# A Visualisation System for a Peer-to-Peer Information Space

Ossi Nykänen (ossi.nykanen@tut.fi), Jaakko Salonen (jaakko.salonen@tut.fi), Matti Haapaniemi (matti.haapaniemi@tut.fi), Jukka Huhtamäki (jukka.huhtamaki@tut.fi)
Tampere University of Technology,
Hypermedia Laboratory
P.O. Box 553, FIN-33101 Tampere, Finland
Tel: +358 3 3115 3544, Fax: +358 3 3115 3549

**Abstract.** Peer-to-peer (P2P) networks provide new opportunities and challenges for distributing and using data and services without a centralised design. An interesting use case for P2P systems is the interplay of information repository and data visualisation applications. In this article, we investigate the potential of distributed information and data processing services and evaluate their applicability in visualisation systems. In short, we generalise our previous work of a pipeline-based client-server visualisation system, decentralising the pipeline components over a service network. We analyse this architecture with respect to the requirements of both P2P information repositories and visualisation applications, in the context of the required use cases. Finally, we demonstrate the technical implementation via pre-studies and discuss P2P knowledge as a complementary topic for P2P visualisations. The work is based on an ongoing research project that will deliver a component-based P2P visualisation system, in a European network of excellence thriving to establish a distributed open knowledge system as a digital ecosystem, based on the ideology of open source communities.

**Keywords:** Visualisation, Peer-to-Peer Networks, Pipeline Processing Systems, Digital Ecosystems

## 1 Introduction

According to several Internet Service Providers, more than half of Internet traffic in 2006 was due to Peer-to-Peer (P2P) applications (Steinmetz & Wehrle, 2006). While this observation does not make client-server (or decentralised) networks obsolete, it highlights a new emerging trend in Internet computing: Dynamic application networks are composed in ad hoc fashion based on the particular needs of grass-roots applications.

In short, Peer-to-Peer networks consist of equal nodes which aim for the shared usage of distributed resources, avoiding central services. Compared to client-server, this change of paradigm in accessing data and services has some implications in application development. Rather than manipulating data upon location, in some cases it can only be referenced upon content.

In this article, we report and analyse a lightweight framework for implementing data-driven visualisations based on a network of heterogeneous Internet data sources that may include P2P component(s). Rather than considering P2P networks as such, we discuss the implicated tasks and challenges for designing and implementing effectively pipeline-based visualisation applications. Besides outlining the architecture of a component-based visualisation system, we illustrate the relationship between the abstract design of component-based processing and simple opportunistic grass-roots implementations of component clients.

Our work relates to information and knowledge visualisation in the context of a large European Network of Excellence Project OPAALS (Open Philosophies for Associative Autopoietic Digital Ecosystems; see http://www.opaals.org/) coordinated by London School of Economics and Political Science. Our work has been supported by the European Commission (IST network of excellence project OPAALS of the sixth framework program, contract number: FP6-034824). The overall objectives of the OPAALS project are: to build a sustainable interdisciplinary research community in the emerging area of Digital Ecosystems (DE, http://www.digital-ecosystems.org/), and to develop an integrated theoretical foundation for DE research. An important part of the project is building the Open Knowledge Space (OKS) to support the related interdisciplinary research and Small to Medium Enterprise (SME) commercial activities in the spirit of open knowledge. For purposes of this article, the OKS could be approximated simply as a repository and a model used to capture and manage the community knowledge, integrating open source and evolutionary principles. It is believed that information visualisation yields the potential of integrating and exploring knowledge, effectively facilitating knowledge socialisation and thus (scientific) innovation.

The timeline of the Network of Excellence as funded project is organised into three phases. During the first phase, we implemented a prototype of a pipeline-based visualisation system called *Wille (Phase 1)* that demonstrates the basic architecture and potential of the OKS approach (Nykänen, Mannio, Huhtamäki, & Salonen, 2007). Now, during the second phase of the project, the underlying technical OKS infrastructure is migrating into Peer-to-Peer. This provides both opportunities and challenges for information management and visualisations. While there is a consensus, e.g., on using

Semantic Web technologies (see, e.g., Herman, 2008) and RESTful design (Richardson & Ruby, 2007) for this task, information management in P2P requires both technical and community effort.

Due to the significance of this application domain, several different approaches to information visualisation exist. Mainstream Web applications such as Yahoo Pipelines (http://pipes.yahoo.com) and IBM Many Eyes (http://many-eyes.com) are in practice doing a good job in demonstrating the concept of collaborative, distributed data-processing and data-driven visualisations. Moreover, there are several visualisation tools and visualisation frameworks including Prefuse (http://prefuse.org) and Processing (http://processing.org) that can be used for creating tailor-made visualisations. In addition, a large number of Web APIs exist, providing data for mashups and visualisations in explicit formats. Examples include Google Maps API (http://code.google.com/apis/maps/) for geographical data and Audioscrobbler (http://www.audioscrobbler.net) for data representing the listening habits of the users of an online radio service Last.fm. Interestingly, major companies have also partly opened infrastructure for developers. For instance, Google App Engine (http://code.google.com/appengine/) and Amazon Web Services (http://aws.amazon.com) can be used to build RESTful applications for processing and visualising data. (About REST, see (Richardson & Ruby, 2007).)

However, from the visualisation designers' point-of-view, there are still some issues in building visualisations. Many 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 different visualisation services in general provide users the possibility to copy or clone existing data, data-processing pipelines, and visualisations within a service, adding new service components may not be possible. The creation and management of both visualisations and data sets in existing applications is often done manually, rather than 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 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, the emerging P2P networks provide the technical framework for achieving this. 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. 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. The main contribution of this article thus lies in presenting a justified rationale and design for a P2P visualisation system, and in illustrating an implementation via experiments.

The rest of this article is organised as follows: Section 2 introduces the key concepts of P2P systems and Section 3 describes our visualisation system design from Client-Server to Peer-to-Peer. Section 4 presents case studies and technical experiments, and finally, Section 5 concludes the article.

## 2 Information Management in Peer-to-Peer Systems

Visualisations are built upon data. In our case, data originates from the OKS that includes data published by the network of OKS (P2P) nodes, and other available Web data. Data integration is based on a common data model, and integration of services is based on a common service interface. In order to address implementing P2P visualisations in a P2P context, we next briefly discuss the characteristics of Peer-to-Peer systems, semantic queries, and the relationship with data modelling and access.

### 2.1 Characteristics of Peer-to-Peer Systems

Intuitively, Peer-to-Peer (P2P) system consists of nodes that collaborate as equal peers, i.e. acting symmetrically as clients and servers when needed, without a central point of control. A refined definition by Steinmetz and Wehrle (2006) asserts that a Peer-to-Peer system is a self-organising system of equal, autonomous entities (peers) which aims for the shared usage of distributed resources in a networked environment avoiding central services.

From the perspective of our work, P2P differs from Client-Server with one crucial account: Data and services are no longer addressed by the address of the server (location), but by the data and services themselves (identity or content). Three well-known mechanisms for storing and retrieving resources in a distributed system include using central server, flooding search, and distributed indexing (Wehrle, Götz, and Rieche, 2006). In practice, distributed indexing may be implemented using Distributed Hash Tables (DHT). Effectively, applying DHT leads into structured P2P systems.

The basic idea behind DHT is establishing a sufficiently large address space and distributing contiguous portions of the address space between the participating nodes. The main function of the DHT system is a lookup function that determines which nodes is responsible for which addresses. The P2P identifiers of resources are typically assigned with a collision-resistant hash function H, based on the hashing of the resource representations. Thus, given a resource D, its P2P identifier H(D) allows referencing it in the P2P system, and perhaps retrieving the resource with the help of the lookup function. In applications, P2P identifiers may further be associated with different kinds of descriptions.

Since nodes may disappear from the network, P2P resources are usually replicated to several nodes and availability is thus achieved via redundancy. Instead of copying actual resources to the P2P system, indirect storage via locator references may also be used. In this case, a request for H(D) does not return D but a locator reference to D, e.g. L(D). Now, using L(D), D may be retrieved with various methods, assuming that the particular node(s) storing it are currently available. Physical data placement is typically done either according to ownership or search strategy (Nejdl and Siberski, 2006).

The strategy of using a simple lookup function assumes that bookkeeping of the P2P identifiers is effectively managed by the applications, i.e. "outside" the P2P lookup system. Thus, while searching for a resource from the P2P system based on its P2P identifier already requires sophisticated query self-organisation, it effectively assumes that the query includes exactly the correct P2P identifier. Coming back to hash tables, this means that either the requestor has itself put the resource into the P2P system and memorised the P2P identifier, or it has explicitly been given the correct identifier by someone else.

Clearly, from semantic applications' (such as visualisation) point of view, simply retrieving resources via lookups is not enough. In visualisation applications, it is important to be able to search resources also indirectly, e.g., by their type, categorisation, or other described properties. Further, some applications need to name things in the P2P network permanently (e.g. metadata about their resources). Since the space of P2P identifiers is typically much smaller than, e.g., the space of Uniform Resource Identifier (URI) names, the P2P network should also provide means for requesting permanent identifiers in a manageable way.

## 2.2 Searching in P2P

To address the very strong assumption of a priori knowledge of P2P identifiers in retrieving resources, the P2P research distinguishes lookups from searches (see, e.g., Stiller and Mischke, 2006). In an ideal case, lookups are not exposed to end-users, but are considered as a part of the P2P infrastructure. As a consequence, P2P systems may be categorised by the expressivity of the supported query language, as key-based systems, keyword-based systems, and schema-based systems (see, e.g., Nejdl and Siberski, 2006). While keyword-based systems extended lookups by simple tagging, schema-based systems in principle allow queries exploiting structured descriptions. While this may seems like a small step, the ability to capture semantics in a machine-understandable fashion opens the door for applications operating on the knowledge level. Common semantics are, however, needed, e.g., for searching visualisation components by service category or other description (see Subsection 3.3).

Schema-based searches in structured P2P networks may be implemented via a structure of super-peers. Instead of flooding queries to all available nodes and aggregating results, searching involves routing queries to super-peers that know which schema elements their peers support. To escape the problem of flooding, some indexing of the schema elements is required. Further, super-peer structure ought to be dynamic, in order to avoid unnecessary specialisation and thus centralisation. In practice, implementing a schema-based P2P query system is rather difficult. A well-documented example is the Edutella framework that provides the Query Exchange Language (QEL) for P2P Semantic Web query interchange (see, e.g., Nejdl et al., 2002; Nejdl, Siberski, Simon, & Tane, 2002; Nejdl & Siberski, 2006). Since the introduction of Edutella, however, other suitable general-purpose Semantic Web query languages have been standardised, including XQuery, and SPARQL (see Herman, 2008).

## 2.3 The OKS Data Model

From the perspective of visualisation systems, the P2P distribution of resources is relevant with two respects: data modelling and subcomponent decentralisation. To support this, we have elsewhere described the OKS Data Model (Nykänen, Salonen, Huhtamäki, & Haapaniemi, 2008). The basic idea is that the OKS network includes nodes that provide P2P core services, including data repository services. Rather than putting resources to the P2P network as such, P2P is likely used for storing references and searching them via semantic descriptions. In other words, the content is owned and provided by the individual nodes while the (some) descriptions are decentralised by the P2P network.

In brief, OKS is composed of resources. An important subclass of resources is descriptions that assert the properties and relationships of (other) resources (including search metadata). According to this design stance, some resources can be identified with URI references. Further, a subset of the resources that can be identified with URI references has (digital) representations. For instance, while a text document might be identified and retrieved by using its URI reference (as a locator), a person might only be identified but not (digitally) retrieved. In short, URI references may be used as a method of uniquely identifying resources, and when appropriate, an enabling mechanism for retrieval. In principle, this design stance allows managing arbitrary (externalised) OKS knowledge via references.

The idea of using P2P network mainly to distribute description resources rather than content resources allows exploiting the Semantic Web technologies to a significant extent. Further, since the OKS may include RESTful applications, this enables also exploiting the existing web services, Web APIs, mashups, etc., when appropriate. This is very important from the perspective of pipeline-based visualisation systems since useful components and normative data sources are readily available and accessible in the Web. To support integration, this also suggests that when specific P2P features are not required, publishing OKS (visualisation) service components according to REST is favourable.

# 3 From Client-Server to Peer-to-Peer

We shall next describe our development of the *Wille Phase 2* visualisation system, transforming the process of making visualisations from Client-Server to Peer-to-Peer. For discussion about information and knowledge visualisation as such, please see (Nykänen et al., 2007; Ware 2004).

## 3.1 Wille Phase 1: Local Pipeline, Local Visualisation Components

The visualisation process of the *Wille Phase 1* visualisation system is depicted in Figure 1 (Nykänen et al., 2007). The source data is typically produced with some authoring tools, e.g. wikis, and a component is created into the visualisation system to import the data ("pull"). Pipeline designer can then use the importer component and other local components to compose a pipeline to visualise the raw data. The pipeline description is then composed with a pipeline application. After the execution of the pipeline the results can be viewed with a local viewer application. For instance, Vizster (http://jheer.org/vizster/) can be used to visualise social networks.

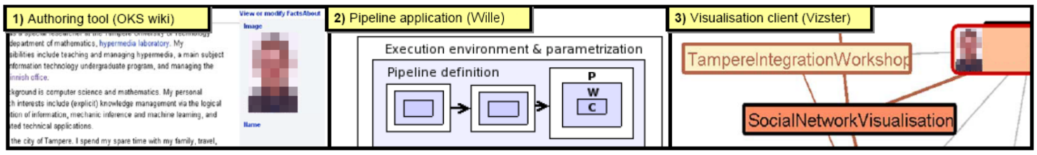
1) Authoring tool (OKS wiki)
View or modify FactsAbout
Image
Name
2) Pipeline application (Wille)
Execution environment & parametrization
Pipeline definition
P
W
C
3) Visualisation client (Vizster)
TampereIntegrationWorkshop
SocialNetworkVisualisation

Figure 1. Conceptual process of *Wille Phase 1* pipeline application

From the perspective of P2P visualisation system, the essential property of *Wille Phase 1* is that the pipeline and the visualisation components are executed locally. Pipeline application reads the pipeline definition and calls pipeline components respectively, and there is no distribution of processing. Further, only the pipeline processor is capable of executing components, thus executing an individual component without a pipeline is not possible. As suspected, the issues of distributed processing and general-purpose component interfaces are addressed in our next version of *Wille* visualisation system.

## 3.2 Wille Phase 2: Towards Distributed P2P Visualisation Components

In the Phase 2 version of *Wille*, the main focus lies in the pipeline components. The idea is that the OKS nodes provide components from which visualisation clients may compile a pipeline, 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.

Thanks to the experience gained during visualisation system development and applications so far, we are able to build the Phase 2 design upon informal findings. For instance, a key concern raised from the design of *Wille Phase 1*, was that users should be able to write visualisations as much as possible by using the technology, tools, and processes they already have adopted, not by migrating into a fixed visualisation platform. Based on this and other similar experiences from phase 1, we assert the following design rationale for the *Wille Phase 2* visualisation system:

1. Visualisation components should be useful as such, without always having to install a specific pipeline processor client to use them. For instance, accessing a simple visualisation component should be possible by writing a shell script or an HTML form.
2. There should be a way to easily create new visualisation components from the existing data processing applications, and offer these as visualisation service components using an appropriate service platform.
3. Besides P2P, the API of visualisation components should follow the conventions of REST when applicable. For instance, it would be nice to integrate existing mashup and feeds to visualisations, without always having to implement complex adapter or wrapper systems.
4. Visualisation components are not (in general) responsible for storing (intermediate/final) results or their computations – this is the responsibility of the visualisation client. For instance, when passing output of a component A to another component B, A does not (in general) have to maintain temporary resources for B, during computations.
5. Visualisation clients are defined by their ability to meaningfully exploit the services of visualisation components, not by their internal making. When writing new visualisation clients, however, accesses to an appropriate, general-purpose visualisation client framework (e.g. programming libraries for common tasks) is typically helpful.
6. Visualisation clients (acting as pipeline processors) typically fall into three main categories: (command-line) scripting (e.g. Python script/program), local application (e.g. with a graphical user interface), and web application (e.g. Ajax or HTML user interface).

From the perspective of visualisation clients and pipelines, visualisation components fall into three categories: Reader, Transformer, and Serialiser (cf. e.g. Cocoon, http://cocoon.apache.org). Readers are concerned in importing data from various data sources for further processing. They typically take care of locating data in origin system, user authentication and data retrieval. While a naïve reader directly access data using resolved locators, P2P readers may utilise the P2P search and lookup functions for the job. Transformers process data into a specific format, perhaps performing arbitrary computations along the way. Finally, serialisers create representations for external applications, e.g. for viewers.

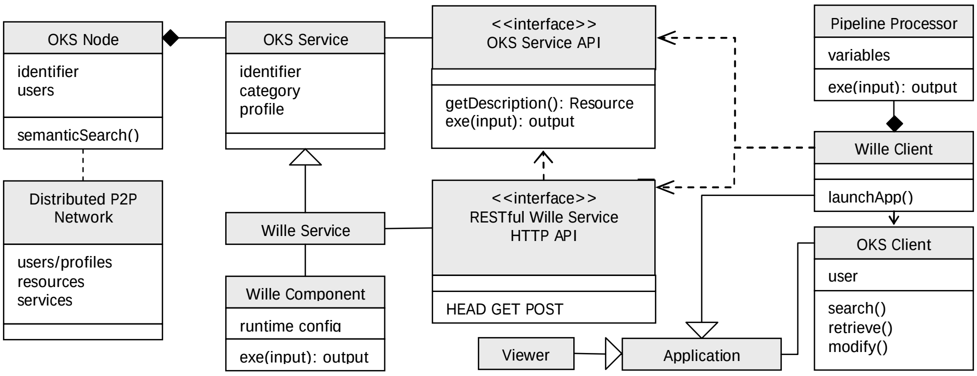

Figure 2. Abstract P2P *Wille* visualisation system architecture (conceptual perspective)

Figure 2 describes the abstract P2P *Wille* visualisation system architecture. The Distributed OKS P2P Network provides services to 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 integrating also existing Web applications. *Wille* Services are implemented via Components that are run in Nodes, and provided to the network via the Service APIs. Finally, *Wille* (Visualisation) Client can access *Wille* Services (and thus component functionality) via the APIs. A non-trivial *Wille* Client (perhaps implicitly) includes a Pipeline Processor that can organise coordinated executions of services into a pipeline. In principle, *Wille* Client is a special kind of Application run by an OKS Client. *Wille* (Visualisation) Client is, in addition, able to launch other applications associated with the OKS Client, e.g., Viewer applications, to show the processed visualisations to the end-user. In principle, a Pipeline Processor may also be published as a *Wille* Service. This requires, however, extending the Processor with *Wille* Service capabilities so that it can respond to, e.g., REST service requests.

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.

Note that as an interesting and highly practical implementation detail, Figure 2 highlights the REST interface of accessing *Wille* Services. In particular, it includes requesting a description or a metadata-only representation of a service using HEAD, and requesting the service component method execution using GET or POST, depending on the size and nature of the arguments, and returning a resource or an archive (typically as a ZIP archive). For this purpose we are thus using an overloaded version of POST, as a Remote Procedure Call (RPC). According to RESTful design (Richardson & Ruby, 2007), POST is commonly used for creating subordinate resources via a kind of "parent" or "factory" resource, or appending data to the resource state. For instance, a weblog service may allow adding subentries via POSTs, creating a new subordinate resource for new entries. However, our design rationale asserts that while a visualisation component may store its results, it is not required to do so. Thus, the only way to achieve the desired (P2P) functionality is to use RPC-style POST, in terms of hybrid REST.

In some cases, however, creating new subordinate resources makes sense. For instance, accessing the functionality of an expensive service (e.g. querying a large archive) is best achieved via a proxy. In a P2P setting, however, the trusted contract of a service should include an explicit note about archiving or publishing component I/O since this may include client-sensitive material.

### 3.3 P2P Pipelines

In a P2P setting, the basic assumption is that a pipeline processor and its components are all 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 a network of individual servers. In the most abstract sense, the pipeline processor itself 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.

For purposes of this article, 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. (In particular, a pipeline processor may in principle be considered to run with the authority of a single user, and the responsibilities of the P2P network do not include automatically composing pipelines based on indirectly asserted supply and demand manifests. This stance does not forbid recursions, though.)
2. Pipeline is composed from individual service components that have specific locators.
3. The P2P network provides a search mechanism using 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 (see Figure 2).

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 include 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 implementing distributed identity or payment services since those are assumed to be included into the P2P infrastructure during later phases.

In general, four levels of abstractions may be identified when requesting a service component:

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.)

Once the pre-conditions of the service infrastructure are met, composing pipelines is relatively straightforward. In short, a pipeline processor is part of a *Wille* visualisation client (see Figure 2). The client maintains user account, working memory, and can access both the distributed P2P services and local applications (e.g. viewer, editor). A pipeline specification includes a (sub)task tree. Each task has input and output, and success criteria. When the order of sibling tasks does not matter, they can be processed in parallel. Each task returns a data object that can be passed to some other task.

The pipeline processor is responsible for managing the data objects (strings, resources, and archives) and may manipulate them when appropriate (e.g. mapping the output of one task into a form suitable to another task). Finally, if the pipeline is successfully completed, the *Wille* visualisation client can pass the last computed data object to some application, e.g., for viewing the results. The success criteria of a task may include a sufficient reference to an appropriate service (e.g. a service identifier, see above), acceptable delay of waiting for the service to complete, acceptable costs, and information whether processing the pipeline can continue even if a particular task fails. A task may fail for several reasons: An appropriate service (e.g. category, price, or access rights) is not available, it fails to respond in an appropriate window of time, or it retrieves or returns an unsuitable data object. Note that in the latter case, an error might be caught only when the pipeline processor attempts to feed the data object to another task. Of course, as a mechanical system, a pipeline is insensitive to factual errors.

# 4 Case Studies

We next illustrate our design of P2P visualisations through experiments. We first outline a concrete conference scenario that includes different kinds of visualisations and review different approaches of implementing it. The idea of lightweight P2P client implementations is then illustrated in terms of simple scripting and Java applications, without the requirement of using an explicit pipeline processor.

## 4.1 Conference Scenario

As a part of the OPAALS consortium, we are organising the OPAALS 2008 conference in October 2008. The conference provides a nice scenario for investigating the practical needs of the real-life use cases for *Wille Phase 1* and *Wille Phase 2*. A set of views and visualisations was created for the workspace of OPAALS 2008. The workspace includes also a conference community for conference delegates as means for discussing the conference themes and networking. The process of creating the workspace can be divided into data collation, visualisation processing and visualisation dissemination. In addition to visualisation, the conference data is used to populate the conference community.

The data representing the main contents of the conference, including people, publications, and events (cf. Möller, Heath, Handschuh and Domingue 2008) was collected from different data sources with a set of tailored reader components. The submission and review process was managed with OpenConf (http://www.zakongroup.com/technology/openconf.shtml) that allows exporting the submission data in a proprietary format. The conference registration data, including basic profile information, was delivered as a regular spreadsheet through email. A pre-existing *Wille Phase 1* pipeline was used to access the profile data of OPAALS community members where "FactsAbout cards", wiki pages with a predefined structure, are used to represent the profiles (see Nykänen et. al. 2007; Huhtamäki 2007). A dedicated *Wille Phase 1* pipeline was created for transforming, aggregating, and querying the raw conference data in a way that an expressive Friend of a Friend (FOAF) -based representation of the conference was created, following roughly the conventions of the Semantic Web Conference Ontology (SWC) (see Möller, Heath, Handschuh and Domingue 2008).

Conference workspace visualisations include simple views such as an interactive program and a list of participants, some traditional mashups including a map of the whereabouts of participants and a tag cloud based on the article keywords as well as a tag cloud created by utilising Self-Organising Maps (see Salonen, 2007) and other more complex visualisations. In addition to visualisations, the collected conference data was used to create and populate the conference community driven by an open source social engine Elgg (http://elgg.org). The participant information was imported into the community and the participants may be given the possibility to populate their profiles with the information they provided during registration. Of course, for motivation, the systems ought to be usable also after the conference. Since the technical OKS infrastructure is still under development, more time was used for implementing components capable of accessing, reading, and interpreting the raw conference data than for actually creating the visualisations. This is typical in information visualisation and semantic data management (cf. Möller, Heath, Handschuh and Domingue 2008). Further, each conference delegate had to fill their profile information manually (possibly by copying data from LinkedIn, Facebook and other sources) to the registration and article submission forms.

A general API for accessing the conference data and digital ecosystem data in general (cf. http://data.semanticweb.org/conference/eswc/2008/html) would add greatly to the possibilities of a visualisation designer. An example of the possibilities provided by explicit data set repenting the article authorship is given by Molka-Danielsen, Trier, Slykh, Bobrik and Nurminen (2007) when creating a visualisation of the evolution of a social network based on co-authorship in IRIS (Information Systems Research in Scandinavia) Conference between 1978 and 2006. In order to give means to access the data programmatically, applications managing conference submission and registration processes would have to provide a RESTful HTTP API or a similar solution. Shared protocols and APIs for authentication and resource-sharing would ease the creation of more integrated and expressive visualisations and tools supporting the conference delegates' work. For now, new accounts have to be created into the conference community even for OPAALS members instead of using the existing DE accounts.

## 4.2 Experimental P2P Visualisation Implementations

We shall next discuss and report implementation experience from two complementary aspects of the visualisation system: Java-based *Wille* Visualisation Service Platform for developing and deploying services, and a common client/pipeline framework for developing visualisation applications.

To get understanding about the next version of our visualisation system, *Wille 2*, a Java prototype was constructed. The prototype focuses on the component-based aspect of the visualisation system by scaffolding the *Wille* Service Platform, which is the part of the visualisation system that hosts and offers visualisation services. Because the objective is essentially to show how services could be deployed to the network and used in pipelines, the peer-to-peer functionality of the prototype was kept minimalistic and the visualisation client was not fully implemented. In this context visualisation client is simply the application that uses the visualisation services. For prototyping it is sufficient to use, e.g., Python scripts as a client to process pipelines. The client architecture will be later refined.

In the early prototypes, communication between the client and the service platform was achieved with Java Remote Method Invocation (RMI). The reasons for using RMI were the lack of a reference framework and possibility to do quick prototyping. After experimenting, the REST architectural style was adopted and HTTP was chosen as the main protocol for accessing components (see Fig. 2). The HTTP servlet implementation was done using Java-based Jetty web server (see http://www.mortbay.org/jetty-6/ and http://java.sun. com/products/servlet/). To maintain the flexibility and OKS integration of the system, the prototype was designed so that selecting appropriate communication protocol for services would be possible.

While clients are in principle able to integrate any components conforming to the OKS Service and *Wille* Service REST API in particular, it is useful to provide a dedicated *Wille* visualisation service platform, or a framework, using which developers can deploy services to the network. Visualisation clients can request a service from a service platform directly without mediators. Additionally, making the job of component developers easier, the service platform is designed to enable integration of external components to the visualisation system. In short, components can be developed with different technologies and integrated into the system and thus shared with others.

Parallel to the work on the visualisation service platform, we worked also on experimental *Wille* clients with Python 2.5 (http://www.python.org/) scripts. The motivation in choosing a scripting language was on reusing readily available features of the language, most importantly variables, loops and native libraries for file I/O and HTTP support.

Three pipelines were created with the client: 1) RDF to graphML converter demonstrating a transformer component and native Java viewer, 2) Self-Organising Map tag cloud visualisation pipeline (see also Salonen, 2007) demonstrating a reader component and Java Applet viewer, and 3) PDF (Portable Document Format) linguistic analyser and visualisation pipeline demonstrating a complete pipeline including an Scalable Vector Graphics serialiser.

Creation of the pipelines was mostly a straightforward process. Each pipeline was written as a single Python script file, less than 100 effective lines of code. Based on the implementations, several repetitive tasks, such as service execution, were identified. Abstracting these tasks into a common framework, a simple class library was written, yielding client scripts less than 50 lines in length.

Based on our experiments, the responsibilities of the clients seem to include the following:

1. Management of temporary data (incl. compression and decompression of component I/O)
2. P2P service search and invocation (potentially incl. optimisation of service execution)
3. Management of and recovery in non-fatal errors (e.g. waiting for or replacing a service), and
4. E-commerce and trust-related interfaces.

In order to abstract pipelines from a particular client implementation and provide pipeline processing as a service itself, declarative pipeline definitions can be created. During Phase 1, Ant scripts were used for this purpose. In the future, conceptual integration with existing pipeline languages such as XProc (http://www.w3.org/TR/xproc/) seems beneficial, for better integration and tool support.

The ability to interface with RESTful HTTP applications was appreciated already during the development of the experimental visualisation pipelines. For instance, in the PDF visualiser pipeline, linguistic analysis was performed with an external service, Topicalizer (a RESTful service freely available at http://www.topicalizer.com/api). This saved us from writing a new component dealing with linguistic analysis. More importantly, it allowed benefitting from the linguistic expertise of the existing service provider. However, from the perspective of a visualisation application, relying upon a single external service provides a serious risk of single-point of failure. Thus, while genuine P2P publication (and replication) of services is not required, for some critical services it might be a necessity.

In practice, it is also useful to be able to equip pipelines with parameters (e.g. data selection and filtering, for customisation). Further, user credentials and tokens are often needed for, e.g., authentication and authorisation for services. A sophisticated pipeline would separate general parameters from e-business authentication and authorisation for enhanced usability and security, etc. Experiments also revealed that mapping between service I/O, iteration-based execution (e.g. loops), service dependency definitions, execution requirements, and abstract service invocation upon meta-data are needed in many pipelines. Using a fully-fledged programming language such as Python, this is relatively simple. Encoding all of these features in a declarative pipeline language would require adding several primitive capabilities to the definition of the pipeline language, which is not feasible (this would yield into a "pipeline" definition with the expressive power of a programming language).

# 5 Conclusion

It seems that P2P has come to stay: By simply looking around we may observe that the nodes of conventional networks decrease in size and increase in diversity and in numbers. Old networks of fixed wiring turn into ad hoc networks with dynamic organisation and several communications media. As the number of the potential use cases increases, also the needs of end-user are multiplied. The ability to create and distribute content and services on grass-roots level makes monolithic service systems look obsolete. As a consequence, the idea of fixed client-server topology is no longer manageable since agents of conventional service networks are effectively turning into nodes of P2P networks.

While this may look like a radical change of paradigm in networked computing, it seems fair to say that the existing technology is "simply" organised to better suit the needs of the end-users. Client-server is still needed since some services simply are best implemented in such a way, and some users and organisations (as legal entities) want to displace resources by ownership. Thus, rather than re-inventing the wheel, the network is (implicitly) given responsibilities that previously were (explicitly) given to (human) network administrators. It seems obvious that visualisation systems composed from services must nevertheless follow this trend. In this case, perhaps the most significant change lies in searching and retrieving data and services. Ideally, visualisation clients must manage searching services, accessing them in various ways, and compensating the services from recently dropped nodes.

While developing our visualisation system, we have discovered that the needs of typical visualisation applications seem to map nicely onto the P2P design philosophy. Also, the idea of component-based visualisations, composed from distributed services in principle allow managing many of the issues of the existing visualisation applications, including adding new data processing components, integrating existing systems, automating manual work, and controlling the visibility of applications and data. Securing the robustness and scalability of composed services, however, implies more requirements to the underlying common network infrastructure, e.g. in order to replace dropping services appropriately. More specific findings include the necessity of supporting the existing "webised" services, not committing to a particular graphical user interface for flexibility, and considering pipelines as a typical design pattern for client applications. Indeed, we feel 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.

Despite the progress being made, however, many of the practical challenges remain. These include social accountability and awareness of pipeline-based visualisation applications, improving the non-technical end-user user experience, reaching critical mass of applications and data, dealing with security issues such as malware, building trust, and supporting e-commerce and other business applications. Further challenges include managing the evolution of the technical infrastructure, and supporting long transactions or off-line applications.

In the near future, we plan to finalise the framework and client specifications, and work on the core visualisations components. This allows building practical visualisation applications and gaining experience for establishing guidelines about generalising the end-user interfaces.

## Acknowledgements

We would like to thank our colleagues involved in the Phase 1 work, including Mr. Markus Mannio and Mr. Antti Kortemaa. Further, we appreciate the help and feedback from the OPAALS community. Many thanks also to the referees who provided valuable feedback.

## References

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.

Herman, I. (Ed.) (2008). *W3C Semantic Web Activity*, World Wide Web Consortium, Available at http://www.w3.org/2001/sw/.

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

Molka-Danielsen, Judith, Trier, Matthias, Slykh, Vadim, Bobrik, Annette, & Nurminen, Markku I. (2007). IRIS (1978-2006) Historical Reflection through Visual Analysis. In *Proceedings of IRIS30,* Tampere, August 2007, Finland. Tampere, Finland.

Möller, K., Heath, T., Handschuh, S., & Domingue, J. (2008). Recipes for Semantic Web Dog Food? The ESWC and ISWC Metadata Projects. In The Semantic Web, Lecture Notes in Computer Science. (Vol. 4825, pp. 802-815). Berlin/Heidelberg: Springer.

Nejdl, W., & Siberski, W, (2006). Schema-Based Peer-to-Peer Systems. In Steinmetz, R., & Wehrle,K. (Eds): *Peer-to-Peer Systems and Applications.* Lecture Notes on Computer Science, Springer.

Nejdl, W., Siberski, W., Simon, B., & Tane, J. (2002). Towards a Modification Exchange Language for Distributed RDF Repositories. In I. Horrocks and J. Hendler (Eds.): *ISWC 2002,* LNCS 2342, (pp. 236–249). Springer-Verlag Berlin Heidelberg.

Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., and Risch, T. (2002). EDUTELLA: a P2P networking infrastructure based on RDF. In *Proceedings of the WWW'02* (Honolulu, Hawaii, USA, May 07 - 11, 2002). ACM, New York, NY (pp. 604-615). Available at http://doi.acm.org/10.1145/511446.511525.

Nykänen, O., Mannio, M., Huhtamäki, J. & Salonen, J. (2007). A Socio-Technical Framework for Visualising an Open Knowledge Space. In: Isaías, P., Baptista Nunes, M. & Barroso, J. (eds.) *Proceedings of the International IADIS WWW/Internet 2007 Conference*, 5-8 October, Vila Real, Portugal (pp. 137-144). IADIS Press.

Nykänen, O., Salonen, J., Huhtamäki, J., & Haapaniemi, M. (2008). *OKS Data Model, version 1.01.* A milestone specification for the OPAALS PROJECT (Contract number IST-034824), July 1 2008 (29 pages). A copy is available at http://matriisi.ee.tut.fi/hypermedia/julkaisut/20080701-oks-dm-v1-01.pdf

Richardson, L, & Ruby, S. (2007). *RESTful Web Services.* O'Reilly, USA.

Salonen, J. (2007). Self-organising map based tag clouds - Creating spatially meaninful representations of tagging data. In *Proceedings of 1st OPAALS workshop* ,November 26-27, Rome, Italy. To appear.

Steinmetz, R., & Wehrle,K. (2006). What is This "Peer-to-Peer" About? In Steinmetz, R., & Wehrle,K. (Eds): *Peer-to-Peer Systems and Applications.* Lecture Notes on Computer Science, Springer.

Stiller, B., & Mischke, J. (2006). Peer-to-Peer Search and Scalability. In Steinmetz, R., & Wehrle,K. (Eds): *Peer-to-Peer Systems and Applications.* Lecture Notes on Computer Science, Springer.

Ware, C. (2004). *Information Visualization (Second Edition): Perception for Design*, Elsevier, USA.

Wehrle, K., Götz, S., & Rieche, S. (2006). Distributed Hash Tables. In Steinmetz, R., & Wehrle,K. (Eds): *Peer-to-Peer Systems and Applications.* Lecture Notes on Computer Science, Springer.