Example truncated -->
      </connections>
    </node>
</NetworkDescription>
```

Guigoh data is also transformed to Wille network XML format for further processing. For representing wiki history data, we use a format already introduced in the Deliverable 10.12 (Huhtamäki, Nykänen, Salonen, 2009a):

```
<?xml version="1.0" encoding="utf-8"?>
<historyaggregate searchterm="CategoryTest">
   <actorlist>
      <actor id="JSalonen">
         <type>contributor</type>
      </actor>
      <actor id="Ossi">
         <type>contributor</type>
      </actor>
      <actor id="huhtis">
         <type>contributor</type>
      </actor>
   </actorlist>
   <pagelist>
      <pagehistory pageid="http://wiki.opaals.eu/CategoryTest">
         <contribution contributorid="huhtis" date="2009-07-08T09:17:26Z"
                       revision="2" size="85" type="edit"/>
         <contribution contributorid="huhtis" date="2009-07-08T09:15:22Z"
                       revision="1" size="18" type="edit"/>
      </pagehistory>
      <pagehistory pageid="http://wiki.opaals.eu/Ossi">
         <contribution contributorid="Ossi" date="2007-04-04T12:42:49Z"
                       revision="8" size="1146" type="edit"/>
         <contribution contributorid="Ossi" date="2007-04-04T12:40:58Z"
                       revision="7" size="1034" type="edit"/>
         <contribution contributorid="Ossi" date="2006-11-13T13:00:44Z"
                       revision="6" size="444" type="edit"/>
         <contribution contributorid="JSalonen" date="2006-11-02T10:32:59Z"
                       revision="5" size="436" type="edit"/>
         <contribution contributorid="Ossi" date="2006-10-30T09:41:22Z"
                       revision="4" size="420" type="edit"/>
         <contribution contributorid="Ossi" date="2006-10-30T09:40:51Z"
                       revision="3" size="0" type="edit"/>
         <contribution contributorid="Ossi" date="2006-10-30T09:35:40Z"
                       revision="2" size="403" type="edit"/>
         <contribution contributorid="Ossi" date="2006-10-24T14:18:56Z"
                       revision="1" size="374" type="edit"/>
      </pagehistory>
      <!-- Rest of the example truncated. -->
  </pagelist>
</historyaggregate>
```

In addition, Vizster-compliant GraphML and Pajek are used as export formats. While new input and output formats can be introduced relatively easily, we suggest keeping the number of internal formats to a minimum. This really fosters the reuse of existing components.

4.2.3. Case: Social Networks in OPAALS

Two main Wille SNA applications are implemented for OPAALS, *sna-opaalswiki* and *sna-guigoh*. The two main sources of SNA data in OPAALS are OPAALS Wiki[46] and OPAALS Guigoh[47], respectively.

Guigoh is a social networking platform providing the users with means to manage their social social networks, a collaborative editing tool, a web conference tool (chat, VoIP and whiteboard) and an agenda system for on-line journals (OKS Glossary, n.d.).

46 http://wiki.opaals.eu/
47 http://opaals.org.br/

The OPAALS wiki was the first of the collaborative tools taken into use in OPAALS. OPAALS wiki uses MoinMoin wiki engine. From early on, the OPAALS wiki has served as the main source for SNA data. Whereas Guigoh has replaced e.g. the Facts About Cards[48] in representing the profile data of OPAALS members, the contribution history of the wiki pages is still a significant source for SNA data.

Next, we will take a tour around OPAALS and see what kind of social networks can be visualised with Wille and how this is done in practice. Note that in order to go through the tutorial, you need to have an account for the OPAALS wiki and OPAALS Guigoh. The examples in the tutorial are tested with using Firefox and the Greasemonkey plugin in an OS X environment.

Please follow the instructions:

- Download a complaint version of Wille 2 Core from Download page (http://wiki.tut.fi/Wille/Download).
- Download the SNA toolkit from its homepage (http://wiki.tut.fi/Wille/SNAToolkit)

Open command-line, change working directory to the location of your Wille installation and start the server as follows:

```
python wille-server.py -d -p 8080 -f sna wille-server.py
```

To verify that Wille server with SNA Toolkit is up and running, open Wille's frontpage (http://localhost:8080/) with your web browser. Note that the above configuration starts the server in debug mode (*-d*) in port 8080 (*-p 8080*) and enables only components assigned to sna profile (*-f sna*). For more information on configuration and command-line options, see Wille tutorial's section on running Wille Server[49].

Also, in order to provide the toolkit with access to OPAALS wiki and Guigoh, you need to create a Wille keyring. For instructions on using keyring, see the corresponding section[50] in the tutorial.

The most straightforward way to use the applications of Wille SNA Toolkit is to start by installing the Visualise With Wille tool:

1. Select a Greasemonkey-compatible browser, such as Mozilla Firefox or Google Chrome
2. When using Firefox, an additional Greasemonkey add-on[51] needs to be obtained and installed as well.
3. Finally, navigate with your browser to */apps/visualise/visualisewithwille.user.js*[52] under your local Wille server in order to install Visualise with Wille Greasemonkey script

You should now have the toolkit installed with the visualisation probe. Next, let's go through some examples on SNA within OPAALS:

1. Login to OPAALS Guigoh (http://www.opaals.org.br/). In order to create a new account, an invitation may need to be requested from a existing network member.
2. When the Guigoh frontpage has loaded, check that you see the text "A Guigoh instance found" appear in the command prompt screen running Wille.

48 http://wiki.opaals.eu/FactsAboutGuide
49 http://wiki.tut.fi/Wille/Tutorial#Running_Wille_Server
50 http://wiki.tut.fi/Wille/Tutorial#Using_keyring
51 https://addons.mozilla.org/firefox/addon/748/
52 Located at http://localhost:8080/apps/visualise/visuailsewithwille.user.js when using port 8080

3. Move to a profile page of a Guigoh user[53]
4. On the up-right part of your browser frame, you should see a widget appearing
5. By clicking links from the box, different visualisations may be opened.

For instance, for a visualisation of the selected user's social network, click "vizster" link[54] from the widget. Launching the visualisation for the first time may take several minutes. This is the time when the *reader.guigoh* service is ran in order to extract data from the Guigoh. Finally, a view to Guigoh social network running in Vizster such appear, as depicted in Figure 4.2.3.1.

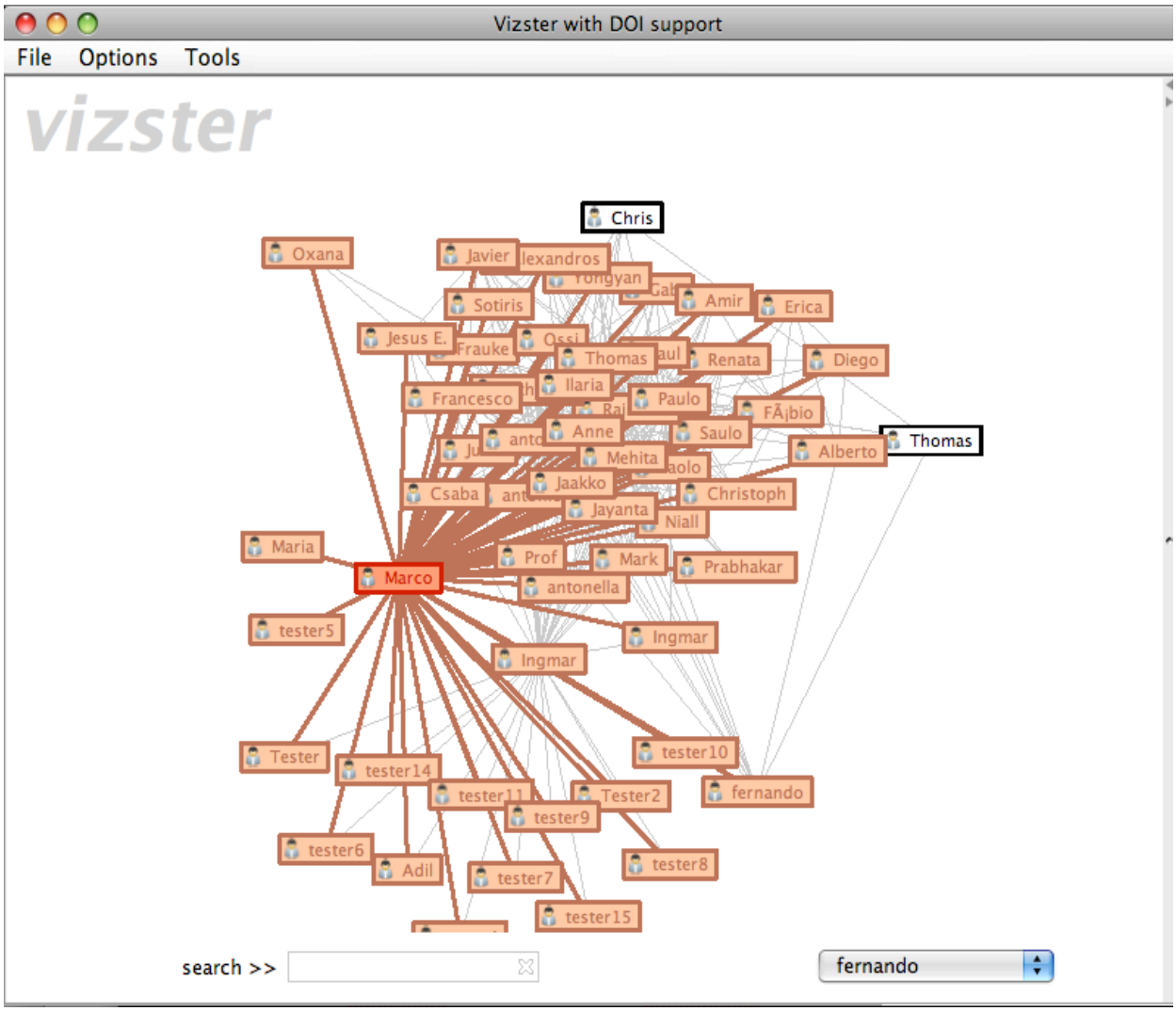


Figure 4.2.3.1: *Guigoh social actors visualised with Vizster*

Next, let's take a look at the network of OPAALS wiki contributors:

1. In order to test the visualisation in OPAALS wiki, move to a wiki category page such as *http://wiki.opaals.eu/CategoryProject/WorkPackage*
2. Again, Wille SNA widget should appear to your browser.

For the visualisation of the OPAALS Wiki, multiple visualisations are available. In order to take a look at a wiki contributor network, you can either visualise it with Vizster ("*vizster"* link) or export

53 Such as http://www.opaals.org.br/ProfileView.do?id=6

54 Note that in Firefox clicking the link will download a JNLP (Java Network Launching Protocol) file. Once this file has been downloaded, you may need to manually open the file from the download manager (accessible from menu at *Tools* > *Downloads*).

the data to Pajek for further processing (*"pajek"* link). For OPAALS wiki page categories, you can also export data representing the evolution of the wiki contributor network *("evolution"* link). This selection launches a data export mechanism that was originally developed to support the analysis of the evolution of wiki contributions (see Colugnati and Carrari Lopes, 2010). In this case, socio-graphs are presented as panels representing the state of the social network in different timeslots instead of static socio-graphs. Data is formatted in a way that it can be imported to Siena and StOCNET[55].

An example visualisation by exporting data to Pajek is provided in Figure 4.2.3.2. For a more aesthetic look, a 2-dimensional Fruchterman-Rheingold layout algorithm was applied to the graph in Pajek. In the figure, wiki pages and wiki users are encoded as nodes, coloured in green and red, correspondingly. Nodes are labelled by their indices in Pajek. Contributions to wiki pages are visualised as arrows pointing from user towards a wiki page.

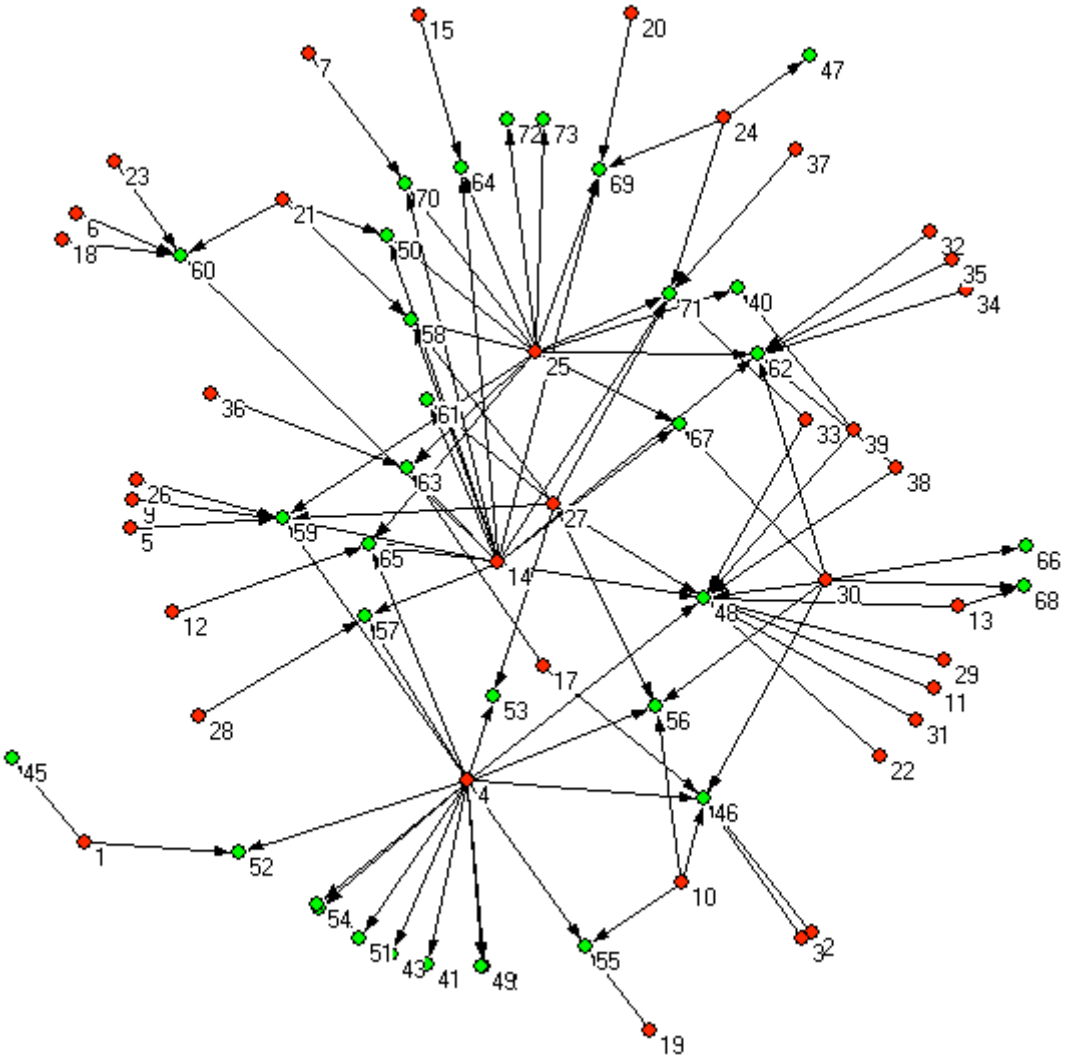


Figure 4.2.3.2. *Contributions related to OPAALS wiki pages in the wiki-category CategoryProject/WorkPackage*

You can also find a Wille SNA Toolkit implementation for Innovation Ecosystems Consortium, which provides a viable future application area for Wille2 SNA Toolkit (see instructions at http://wiki.tut.fi/Wille/SNAToolkit#Testdrive_Wille_SNA_Toolkit). From Wille SNA Toolkit point of view, the next steps are either using the existing components to gain more insight on the

55 http://stat.gamma.rug.nl/siena.html

networks that the toolkit already supports or introducing the Wille SNA Toolkit to new contexts by reusing and recomposing the existing components.

4.2.4. Discussion

Despite our significant efforts during the course of OPAALS, developing easy to use, point-and-click applications for SN visualisation has proven to be a difficult task. Programming skills are still quite often needed to do experimental SNA. Creating a general-purpose crawler operable solely via a GUI, for example, is in most cases impossible in practice.

Wille2 and Wille SNA Toolkit introduce an example of an environment in which new SNA processes can be developed in an evolutionary manner. Existing SNA components range from unfinished scripts that crawl wiki data to create local data instances and case-by-case SNA apps for Guigoh, OPAALS wiki, and Innovation Ecosystems Database to polished, tested and well-documented general-purpose services e.g. capable of producing central SNA figures for the apps (or, more precisely, app developers) to use.

More particularly, Wille serves the SNA work from the following points of view:

1. As in any visualisation domain, the evolution-supporting nature of Wille2 helps in the incremental development of different SN analysis and visualisation solutions. Sometimes rough and sketchy scripts implementing mechanical data-processing routines are enough but, if necessary, they can be developed further into polished apps or services in a straightforward manner.
2. SNA data is sensitive by nature. As Wille is, by default, operated on a local machine, there is no need e.g. to anonymise the data to be analysed as one has to do when using services such as IBM Many Eyes. Even Google Docs and similar third-party services are often regarded as being "public".
3. Delivering the visualisations to the original context of the SNA source data may help the analyst in concentrating in the analysis work instead of solving problems related to data formats and similar issues.
4. The scripting nature of Python complemented with existing Wille components (XSLT processor and others) enable the straightforward development of SNA data crawlers and pre-processors. The end-user knowledge requirements are, of course, increased but it is in many cases impossible to implement all the different needed features with WYSIWYG tools.
5. Python supports nicely SNA development but, importantly, also existing components built with other technologies than Python can be integrated to the toolkit. `numpy` and other Python packages present functionalities that are of use when developing algorithms based on e.g. matrix algebra.
6. Exporting the SNA data to different *de facto* analysis tools is straightforward with Wille SNA Toolkit.
7. Data crawling processes are slow, thus reducing the benefits of context-sensitive crawling. The approach taken in Wille SNA Toolkit is, however, useful also e.g. when delivering previously crawled data to the origin context of the data.

Future work includes e.g. utilising the different SNA metrics in adding details to different visualisations. We have already tailored Vizster in a way that the sociograph nodes can be inflated on basis of different metrics.

More generally, we are interested in testing out the ways that the context-sensitive visualisation change the visual SN analysis. Our working hypothesis is that bringing the SN analysis and visualisation tools to the context where the social actors in the data appear can, at best, help investigators in gaining an understanding on the regularities implied in different social networks in

a more straightforward way that they currently are able to do.

Our objective is to use Wille to complement the existing SN analysis and visualisation tools and, further, enable the application of some of the expressive procedures that SNA introduces to the development of information visualisation solutions in general. We are going to continue developing Wille SNA Toolkit e.g. related to data-driven visualisation oriented innovation ecosystem research done in co-operation with Innovation Ecosystems Consortium and the research of TUT Hypermedia Laboratory Social Network Analysis team.

4.3. Databox View

A common challenge in visualisation is seeing the "big picture" of things. Intuitively this means organising data so that the end-users can perceive global patterns etc. on a glance. We shall next introduce a general-purpose view application of the Wille visualisation system family that allows visually browsing and filtering a large volume of data (tenths of thousands of items). We call the application the (Wille) Databox View.

4.3.1. Introduction

The objective of visualising a large amount of data implies two challenges. First, data needs to be properly accessed from multiple sources and pre-processed appropriately. Second, the viewer application needs to manage a large volume of data efficiently, for a smooth user experience.

So far much of the visualisation system development has been devoted to developing a concrete framework and process model for data modelling, pipeline data processing, and formatting data to existing or configurable viewer applications, with the aim of reusing existing viewer applications as much as possible. From a very practical application point of view, we indeed have reached a point where creating visualisations is not constrained by the repertoire of tools and methods, but the available data and consistent policies of creating and modifying content. While the processes of authoring and data processing go iteratively hand in hand, the topic of authoring policies, however, is strictly speaking out of scope in visualisation system design.

In brief, we have considered the abstract architecture of the visualisation system, peer-to-peer data modelling, and semantic interpretation of data (Nykänen, Salonen, Haapaniemi & Huhtamäki 2008; Nykänen, 2009a; Nykänen, 2009b). One important piece of the jigsaw of visualisation systems is nevertheless still missing. The experiments and pre-studies have quite clearly indicated that the repertoire of open-source viewer applications does not seem to include tools for visually browsing large amounts of pre-processed data and other generated content (that can be delivered in a P2P network). For this purpose, we have implemented a general-purpose viewer application using Java 1.6, called the Databox view, to complete the Wille framework with a simple viewer application capable of interactively viewing reasonably large datasets projected on a 2D canvas.

4.3.2. Implementation

Databox View is a common-purpose visualisation view application that can interactively represent large volumes of tabular data, conceptualised as items on a two-dimensional canvas. Each item is represented with a small rectangular icon and title, and associated with additional properties (suitable for e.g. filtering). Item data can be further decorated using simple vector graphics.

Databox View enables an efficient creation of visualisation applications. A typical application consists of two components: 1) A data processing pipeline yielding a simple item table file and an associated graphics file; and 2) a Databox View project file. Opening the project file launches a Databox View application that allows visually browsing the data.

Note that the item and the graphics data typically depend on a particular data processing pipeline.

Thus, the data content and the visual appearance of Databox View may change upon applications. A more complete explanation of the Databox View application and the actual runnable application are accessible at the Wille wiki.

The idea of encapsulating the last two steps of a data processing/visualising pipeline (when needed) into a single application supports (P2P) plug-and-play use cases. In brief, since the data can be transmitted alongside with the runnable Databox View JAR-file, a complete visualisation application can be transmitted and distributed in a network with relatively modest assumptions. When the data sources experience changes, the Databox View can refresh the view by reading the (local) files, again without assuming much of the network implementation. In brief, these files could be cached using the Wille framework within the local node, again encapsulating much of the activities from the View component.

4.3.3. Case: OKS from 50,000 Feet

A concrete use case for the Databox View application is introduced in the Deliverable 6.11 - View to the evolution of the OKS from 50,000 Feet. In brief, the purpose of this application is to see the big picture of the OPAALS OKS, focusing onto the currently available data, with the aim on visualising aspects of evolution in the OKS. We shall next outline the visualisation aspect of this work, to illustrate the Databox View application.

The data for the application are requested from several sources, pre-processed, suitably transformed, and finally normalised into a single database representation suitable for interactive viewing. In an ideal setting this process would be fully automated and the visualisation tools indeed support this. However, in practice, this study includes data that effectively required manual processing, due to inconsistencies in archive locations, data structures, and/or semantics. As a consequence, some manual efforts were required to compile and clean the data.

In addition to actual sample data, visual mappings and decorations are also needed to support visualisation. In our case this requires specifying a visual 2D visualisation model on which the items are projected. The item projection was kept as simple as possible: The X coordinate was computed by projecting items on a timeline, and the Y coordinate was computed by sorting items alphanumerically by title, starting from the top. The items were decorated by a simple calendar grid, with a specific slot for items with unknown dates. In general, the applicable projection methods obviously strongly depend on the visualisation metaphor and the available item properties. Also, visualising e.g. taxonomy or rule-based knowledge would require introducing a more complex visualisation model and an associated intuitive metaphor.

In brief, the data included in the visualisation application includes community wiki pages, community-wide emails (the "Opaals-All" mailing list), reported project deliverables, and reported publications. However, because of the limited availability and the practical challenge in retrieving the data, all conceivable project data could not be included. In particular, the data includes final versions of wiki articles between June 2006 and March 2010, emails between June 2006 and March 2010, deliverables between 2006 and 2009, and finally publications between 2006 and 2009. However, for practical purposes, the data is by no means complete.

After processing, the data includes total 1598 items with 13 properties. When represented as an uncompressed file (including markup), this comprises about 6.0 megabytes of textual data. (A default zip compression reduces the size to roughly 1.6 megabytes.)

Figure 4.3.3.1 presents a global view to the sample data. In brief, items are associated with coloured icons and text labels on a 2D plane. The colour codes indicate the type of the items, including OPAALS Wiki articles (Jun 2006-Mar 2010; blue), Deliverables (2006-2009; maroon), Publications (2006-2009; green), and Opaals-All Emails (June 2006-Mar 2010; yellow). Other item properties may be examined by viewing item information.

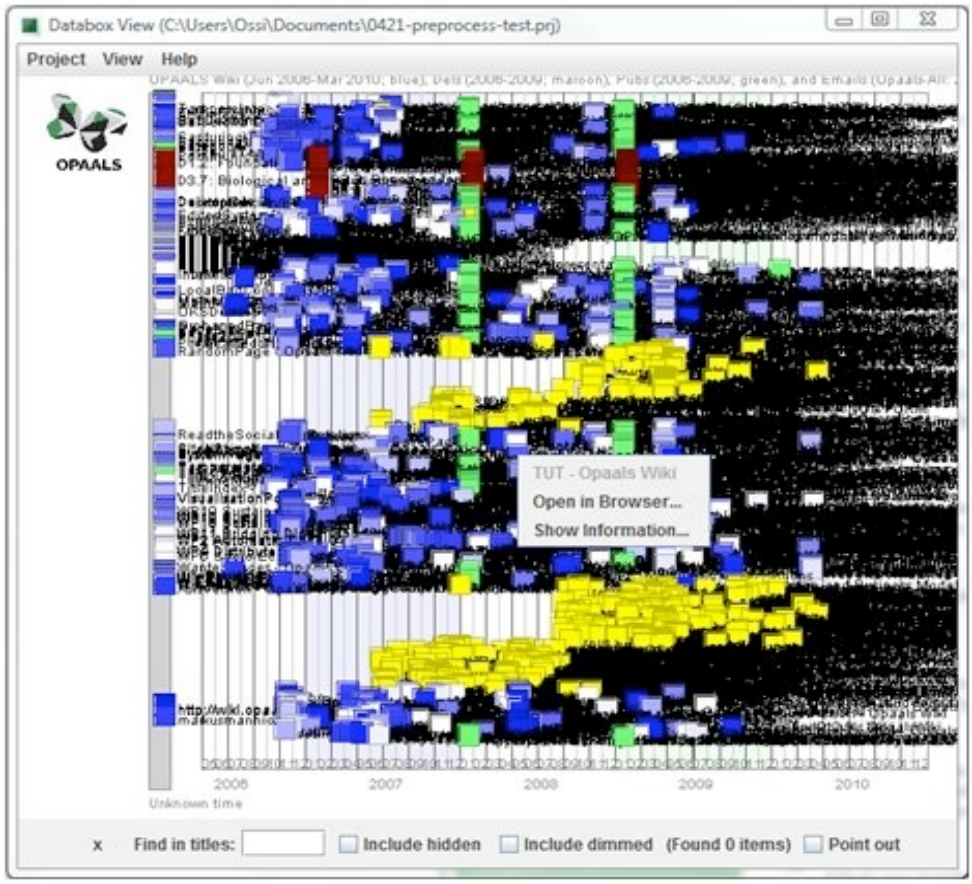


Figure 4.3.3.1. *Databox View to the OPAALS OKS*

With the help of the interactive application, users can navigate within the item data by zooming and panning (Figure 4.3.3.2), filter and highlight data, and open particular items for further examination in the desktop Web browser.

Figure 4.3.3.2. *A close-up view for some early 2008 data*

The fact that the visualisation may represent also data content, and not only metadata, provides a nice setting for non-trivial explorative analysis. Figure 4.3.3.3 (above) illustrates the usage of specific terms in emails between mid 2007 and early 2010 (n=480), highlighting specific aspects of the data.

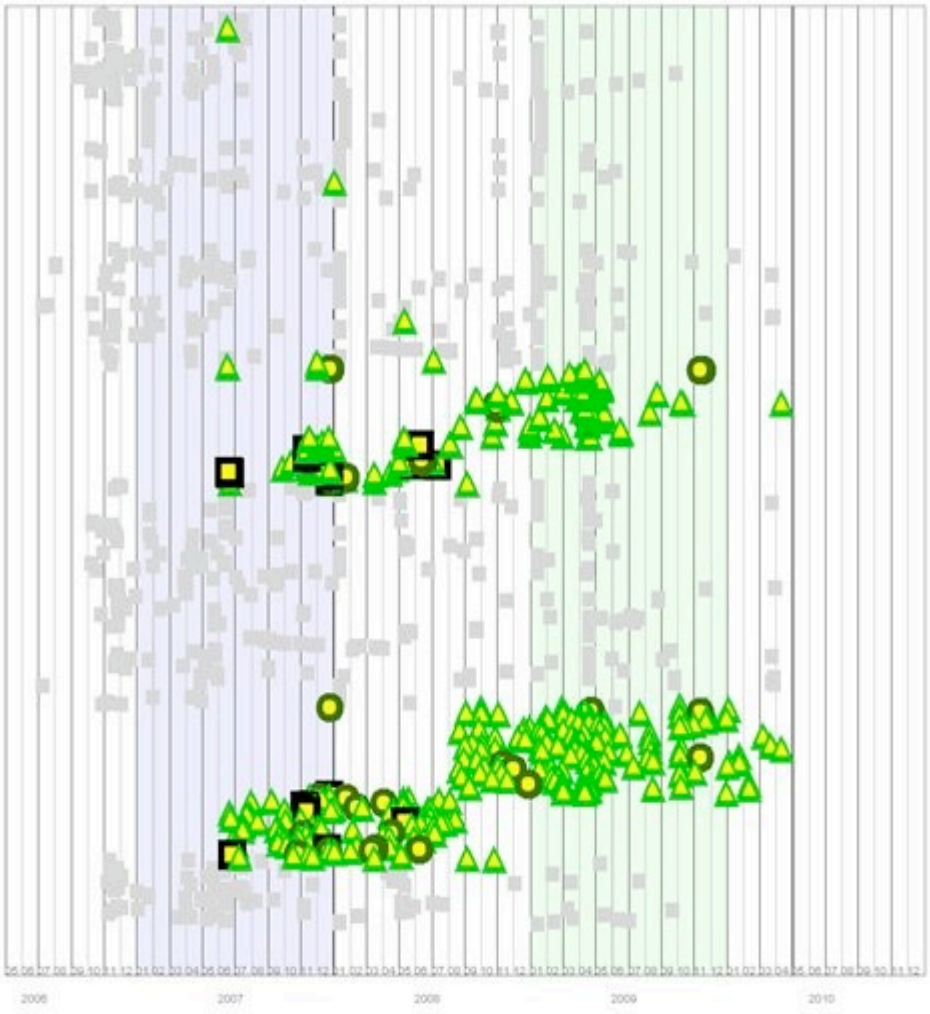


Figure 4.3.3.3. *A trend of terminology in emails*

The view is computed as follows: First, all item text labels are hidden for clarity and all items besides emails (by type property) are dimmed (presented as small light grey boxes). The email items are then highlighted and depicted as light green triangles. The email items whose title or actual content includes the word "[Ee]cosystem" (the *title* or *pageData* property match with the regular expression) are emphasised and depicted as darker ovals. Finally, email items whose content in addition includes the word "[Aa]pplication" are highlighted as boxes with thick black border.

Interpretation of this view is outside the scope of our discussion here. (The example nicely highlights, e.g., the issues of evolution of activities, evolution of language, non-representative sample, and evolution of data semantics.)

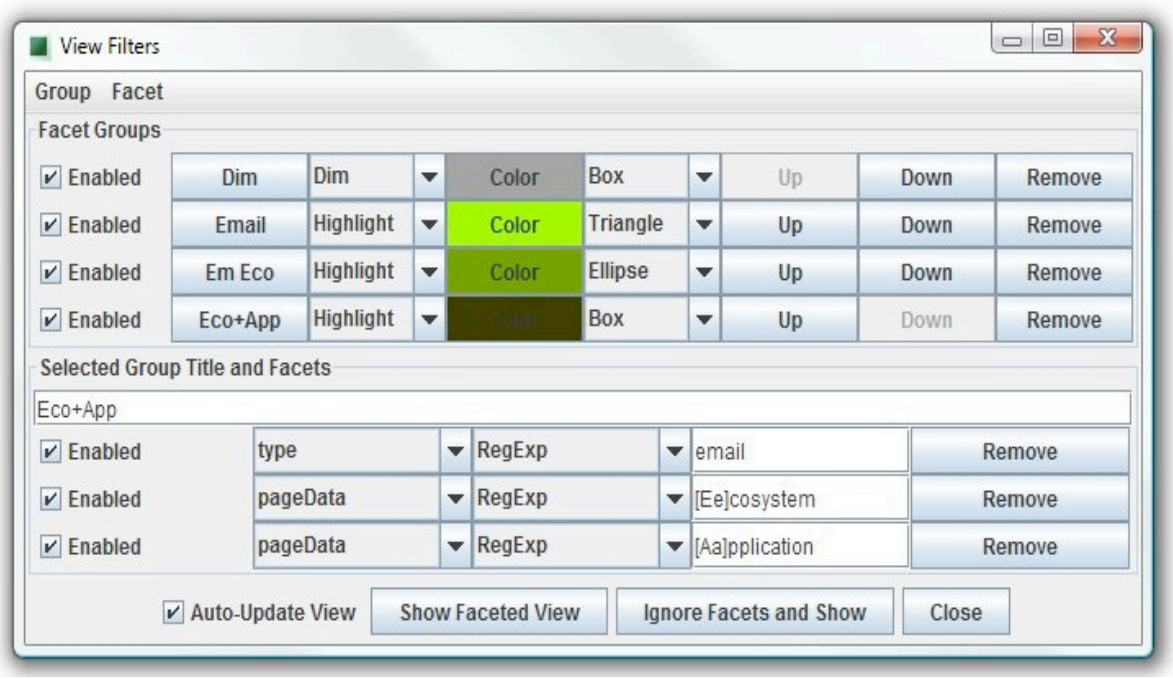


Figure 4.3.3.4. *Databox View filter dialogue*

The ability to visually navigate within the data, filter data, and get a visual frame of reference to the data should help grasping the related and partly tacit processes that underline the visualisation. Figure 4.3.3.4 depicts the Databox View filter manipulation dialogue that was needed for setting up the view discussed above. Nevertheless, it should be emphasised that the availability of data dominates visualisations. The data that is not recorded (e.g. informal discussions), that is not accessible (e.g. group-specific or private data) or that cannot be mechanically interpreted (e.g. free-form research notes) is obviously missing from the visualisations. Unless being careful, this may to lead misinterpretations in analyses etc.

4.3.4. Discussion

The case study highlights the importance of well-managed data processes in visualising knowledge spaces. In brief, since visualisations can only represent available data, the potential of data pre-processing and visualisation is fundamentally determined by the quality and availability of data sources. Further, the objective of interactively visualising large volumes of data implies both challenges for designers and end-users. In particular, the complexity and scalability of algorithms and data structures becomes quickly an issue, and the understandability of views greatly determines the applicability of suitable visual models. It is very easy to brainstorm with complex projection algorithms for item data, but experiments suggest that for practical reasons (GUI simplicity, user training, etc.), simple projections tend to work best in concrete applications. In brief, the key factors of successful visual models are high level of affordance in itemisation and in item projections, intuitive/familiar browsing functionality, and that small changes in the source data imply small changes in the visualisation.

5. Conclusion

In this deliverable, we have presented the final version of the visualisation system and demonstrated it in the context of OPAALS. In brief, this work has provided the following results: rationale, design principles, and a concise application catalogue (this document), specific application examples (attached executables and data), and the Wille visualisation system Web site, for continuous work and further activities.

Since the properties and impact of the Wille visualisation system has already been thoroughly discussed in Deliverable 10.12 (see the structured conclusions section), we only add few notes here and consider the future research of the area.

Looking back in time and considering the research and development process of the visualisation system, it seems evident that two main initial decisions strongly influenced the direction of our research. First, the early engagement with Peer-to-Peer (P2P) infrastructure provided a strict requirement of self-sufficiency in applications. While the OPAALS OKS did not transform into a complete P2P system during the funded OPAALS project phase, this early decision paved road for a robust design that, when required, can easily be "loosened" to match the needs of e.g. “Webised” applications.

Second, the early research and prototypes (Deliverable 10.6) clearly demonstrated the limits of declarative definitions of visualisation systems. In particular, the strict requirement of taking (infrastructure, APIs, schemata, data) evolution into account pointed out the need of introducing scripting to the very core of the visualisation system. This provides the sufficient flavour of agility to applications. To certain extent, one might even say that the evolutionary concepts of the project effectively materialise in terms of acknowledging the challenge of meeting evolutionary requirements rapidly. This points out that the level of abstraction (and thus encapsulation) of the concrete P2P system applications is fundamentally dependent on the level of abstraction (and thus encapsulation) of the underlying concrete P2P system infrastructure. However, in a state-of-the-art research project where production-level maturity cannot be assumed, this merely implies that a certain gap between the abstract concepts and implementations must be accepted.

As a consequence of the above line of thinking, Wille2 design is well-suited to support evolutionary visualisation development. Implementations can easily evolve from (1) rough, sketchy scripts implementing batch processes to (2) browser-operable apps, (3) reusable components to be consumed as Wille services or (4) re-composable widgets for future (5) visualisation dashboards. The visualisation system is easily extendible. Its design allows useful pieces of existing software to be easily wrapped as various Wille component techniques. Further, existing visualisation tools can be served with data, delivered as parts of Wille Apps and made reusable as Wille Widgets. In a way, the toolkit enables high-end visualisation development done in a mash up fashion. Default Wille front-end based on Web technologies allows lightweight, loosely coupled integration between the toolkit and Web applications in where the information visualisation user works. Importantly, Visualisation can be injected to the systems where the data originates from. Further, the use of Web technologies for Wille front-end enables us to demonstrate the implementation of context-sensitive application with the visualisation system. Data on the context of the visualisation use is circulated among different Wille apps and the user is provided with a catalogue of possible visualisations. Moreover, the possibility to use an intelligent probe connecting Wille to the user's context presents interesting means for collecting data to be visualised directly from the user's browser.

The research rationale and decisions have also introduced certain restrictions for the visualisation system applicability. The processor and encapsulated service-oriented design stance of visualisations clearly emphasises pull kind of data processing. This means that while complex reactive end-user applications can be implemented, the certain agnostic attitude towards the

underlying network infrastructure suggests working with cached data. In turn this implies that the applications' perspective to the data is typically somewhat delayed and/or partial. Following this line of thinking, one might also question the need for automated data collection and processing as it always introduces some overhead unless the task is often repeated. In some applications, the most viable method of preprocessing data is simply to do it manually.

Also, the quite clear distinction between visualisation and authoring tools that originates from the evolutionary paradigm of fostering the use of various tools and data formats emphasises the view aspect of the visualisations. Finally, while scripting clearly empowers application designers, it may give rise to the barrier of end-users configuring their applications, assuming no other explicit configuration mechanism is provided. However, the P2P nature suggests executing many of the applications locally, which in itself may suggest a certain "a bit experienced" visualisation end-user profile.

This work continues in the current and new domains. In itself, the catalogue approach demonstrates three directions of future research and development: Development of the core visualisation system, migrating into a specific problem domain and integrating new modules to the system (such as social network analysis and context-sensitive visualisations), and developing more sophisticated (general-purpose) view applications and process models. In addition to the specific research questions, the partners involved in the work have organised several new research initiatives, which from the visualisation point of view aim also to take the research of the actual interface concept a step further.

6. Acknowledgements

This work has been kindly supported by the partners involved in WP5, WP6, WP9 and WP10. This particular Deliverable is a result of collaborative work of the OPAALS project team at TUT. The authors wish to explicitly thank their TUT colleagues Thumas Miilumäki, Jarno Marttila and Ismo Karjalainen for their contributions.

7. References

Colugnati, F., & Carrari Lopes, L. (2010). Analysing collaboration in OPAALS'wiki: A comparative study among collaboration networks. In Proceedings of 3rd OPAALS Conference on Digital Ecosystems, March 22-23, Aracaju, Brazil. Berlin/Heidelberg: Springer.

Danis, C. M., Viegas, F. B., Wattenberg, M., & Kriss, J. (2008). Your place or mine?: visualization as a community component. In Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems (pp. 275-284). Florence, Italy: ACM.

Freeman, L. C. (2000). Visualizing Social Networks. Journal of Social Structure, 1(1), [np].

Freeman, L. C. (2009). Methods of Social Network Visualization. In R. A. Meyers (ed.) Encyclopedia of Complexity and Systems Science. Berlin: Springer.

Haapaniemi, M., Huhtamäki, J., Kortemaa, A., Mannio, M., Nykänen, O. & Salonen, J. (Eds.). (2007). Deliverable D10.6: A proof-of-concept implementation of a visualisation client. OPAALS project deliverable, Phase 1.

Huhtamäki, J. (2007). Community visualisations in Open Knowledge Space: Uncovering rabbit holes in a digital ecosystem. In Proceedings of 1st OPAALS workshop, November 26-27, 2007, Rome, Italy.

Huhtamäki, J., Nykänen, O., & Salonen, J. (2009a). Deliverable 10.12: Component-based visualisation system with collaborative OKS core scenarios. OPAALS project deliverable, Phase 2.

Huhtamäki, J., Nykänen, O., & Salonen, J. (2009b). Catalysing the Development of a Conference Workspace. In HCI International 2009 Conference Proceedings, July 19-24, 2009, San Diego. Berlin/Heidelberg: Springer.

Huhtamäki, J., Nykänen, O., & Salonen, J. (2009c). Deliverable 10.11: Specification of the P2P configuration of the visualisation system. OPAALS project deliverable, Phase 2.

Nykänen, O., Salonen, J., Haapaniemi, M., & Huhtamäki, J. (2008). A Visualisation System for a Peer-to-Peer Information Space. Proceedings of OPAALS 2008, 7-8 October 2008, Tampere, Finland. pp. 76-86.

Nykänen, O. (2009a). Understanding Data via an RRS in RDF/XML. In IADIS International Conference Applied Computing 2009, Rome (Italy), Nov. 19-21, 2009, Vol. I, ISBN 978-972-8924-97-3.

Nykänen, O. (2009b). Semantic Web for Evolutionary Peer-to-Peer Knowledge Space. In Birkenbihl, K., Quesada-Ruiz, E., & Priesca-Balbin, P. (Eds.) Monograph: Universal, Ubiquitous and Intelligent Web, UPGRADE, The European Journal for the Informatics Professional, Vol. X, Issue No. 1, February 2009, ISSN 1684-5285, CEPIS & Novática. Available at http://www.upgrade-cepis.org/issues/2009/1/upgrade-vol-X-1.html

OKS Glossary. (n.d.). Retrieved from http://www.opaals-oks.eu/about-oks/oks-glossary.html

O'Leary, D. E., (2008). Wikis: 'From Each According to His Knowledge'. In Computer, pp. 34-41, February, 2008. IEEE Computer Society.

Salonen, J., Huhtamäki, J. (2010). Launching Context-Aware Visualisations. Proceedings of the 3rd International OPAALS Conference on Digital Ecosystems: OPAALS 2010. 22.-23. March 2010, Aracaju, Brazil. To appear.