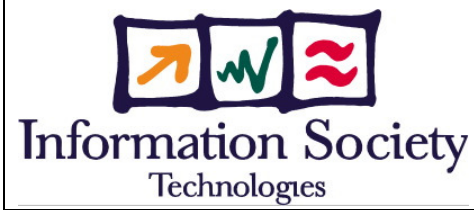
OPAALS PROJECT

Contract n° IST-034824

WP10: Sustainable Community Building

Del10.11 - Specification of the P2P configuration of the visualisation system



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

Contract Number: IST-034824

Project Acronym: OPAALS

Deliverable N°: 10.11

Due date: Month 34

Delivery Date: Month 35

Short Description:
In this deliverable we report and specify a P2P configuration of Wille Visualisation System.

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

Partners contributed: UNIS, UniKassel, LSE

Made available to: OPAALS Consortium

Versioning		
Version	**Date**	**Name, organization**
v1	01/09	TUT internal draft
v2	01/09	Consortium comment draft
v3	04/09	Final version

Quality check

Internal Reviewers:

Csaba A. Szabó (CN), TV Prabhakar (IITK)

Dependences:

Achievements*	Organisation of a visualisation system to component based, deployable on top of a P2P infrastructure
Work Packages	WP3: visualisation requirements for P2P network models WP5: technical specification for component-based visualisation system WP6: Framework for visualisations in evolutionary P2P knowledge space WP10: P2P framework for OKS visualisation
Partners	IITK, IPTI, UniKassel, LSE, SUAS, UNIS, TechIDEAS, WIT
Domains	Computer Science: Technical framework Natural Science: Informal requirements Social Science: Informal requirements
Targets	• D10.12: Component-based visualisation system with collaborative OKS core scenarios • P2P infrastructure developers • Partners and third parties implementing P2P visualisation systems • Computer Science community
Publications*	Haapaniemi, M. 2008. Designing a Peer-to-Peer Visualisation System: a Prototype and a Specification. Master of Science Thesis. Tampere University of Technology. Nykänen, O. Semantic Web for Evolutionary Peer-to-Peer Knowledge Space. 2009. Novatica Monograph. To appear. Nykänen, O. 2007. Interpretation Logics. Proceedings of the 1st OPAALS conference, 26-27 November 2007, Rome, Italy. Nykänen, O., Salonen, J., Haapaniemi, M., & Huhtamäki, J. 2008a. 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., Salonen, J., Huhtamäki, J., & Haapaniemi, M. 2008b. OKS Data Model, version 1.01. A milestone specification for the OPAALS PROJECT (Contract number IST-034824), WP10: Sustainable Research Community Building in the Open Knowledge Space. Contribution to the Milestone M10.10: OKS Data Model (M24), 1 July 2008 (29 pages).

PhD Students*	Jukka Huhtamäki: Researcher
Outstanding features*	State of the art P2P architecture sufficient for international publication
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 the first phase of Network of Excellent project OPAALS, a proof-of-concept visualisation system for Open Knowledge Space (OKS) was designed (see Haapaniemi, M., Huhtamäki, J., Kortemaa, A., Mannio, M., Nykänen, O. & Salonen, J. 2007., Nykänen, O., Mannio, M., Huhtamäki, J. & Salonen, J. 2007), as part of Work Package 10. *A visualisation system* is an application, that can be used to produce information visualisations. Rather than duplicating the functionality of authoring systems, we consider visualisations as complementary part of typical modify/view use cases.

In phase 1, the system was built by integrating open source software into a pipelined software architecture, for visualising data independent of the data domain. The resulting proof-of-concept implementation was based on centralised design: all components must be installed and executed within the same system (Haapaniemi et al. 2007). A proof-of-concept visualisation system, hereafter referred to as *Wille Phase I,* was created.

In the second phase of the OPAALS project, design for making the visualisation system run in a peer-to-peer (P2P) environment, was considered. In this deliverable we will report and specify a P2P configuration of Wille Visualisation System. Wille Visualisation system's design as a component-based visualisation system, with collaborative OKS core scenarios, will be reported in another deliverable, D10.12. A component based, P2P-configurable visualisation system, *Wille Phase II*, is to be defined by the combination of these two specifications.

Two high-level objectives for the specification in this deliverable are 1) to describe a new component-based architecture and 2) to define how to apply a P2P networking architecture to the system. Reflecting on the relationship of Wille and the OKS throughout the work is considered to be important. The required knowledge for the specification is gathered by studying existing P2P applications and prototyping the new system.

Due to the assumed dependency to the underlying P2P infrastructure, detailed identity, security, and trust considerations are out of scope of this deliverable. Note also that the implementation details of the visualisation system are not completely specified in this document. For instance, the resource consumption of the visualisation system may need to be controlled, e.g. via the use of restricted access to processing power, file system, and network access. Technically, this is possible to achieve, e.g. using Java sandboxing capabilities.

The rest of this deliverable is organised as follows:

Chapter 2 gives the background and a short overview of our whole work. Especially relationships to OKS and different P2P architectures in the context of OPAALS are discussed.

Chapter 3 defines an abstract architecture for the P2P configuration of the Wille Visualisation System. This chapter gives the understanding of the roles and responsibilities of different subsystems in Wille Phase II's P2P configuration.

Chapter 4 presents an informative case study that shows how the specified architecture works in a visualisation scenario. The case identifies different actors and the flows of information between them, providing an exemplary overview of the architecture in practise.

In *Chapter 5* we conclude our work, and discuss its results.

Much of the work for this deliverable has already been reported in form of two publications (See Haapaniemi 2008 & Nykänen, O., Salonen, J., Haapaniemi, M., & Huhtamäki, J. 2008a). For a reader interested in more verbose descriptions of some aspects of our work, we encourage to obtain these two publications.

2. Background

Defining Open Knowledge Space (OKS) has been an effort going on in parallel to the construction of the peer-to-peer configuration of the Wille Visualisation System in Work Package 10. When we started prototyping our visualisation system, a preliminary architecture for the underlying P2P network, had already been published (see Razavi, A., Moschoyiannis, S., Krause, P. 2007a), along with a report on formal analysis of autopoietic P2P network and predictions of its performance (see Razavi, A., Moschoyiannis, S., Krause, P. 2007b).

Models for distributed accountability, identity and trust had also been investigated (See Noguera 2007, Ion, M., Telesca, L., McGibney, J., Botvich, D. 2007 & Malone 2007). However, implementation and integration of these efforts into a conforming P2P system, are tasks still under work. In order to claim some characteristics of OKS and its data architecture, an OKS data model was published as a milestone specification (See Nykänen, O., Salonen, J., Huhtamäki, J., & Haapaniemi, M. 2008b).

In practice, this resulted in two parallel tasks in OPAALS project: one in which the underlying OKS peer-to-peer platform was being defined elsewhere (especially in tasks of Work packages 3, 4 and 7), and one in which we defined a P2P configuration for our visualisation system. The whole process could be described as reflexive, recursive and self-reinforcing; considerations from OKS P2P definition influenced visualisation system design, and vice versa. Keeping this in mind, it was found out practical to apply an iterative development process. The process resulted in creation of three P2P prototypes.

2.1. P2P Prototypes

In the beginning of P2P prototyping, we studied existing P2P systems, to gather required knowledge and experiment with early prototypes. Two short case studies were done about Chord Project and Gnutella. We also prototyped Sironta, an example of an OPAALS OKS desktop application (Haapaniemi 2008, pp. 28-29). After studying existing P2P systems, a decision was made to gather knowledge by building a simple system fulfilling the requirements. From these grounds, the first two prototypes were created: initially an RMI-based prototype, followed by an HTTP/REST-based prototype. (Haapaniemi 2008, p. 30). The second P2P prototype was used in development of several sample visualisations. Based on visualisation development experiences, a third prototype was created by further improving the second prototype.

2.1.1. First P2P prototype

The first P2P prototype was created as an RMI (Java Remote Method Invocation) application. It demonstrates the basic ideas of distributing Wille Visualisation Environment. Instead of a single centralised component, the prototype allowed distribution of pipeline components as services into distinct nodes. These services were invoked from a visualisation client, effectively composing services from various sources into a virtually pipeline-based visualisation. (Nykänen et al. 2008a, pp. 30-33).

The first P2P prototype demonstrated some general ideas about making Wille distributed, but further development was found out to be necessary.

2.1.2. Second P2P prototype

In the second P2P prototype we further refined the design. An important addition was support for services execution based on HTTP (HyperText Transfer Protocol) (Nykänen et al. 2008a). HTTP, unlike RMI, is more technology neutral, making it a better choice as interfacing technology for OPAALS.

We also found necessary to decouple services from the P2P layer. Important motivation for this is the ability to exploit and integrate existing Internet-services. This was found out to be important especially in grass-roots visualisation development (See e.g. Nykänen et al. 2008a). Although HTTP was used as the communication protocol between nodes in the second P2P prototype, the system was designed to support any P2P-specific protocol as well. With some additional implementation effort, a component supporting any other protocol, such as Sironta's XMPP-based protocol, could be made available. (Nykänen et al. 2008a, p. 34).

2.1.3. Third P2P prototype

Development of P2P visualisation services was found out to be somewhat difficult in the second P2P prototype. Services were implemented in Java and adding a new services required rebuilding the whole system from source code. In the third prototype, we completely decoupled service implementations from service platform, and created a configuration file based system for connecting the services to the platform. Configuration file contains all necessary information to wrap a component as a service to the service platform. An important benefit for this change was that it was now possible to write services in a technology of choice and simply integrate them to the Java-based Node Platform. Services can now be more easily re-distributed, e.g. as ZIP archives.

The separation of actual visualisations and the services they use was also further emphasized. To provide the visualisation clients with pipeline processor like abilities, we created a small Python client library. Motivation for choosing a scripting language included ability to reuse readily available features of the language, most importantly variables, loops and native libraries for file I/O and HTTP support. With this client library, the user of the third prototype can logically create pipelines without the limitation of having to fix the pipeline to a static structure of services.

Several experimental visualisations and visualisation pipelines were created. During development of the visualisations, we further developed both the Python client and the platform. (See Nykänen et al. 2008a, Section 4.2).

2.2. Discussion

In the three-phase prototyping process, we ended up with a combination of Java based Wille Node Platform for service publishing and Python based client for service invocation and visualisation client development.

Java provided us with potential to easily integrate to Java based P2P systems such as Sironta, while a full-fledged scripting language, such as Python, enabled us to efficiently deal with the grassroots challenges of visualisation development (see Nykänen et al. 2008a, ch 4.2).

Ideally, OKS P2P infrastructure would implement a Service Platform. However, in order to claim a P2P visualisation system, to support collaborative visualisations development and to integrate existing web APIs and services, implementing a lightweight service platform as a component of Wille Phase 2, was found out to be necessary.

From visualisation development perspective, the splitting of platform and visualisation applications, however, also caused some issues. For a developer, it is inconvenient to develop a visualisation application in Python while forcing the use of a Java-based platform for service publication. This also leads to performance issues: service invocations must be circulated via HTTP even when service platform is available locally. When visualisation is offered via a web API (Application Programming Interface), the system also requires to run two web servers locally in parallel. While we acknowledge the need for a technology-agnostic visualisation system, from visualisation developer's point of view, it would make sense to create a completely Python based platform as a lightweight solution.

3. Abstract Specification

Based on prototyping and experiences from Phase 2 work, we present an abstract specification of Wille Visualisation Environment's peer-to-peer configuration.

On a high level, Wille visualisation system is divided into two main parts: *a visualisation client* and *a service platform*. In this chapter the abstract architectures of Wille client and Wille service platform are specified. Implementations of these applications can be built based on these specifications. We also discuss how the visualisation client and service platform would utilise different P2P services, from an underlying P2P network. This chapter is partly adapted from a research paper published in OPAALS 2008 conference (Nykänen et al. 2008a).

3.1. Abstract Architecture

Let us first consider the architecture of a stand-alone Wille client. A *stand-alone Wille client* is an application that can be used to build visualisations using Wille Visualisation Environment. From end-user's point of view, *visualisation applications* can be built either using a stand-alone Wille client or a visualisation client that has been tailored to suit a specific visualisation task.

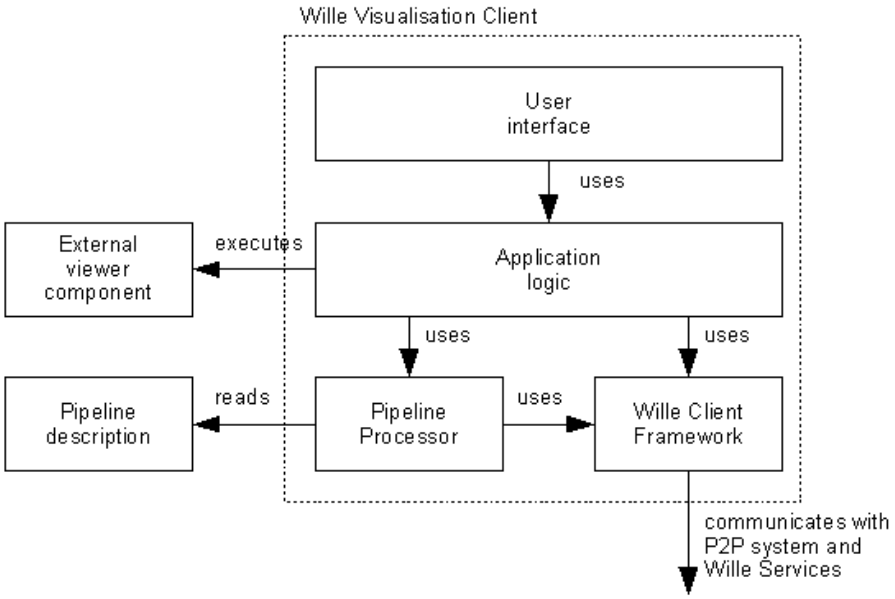


Figure 1: An abstract architecture of a stand-alone Wille client (Haapaniemi 2008)

Architecturally, we can identify the following key parts from both types of Wille clients (illustrated in Figure 1):

- *Wille client framework*,
- *User interface*,
- *Application logic* and
- *Pipeline processor*

Wille client framework is a very important part of the visualisation client handling all communication with Wille services, optionally available from a P2P network. In effect, it handles all communication going outward from the local system. The framework abstracts the actual requests to services, meaning that it hides the communication protocol from the rest of the client.

Internally it can use any communication protocols, for example HTTP or RMI, in communication with Wille services. It is possible to extend the framework to support communication with an underlying P2P network.

Every application needs a *user interface* (UI) and in the case of a visualisation client, the user interface can be, e.g., a command-line interface. When the architecture of the client is planned so that the user interface is a separate component (as Figure 1 shows), changing it to a graphical one is possible, if desired. As a software component, the UI communicates with application logic to execute pipelines and individual components based on user's commands.

It may not always be practical to use a single, general purpose user interface. Since different visualisation tasks may pose very different requirements to the application design, it may be motivated to design specialised visualisation clients for different visualisation tasks. This issue will be further elaborated in Deliverable D10.12 (See Huhtamäki, J., Nykänen, O., Salonen, J. 2009).

Application logic is a software component that manages the overall execution of the client application. Application logic can send requests to individual services using Wille client framework or it can execute entire pipelines using the pipeline processor. It also manages all viewer components found in the local system.

Pipeline processor is a component which is capable of reading pipeline definitions and, based on the instructions in the definitions, performing requests to Wille services using Wille client framework. Pipeline processor does not have to be an explicit component of a visualisation client. Therefore pipeline processor may be considered as a logical component that manages composition and execution of visualisation services. Pipelines processor does not need to communicate with services or the P2P network directly, as that is the task of Wille client framework. Comparing to Wille Phase I, a visualisation pipeline is now regarded as a composition of distributed Wille services, when before it was strictly a composition of local visualisation components.

Let us next consider the second main part of the architecture, *Wille service platform*. *Wille application server* is a central component in a *Wille' service platform* as it is used to host and provide services to a network. Creating a Wille service always requires creating a *Wille* service wrapper. During the start-up of Wille service platform, the application server checks what *wrappers* are installed and makes them available in the network. In principle, the application server may support any protocols that are suitable for the task. For example, it can be a web server or an RMI registry.

To elaborate the definition of a *Wille' service*, we can state that a Wille service is formed by *a service wrapper* and *external components* together (see Figure 2). The implementation of a service platform always dictates the implementation details of a Wille service wrapper. Wille services can use external components to accomplish their tasks, similarly as in Wille Phase I. Also similar combinations of wrappers and components are possible as in Phase 1. An external component refers to, for example, a third-party visualisation or data processing tool. Integration these tools into Wille is a key principle.

3.2. Implications for OKS P2P Architecture

Let us next consider the presented abstract architecture in the context of OKS and its underlying P2P infrastructure. For purposes of discussing the P2P characteristics of Wille Phase II, it is helpful to consider a simplified P2P architecture. This simplified P2P architecture solely aims at providing a setting for our analysis and specification, not necessarily providing the normative definition of the P2P node architecture that is being developed in OPAALS.

The basic idea is that P2P nodes provide either descriptions of or actual visualisation services from which a *visualisation client* may compile *pipeline*s, effectively acting as a pipeline processor when complex computations are involved. The visualisation framework does not specify how components and data sources are found in the P2P network. Rather, it defines the minimal interface using which visualisation components can accept input and provide output. (Nykänen et al. 2008a, ch. 3.2).

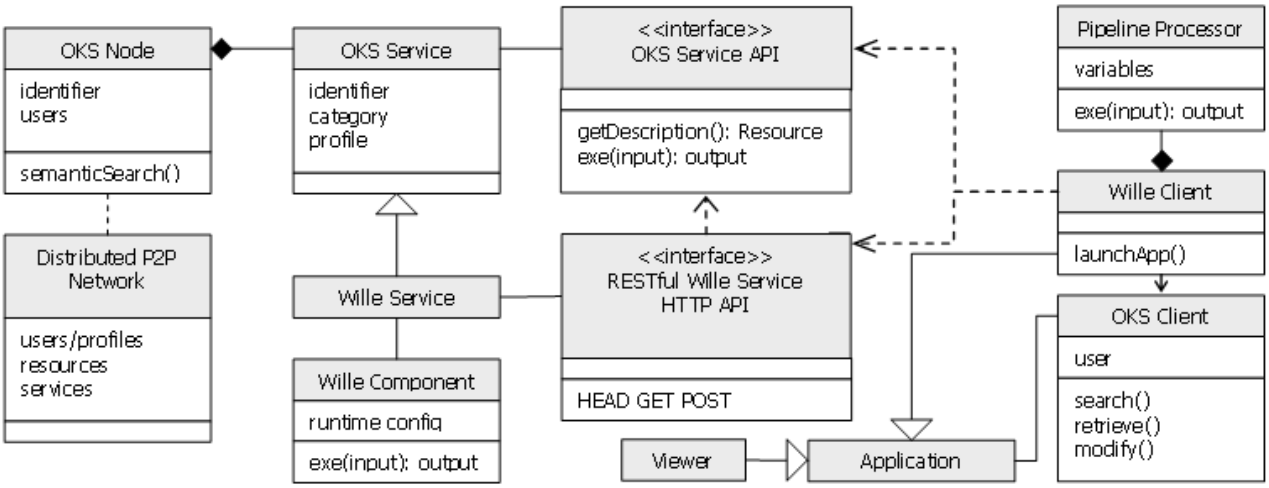


Figure 2: Abstract P2P Wille visualisation system architecture (conceptual perspective) (Nykänen et al. 2008a, p. 5).

Figure 2 describes the abstract P2P Wille visualisation system architecture. The underlying, distributed OKS P2P Network provides services to nodes of this network (OKS Nodes). Nodes offer services via the OKS Service API. Wille (Visualisation) Service is a kind of OKS service that implements the Wille Service REST API, enabling to integrate existing Web applications and services (Nykänen et al. 2008a, p. 5). The architecture depicted in Figure 2 is, by no means, exhaustive. For instance, a potential use case of adding resources via PUT method is not considered.

3.2.1. Service search

The Distributed P2P Network should provide a semantic search to look for Wille Services and data. Assuming that Wille Services and OKS (and other) Repositories provide a REST API, once appropriate service and repository references have been found, a client can directly request services upon location. In a more complex setting, resources and services are both requested from the Distributed P2P Network as a uniform entity, requiring appropriate OKS (P2P) interfacing for both. (Nykänen et al. 2008a)

As such, an underlying P2P network can be used in two possible ways:

1. For resolving service locators, or
2. For requesting actual services.

If the P2P is used only for resolving service locators, integration of existing web services utilising HTTP and REST, would become straightforward. If actual services would be requested based on search, integration of existing services could be done e.g. by implementing adaptor services. While it is crucially important that service components can be accessed via a uniform API, general-purpose services are useless unless efficiently found. To meet the requirement of searching services, the P2P network must provide their descriptions.

In principle, service descriptions might include descriptive properties, classifications, capabilities, and limitations. For instance, a particular component is implemented by some legal entity, it performs some generic or specific task (e.g. performing an Extensible Stylesheet Language (XSL)

transformation or retrieving a company catalogue), it accepts its input in a specific form, using the service is restricted to some stakeholders, one execution costs three cents, it computes in three seconds in average, etc. (Note that the Phase 2 design of our visualisation system does not really aim at implementing a distributed identity or payment services since those are assumed to be included into the P2P infrastructure during later phases.)

Finally in general, four levels of abstractions may be identified when requesting a service component: (Nykänen et al. 2008a)

1. Directly by locator (node identifier, service identifier).
2. Indirectly by service identifier. (When several nodes provide identical services.)
3. Indirectly by service category. (General-purpose services of different implementations.)
4. By service description. (The most general case that nevertheless requires sufficiently detailed descriptions of services and some a priori agreements about their semantics.)

Effectively these levels abstractions can be considered as requirements for the underlying P2P network.

3.2.2. Service execution

After a matching service has been found, a method for its execution should be provided. To exploit possibilities of the P2P system, services should be designed to be interchangeable within a specific group or type of services. Therefore consecutive runs of a pipeline or visualisation client may invoke instances of a service from different nodes. This specifies that service execution must be stateless. Different kinds of data objects can be given to services as parameters, as well as can be received as results of service execution (Nykänen et al. 2008a).

3.2.3. Service composition and P2P pipelines

In a P2P setting, the basic assumption is that a pipeline processor and its components are both run locally, and the data is retrieved from well-defined information repository servers, is no longer valid. Further, also the components participating in the pipeline and data could be distributed.

The novelty of the P2P is that - in principle - component services and data might be requested from the network as a single entity rather than as a network of individual servers. In the most abstract sense, the pipeline processor could be abstracted into a service orchestration specification of acceptable output and tolerable constraints. In this case, the "pipeline" could simply be perceived as behaviour of the service network entity, processing a service request via composition and self-organisation. (Nykänen et al. 2008a, p. 6).

For practical purposes, we simplify this abstract setting with the following assertions:

1. While a pipeline processor could be implemented as a service provided by the P2P network entity, we may assume that it is in fact implemented via a single client requesting individual services from the network.
2. Pipeline is composed from individual service components that have specific locators.
3. The P2P network provides a search mechanism by which service components can be queried with respect to semantic descriptions. Once an appropriate service component is found, it can be directly requested by using its locator, without explicit help from the P2P network. (In practice, the P2P network may however implement a transparent resolution service for achieving this.) From a service-oriented perspective, the P2P network may thus be perceived as a broker provided to service requestors (clients) and providers (nodes).
4. Should the P2P node providing the service drop from the P2P network, it is possible to try to find another, appropriate replacement service from the network. Abstract visualisation services must conform to an appropriate service API. As a practical simplification, visualisation services might be accessible using a REST API implemented over HTTP.

3.2.4. Publishing services

It should be possible to publish new services to any node of the network. This may be desirable for several reasons, including testing and publishing new components locally and changing an existing service and republishing it with some modifications. Clearly publishing services to the user's own node is especially highly desirable, but pushing services to an arbitrary node, may be valuable. Service publishing should clearly require extra priviledges over execution: the user may be required to authenticate and provide required credentials in order to publish a service.

Locally, a user should be able to publish services by providing the local service platform with a new component and making its description available. Note that in many cases this should be sufficient: the P2P network itself should already provide a mechanisms for making the service description available network wide.

Network wide service publication, however, is a requirement for the underlying P2P platform. A well-designed P2P platform would offer several beneficial properties over a single local service, especially ensuring high service availability and good scalability.

4. Informative Case Study

Let us next consider a more informative case study of how Wille Phase 2 visualisation system can be used in a P2P setting.

4.1. Case Description

Let us consider an example scenario where two individual users have installed Wille visualisation environments to their local machines, for their own benefit. User A is using his node, node 1 that has been connected to another node, node 2, administrated by user B. Both users have their own visualisation applications that potentially use partially the same underlying P2P services. Both users may use visualisations that require 3rd party components and services. Especially access to existing web APIs may be required. Note that for simplicity we assume a trivial case in which both users have complete control over their local nodes, but only execution rights to each others nodes.

Let us consider *an example visualisation* where user A uses an experimental *image search visualisation*. The experimental image search visualisation provides user with an explorative view to collection of images, located from a group of data sources including 1) his/her own files, 2) files he has access to in the P2P network and 3) files that are publicly available in the web via a service API, such as via flickr.com API. Effectively, the visualisation provides search the all, both local and global, the user has any access to.

From end-user's point of view, the whole process of using the visualisation can be described as follows:

1. User opens *image search visualisation* and enters an initial search criteria, e.g. some keywords.
2. Matching search results with image thumbnails are displayed.
3. User glances over the results to gain understanding of their importance and may choose to refine given search criteria.
4. New search results are requested and displayed, until user chooses to finish the task

From architectural point of view, the following parts of the system are used as follows:

- User uses the *image search visualisation*, that has been created by configuring a Wille client
- Services are requested from *Wille Application Server(s)* that use an underlying *P2P network* for service search and/or service invocation
- Note that requested services may be 1) local, 2) from an arbitrary node of the network or 3) outside the P2P network, such as available as a third-party provided web API

4.2. Practical Considerations

Unlike in implementations of earlier centralised design of Wille Visualisation environment, P2P configuration no longer assumes that services are executed in a set order that creates a fixed pipeline. Instead, each service should be considered as atomic steps that can be executed any time during use of the visualisation, in any possible order or any number of times. While each service has its unique identity and is primarily provider-dependent, it is assumed that at least a portion of services can be designed to be exchangeable. In our example, we can identify following services that would benefit from being replaceable:

- Image thumbnail generator
- P2P network search service
- Keyword spell-checker

Since these services can be provided as separate components, they can be designed as replaceable. In case a common API would be shared, the visualisation could match them by their type, search for them and select the one with the most optimal characteristics. Hence, these services could be searched indirectly by their service identifiers *or indirectly by their categorisation*.

Some of the services, however, while compliant on the API level, may still be not interchangeable for various reasons. In the example, these services include:

- Local file search service
- Service-provider specific search (such as flickr.com search)

These services can not be replaced with anything else since they are bound to the data they provide. In the spirit of Wille's P2P design, it would be clearly desireable to make as many of the services replaceable as possible. In the case of data-dependent services this would require a P2P file system that would replicate required files from local system into a P2P repository. In the case of for example flickr.com API, such a matter would be out of our hands. Yet, it would be valuable to be able to integrate into systems that are not part of the P2P network envisioned as part of OKS. Hence, it is important that at least some of the services may be invoked *directly, by their locator*.

5. Conclusions and discussion

In this paper, we have presented a peer-to-peer configuration for Wille Visualisation System. In brief, a pipeline-based centralised system was refit into scalable P2P system by separating stateless visualisation components, offered as services, from custom-built visualisation applications.

The new architecture has several important implications. Firstly, with the help of underlying OKS P2P, the system can be scaled from single node system into a full-scale P2P network in order to utilise distribution, e.g. for enhanced reliability and efficiency. Secondly, change from pipeline system into a component based systems provides more flexibility for visualisation development. Visualisations do not need to be compiled by a single static pipeline. Instead, new data may be requested on demand. While pipelined design is not enforced anymore, it may be achieved by service composition. Finally, since services capsulated and are separated from visualisations, virtually any technology can be used to implement both individual services and actual visualisation applications.

The proposed P2P configuration is designed to scale from a single user and node into a full-grown network of P2P nodes. We have established the principles that make this growth possible, but clearly more practical experiences need to be gathered in order to fully understand the systems' applicability. However, at this point the following details about the system's scalability should be noted:

1. In a single-node system, P2P configuration always introduces some overhead, which suggests that visualisations would employ P2P services only when the benefits, such as service replaceability, would surpass the effort required.
2. Design of the underlying P2P network, greatly affects scalability of the visualisation system. It is important to use a P2P network that provides ability to search for services indirectly, e.g. by their type, categorisation, or other described properties, while employing design that overcomes scalability problems associated with semantic search. Structures that are suggested to overcome these issues, include super-peer and dynamic super-peer designs in the P2P network (Nykänen et al. 2008a, p. 3).

Regarding software design, it may be beneficial to integrate the presented Wille client framework as a part of a Wille service platform. By doing this, Wille services would gain access to other Wille services bringing the concepts of a visualisation pipeline and a service closer to each other. Then it would be possible to write a Wille service which makes use of other services, thus behaving like Wille client. To the user of a service, it might not be possible to know the internal construction of the service. From a software engineering perspective, exactly the same Wille client framework, which is used on a Wille client, could be used as an embedded component in Wille service platform.

Several issues in the design remain still to be solved, especially due to lack of an available P2P system that would support the requirements set by Wille's P2P configuration and related data modelling needs (See Nykänen et al. 2008a and Nykänen et al. 2008b). P2P configuration of the OKS has been designed in parallel to our work. Therefore there has been no opportunity to test our work against a stable software specification. Once a stable software specification is delivered, it is possible to validate the P2P aspect of our visualisation system, and accordingly refine the abstract specification.

Acknowledgements

This work is a result of collaborative work of the OPAALS project team at TUT. In particular, we appreciated the past contributions of Matti Haapaniemi, Antti Kortemaa and Markus Mannio. We also value the formal and informal input from the other contributing project partners.

References

Haapaniemi, M. 2008. *Designing a Peer-to-Peer Visualisation System: a Prototype and a Specification*. Master of Science Thesis. Tampere University of Technology.

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

Huhtamäki, J., Nykänen, O., Salonen, J. 2009. *Component-based visualisation system with collaborative OKS core scenarios*. Deliverable D10.12 in Work Package 10 of OPAALS PROJECT (Contract number IST-034824). To be published.

Ion, M., Telesca, L., McGibney, J., Botvich, D. *Trust Model for the DE*. Deliverable D4.3 in Work Package 4 (Distributed Accountability, Identity and Trust) of project OPAALS (Contract number IST-034824).

Malone, P., 2007, *Distributed Accountability Model for an Autopoietic Peer-to-Peer Network*. Deliverable D4.2 in Work Package 4 (Distributed Accountability, Identity and Trust) of project OPAALS (Contract number IST-034824).

Noguera, J., 2007. *Distributed Identity Model for the Digital Ecosystem*. Deliverable D4.1 in Work Package 4 (Distributed Accountability, Identity and Trust) of project OPAALS (Contract number IST-034824).

Nykänen, O., Mannio, M., Huhtamäki, J. & Salonen, J. 2007. *A Socio-Technical Framework for Visualising an Open Knowledge Space*. Proceedings of the International IADIS WWW/Internet 2007 Conference, 5-8 October, Vila Real, Portugal, pp. 137-144.

Nykänen, O., Salonen, J., Haapaniemi, M., & Huhtamäki, J. 2008a. *A Visualisation System for a Peer-to-Peer Information Space*. Proceedings of OPAALS 2008, 7-8 October 2008, Tampere, Finland. pp. 76-86. Available online at http://matriisi.ee.tut.fi/hypermedia/julkaisut/20080807-nykanen-etal-p2pvis.pdf

Nykänen, O., Salonen, J., Huhtamäki, J., & Haapaniemi, M. 2008b. *OKS Data Model, version 1.01*. A milestone specification for the OPAALS PROJECT (Contract number IST-034824), WP10: Sustainable Research Community Building in the Open Knowledge Space. Contribution to the Milestone M10.10: OKS Data Model (M24), 1 July 2008 (29 pages). Available online at http://matriisi.ee.tut.fi/hypermedia/julkaisut/20080701-oks-dm-v1-01.pdf (PDF 0.5 MB)

Razavi, A., Moschoyiannis, S., Krause, P. 2007a. *Preliminary architecture for P2P network focusing on hierarchical virtual super-peers, birth and growth models*. Deliverable D3.1 in Workpackage 3 (Autopoietic P2P networks) of project OPAALS (Contract number IST-034824).

Razavi, A., Moschoyiannis, S., Krause, P. 2007b. *Report on formal analysis of autopoietic P2P network, together with predictions of performance*. Deliverable D3.2 in Workpackage 3 (Autopoietic P2P networks) of project OPAALS (Contract number IST-034824).