OPAALS PROJECT

Contract n° IST-034824

WP10: Sustainable Community Building

Del10.6 – A proof-of-concept implementation of a visualisation client

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

Contract Number: IST-034824

Project Acronym: OPAALS

Deliverable N°: 10.6

Due date: November 2007

Delivery Date: November 2007

Short Description:

Proof-of-concept requirements, architecture, design, and implementation of a visualisation system for Open Knowledge Space (OKS). The system integrates open source software with a pipelined software architecture and is usable for visualising data independent of the data domain.

Author: (in alphabetical order) Matti Haapaniemi, Jukka Huhtamäki, Antti Kortemaa, Markus Mannio, Ossi Nykänen, Jaakko Salonen

Partners contributed:Tampere University of Technology (TUT)

Made available to: OPAALS consortium

Versioning

Version	Date	Name, organization
V1 for 1st consortium review round	19.11.2007	Haapaniemi, Huhtamäki, Kortemaa, Mannio, Nykänen, Salonen (TUT)
V2 for 2nd consortium review round	4.12.2007	Haapaniemi, Huhtamäki, Kortemaa, Mannio, Nykänen, Salonen (TUT)
V3 for final delivery	12.12.2007	Haapaniemi, Huhtamäki, Kortemaa, Mannio, Nykänen, Salonen (TUT)

Quality check

Internal Reviewers: Oxana Lapteva (UniKassel), Jesús E. Gabaldon (TechIdeas), Juanjo Aparicio (TechIdeas)

Dependences:

Work Packages	WP2, WP5, WP6, WP7, WP9, WP10
Partners	LSE, IITK, IPTI, TechIdeas, UNIS, UniKassel, SUAS, UCE
Domains	Computer science: information visualisation, software architecture, data mining, knowledge modelling, Semantic Web, open source Social science: information visualisation, social networks
Targets	Research communities, SMEs

Specification for OKS visualisation system

Architecture and design, Phase I

Annex I: Full source code package with user documentation

Available from https://matriisi.ee.tut.fi/svn/opaals/public using the following user name and password pair:

User name: opaals
Password: tampere

OPAALS Project (Contract n° IST-034824)

Hypermedia Laboratory
Tampere University of Technology

Specification for OKS visualisation system

Architecture and design, Phase I

29.11.2007

Table of Contents

List of abbreviations

DBE	*Digital Business Ecosystem*
FOAF	*Friend of a friend*
GraphML	*XML-based file format for representing graphs*
GUI	*Graphical User Interface*
HTML	*Hypertext Markup Language*
IDE	*Integrated Development Environment*
MathML	*Mathematical Markup Language*
OKS	*Open Knowledge Space*
OPAALS	*Open Philosophies for Associative Autopoietic Digital Ecosystems*
P2P	*Peer-to-peer computer network*
RDF	*Resource Description Framework*
RSS	*Family of web feed formats used to publish content*
TreeML	*XML-based file format for representing data trees*
SBVR	*Semantics of Business Vocabulary and Rules*
SOM	*Self-organising map*
SPARQL	*SPARQL Protocol and RDF Query Language*
SQL	*Structured Query Language*
SVG	*Scalable Vector Graphics*
URI	*Uniform Resource Identifier*
VRML	*Virtual Reality Modeling language*
X3D	*ISO standard for real-time 3D graphics*
XML	*Extensible Markup Language*
XSLT	*Extensible Stylesheet Language Transformations*
XTM	*XML Topic Maps*

1 Introduction

This document specifies the *implementation of a visualisation client* which is the concrete deliverable (D10.6) from OPAALS task 10.5. The document offers an overview to the technical requirements, architecture, and design of the deliverable. Using this document it is possible to understand the rationale behind the design of the deliverable, expand and extend it if necessary, and integrate it as part of other projects.

Chapter 2 summarises the initial needs and requirements for the system

Chapter 3 details the relevant architectural principles and concepts.

Chapter 4 specifies the actual system design using views to the users, data, system components, and the possible physical layouts.

Chapter 5 gives a summary of the system.

1.1 Key concepts

The definitions below briefly summarise the concepts which are used throughout this document especially in Chapters 3 and 4.

This document specifies and describes the proof-of-concept *visualisation client* that is the outcome of OPAALS task 10.5. In technical terminology, the visualisation client or virtual knowledge workspace described in the OPAALS description of work is essentially an information **visualisation system** or environment that can be used to construct **visualisations**. In visualisation system specification context, a visualisation can be thought as the generic type of data, models, interactions, and representations for specific usage.

Models from data can be created with different methods for different purposes where a **visualisation model** is a specific type of model as an abstraction of the raw data for visualisation purposes. **Visualisation representation** is the result of the mapping of the visualisation model to graphics primitives creating a concrete, graphical view to the data. A visualisation can be instantiated to create a **visualisation instance** which is a concrete visualisation with specific data, model, and representation.

Open Knowledge Space (OKS) and other OPAALS project specific concepts are defined in other project documentation such as the Description of Work (OPAALS Contract 2006) and OKS specification (Briscoe & Iqani 2007).

1.2 Context

The context of the visualisation system specified in this document is shown in Illustration 1.

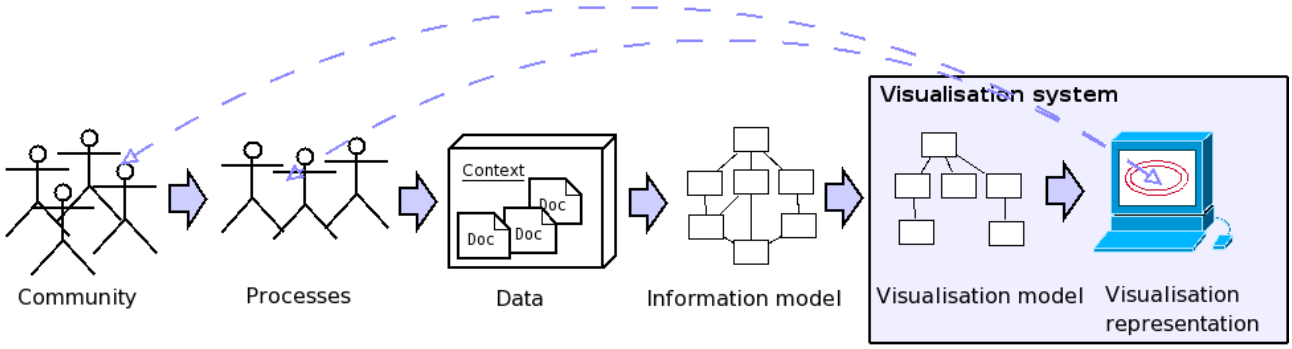

Illustration 1: Context of the visualisation system.

The image illustrates how the data is produced by a community with its informal or formal processes and stored to a data repository (e.g. OKS). The stored data includes the actual content, context, and any available metadata describing the data itself which can be used as source for visualisations. The visualisation system accesses this data in order to provide a representation of the data as a concrete visualisation instance with intermediate modelling steps.

Thus, this document does not describe the community, processes, or the contextual data but the system of processing this data to a visualisation. The visualisations themselves can be used for supporting the community processes and they can describe the community and the data, but in itself they do not define any of these. This connection is shown with the bidirectional arrows from the visualisation system to the community and processes which also highlights the process aspect of visualising information.

The visualisation system and process can act as a critical connection from the technical infrastructure (e.g. OKS) to the community and its other processes. The visualisation process supports the online community building aspects, for example, by increasing the motivation to participate. With this viewpoint, visualisations can be seen as the means of motivating community members by making the contributions visible for the rest of the community. This visibility can then increase the attractiveness of the community and motivate existing members for more active contribution. Additionally, visualisations can increase the community awareness by providing views to the current topics in the community.

Visualisation system support for interactive visual data exploration and mining processes provides a natural bridge from the community data to knowledge browsing and formation. These processes help users in interpreting and navigating the knowledge space to find interesting information and areas to contribute to. In connection to the content creation process and tools, visualisations can also provide the means for validating user input by ensuring that the contribution has the intended effect in the knowledge space. The effect can be, for example, a visualisation illustrating the semantics of a concept by showing it in some specific contextual setting.

2 Needs and requirements

This chapter describes the collected user needs and requirements for the visualisation system used for constructing visualisations. The aim of the chapter is to give an overview to the user expectations and the functionalities needed in a visualisation system for the OKS rather than detail all the requirements in this document. The presented user needs and requirements form the basis for the decisions made in the architectural concepts presented in Chapter 3 and the actual design in Chapter 4.

Due to the evolutionary nature of the OKS and the incremental way of working, the requirements are constantly evolving and the information presented in this static document represents a snapshot of the requirements situation which may not be complete in future. However, this information tries to give a sufficient view to the expectations thus providing means to understand the architecture and design rationale.

First the lists of OPAALS user and visualisation model needs collected from the OPAALS community are discussed. The community needs have been analysed to produce a list of system requirements discussed in the second part.

2.1 User and visualisation model needs

Concrete visualisation needs and ideas for visualisations have been collected from the OPAALS community both in workshops and in OKS community tools (e.g. in *http://wiki.opaals.org/TampereIntegrationWorkshop/Agenda/OKSVisualization* and *http://wiki.opaals.org/CapturingVisualizationNeeds*). The tool in wiki works as a place for constantly collecting new arising visualisation needs from the community. The needs are analysed to provide use cases and technical requirements to the visualisation system to be used to fulfil these needs.

Appendix A presents examples of the collected high level OPAALS user needs which are described in more detail in OPAALS wiki. The needs describe the community expectations and guide the design of the visualisation system and range from very specific cases, such as SBVR vocabulary visualisation, to a more general level OPAALS project visualisation. Satisfying any of the user needs may require the availability of several visualisations comprising of different visualisation models, representations, and interactions according to the user's preference. Thus, it can be seen that the needs are very diverse and cannot be satisfied with any single visualisation but require several, differing visualisations of the data.

In addition to the specific use cases discussed above, a number of more specific needs concerning the type of visualisation models that should be supported have been collected. Appendix B lists community collected needs that are related to the visualisation models. These needs for the models range from graph-like concept map visualisations to geographic and virtual landscapes. While the model needs can generally be satisfied by adding support for the type of model requested, support for different types of representations and interactions is still required to suit the user's preferences. As opposed to the user needs above, the models do not include any reference to the data to be used and thus need to be usable with various types and sources of data. As with the specific user needs, it can easily be seen that the models cannot be supported with a single visualisation.

2.2 System requirements

Appendix C presents a snapshot of the analysed high level system requirements that are further split to technical design. For each requirement a title and a description or a list of related use cases are presented to give more detailed view on each item. The use cases themselves are not detailed in this specification.

The requirements are an end result of the analysis process of the community and project needs including prioritisation, removing duplicate requirements, etc. These are the general functional and non-functional requirements that a visualisation system for OKS should fulfil in order to meet the community and project needs. However, as mentioned earlier the work happens in an incremental and iterative way (as opposed to the traditional waterfall software engineering paradigm) which makes this list ever evolving. To cope with this evolution, a component-based approach has been taken where many of the requirements are actually requirements for specific components to be included into the system architecture and thus not part of the visualisation system architecture itself. This approach is described in detail in Chapter 3.

In any case, the design of the architecture must take into account the functional needs for the individual visualisations to support the inclusion of suitable components to meet these needs. Additionally, the system needs to be able to manage these components and their interfaces. The architectural aspects of the visualisation system are highlighted with the architectural requirements presented in Appendix D. These requirements are either a generalised from the analysis of the above system requirements or implied by the design of the OKS.

3 Architecture

The community needs and requirements presented in Chapter 2 call for an architecture that can support very diverse visualisation needs, models, representations, and client environments. This chapter presents the architectural concepts and decisions made for the specification of the visualisation system.

First, the results of a study to knowledge management and visualisation systems are summarised to lay basis for community visualisation system architecture.

Second, the actual architecture and related concepts are described.

Third, the architectural integration to the OKS infrastructure is explained based on the current understanding of the OKS architecture.

3.1 Community and knowledge management systems

At high level information visualisation is about using interactive visual representations of abstract data in order to amplify cognition (Ware 2004). Accessing the data happens through knowledge management systems which in this case are dependent on communities and web accessible tools. A study of existing community knowledge management and visualisation systems was made to understand the current state-of-the-art in this area (Salonen 2007) since this information is not readily available in published research articles due to the fast-paced evolution of web technologies and communities. The study included classification of knowledge and visualisation needs in on-line communities to understand and anticipate the architectural requirements that an on-line research community would benefit from in a visualisation system.

As a conclusion, four essential areas of knowledge and visualisation needs in community-managed systems were identified: documents, people and discussions, ontologies, and folksonomies.

The results of the study are aligned with the varied needs gathered from the OPAALS community in the sense that visualisation of community knowledge cannot be effectively done with any single visualisation. To effectively visualise community and knowledge it is necessary to have a visualisation system architecture that can be adapted for various use cases and models in the four areas of knowledge mentioned above and detailed in the study.

3.2 Pipelined visualisation architecture

The presented visualisation system architecture uses the widely-used notion of approaching information visualisation and data pre-processing with pipeline architecture (see, e.g. Kreuseler & Schumann 2002; Xproc 2006; *http://cocoon.apache.org/*) with the assumption that visualisation techniques can be understood as series of parameterised transformations from raw data to user view (see, Chi 2000). As such the visualisation system can be understood as a pipeline where several steps from accessing and filtering of the data to the actual rendering of the final visualisation are performed. More detailed description about the designed architecture in information visualisation research context has been published (Nykänen et al. 2007).

3.2.1 Overview

Pipeline components are the "construction blocks" put in the pipeline that define the final output of

the system. In the pipeline view constructing visualisations is about selecting the suitable components to be put in the pipeline, defining the parameters for the components, and connecting them in appropriate manner to reach the desired visualisation as an end result.

Traditionally visualisation pipeline environments contain components that are developed to the specific visualisation pipeline environment that is being used (e.g. Haeberli 1988;Upson et al. 1989). While this is a viable approach in many cases, it does not address the specific issues raised for the OKS environment in supporting existing domain tools and open source community development. As no software development experience can be assumed from the users, the integration of existing components is often the only way to achieve certain functionality while potentially achieving major savings in resources. Additionally, the evolving nature of the OKS and the heterogeneous community prevent the visualisation components to be tied to specific technologies or programming languages.

Thus, an approach of enabling the integration of open source components regardless of their implementation details is taken for the visualisation pipeline. The characteristics of the open source model of development and its relation to OPAALS are studied and well explained, for example, by Fitzgerald, Gaughan, and Ågerfalk (2007).

In the pipeline architecture some of the key concepts presented in Chapter 1 - visualisation model, representation, and interaction - can be realised by separate components in the pipeline. For example, one component could create the visualisation model out of the more general information model, another component can map the graphic primitives to the visualisation model, and yet another provide the interactive interface to the representation. Thus, the visualisation instance is the end result of a pipeline processing within the visualisation system.

3.2.2 Pipeline structure

The components in the pipeline need to satisfy only a minimal set of conditions to be integrated to the pipeline enabling the use of ready-made or open source development. For any component C to be integrated to the pipeline architecture, a wrapper W is created for the component adapting the component as part of a visualisation pipeline. In essence, the wrapper contains description of the component and general instructions on how the component is run, thus creating a defined interface for the component to interact with other components as part of a pipeline as shown in Illustration 2.

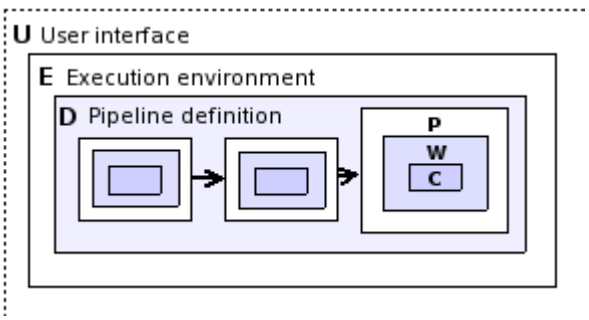

Illustration 2: Pipelined visualisation system architecture

The minimal interface consists of the input and output locations for the data the component is using but it can also contain any other functionality that the component supports. The wrapper can be reused in different pipelines and needs to be created only once for a component in the visualisation system. Typically one component has one wrapper but in cases where a component may perform several distinct functions it is possible to create several wrappers for each functionality of one component. In this case one physical component can be thought as containing several logical

components where the selection of the wrapper decides the logical functionality that is to be used in a pipeline. For example, if one physical component performs both data transformation and display functions depending on its parameterisation, separate wrappers may be defined for each of these logical functions. Additionally, one wrapper may use multiple components and wrap their behaviour to one logical entity. In this case there is no difference in the wrapper behaviour compared to the single component case.

Together a component C and a wrapper W create a *pipeline component* that can be instantiated to any pipeline in the visualisation system. In order to instantiate the pipeline component a set of parameters P are needed to describe the functionality of the component in a specific pipeline. The parameters instruct the wrapper and component to work in the way that fulfils the needs of the specific pipeline and components connected to it. The component C, wrapper W, and parameters P form a *pipeline component instance* that is pipeline component for a specific use case. A visualisation pipeline is formed by connecting several component instances together which involves selecting the pipeline components to be used and parameterising them to match the desired functionality of the pipeline.

Pipeline definition D is a specification defining the pipeline structure identifying, for instance, the components and their execution order. Together the component parameterisation P and the specification D define a *pipeline instance* that specifies the exact output of the pipeline. Separating the pipeline specification and its execution instances has advantages in visualisation system design, for example, when creating large number of visualisations (Bavoil et al. 2005).

3.2.3 Pipeline execution

In order to execute the pipelines an execution environment E is needed. In broad sense, the execution environment consists of the operating system, any external software that can be used by the visualisation system, the software needed to execute the pipelines (D), software libraries that may be used by multiple components, and any global parameterisation needed across all pipelines in the visualisation environment. In addition to the pipeline execution, the execution environment can perform other tasks, such as caching intermediate results which enables better support for large data volume exploration (Bavoil et al. 2005).

User interface U can be considered as an external and optional part of the visualisation system but, in practice, such an interface is needed for human use of the system. It should be noted, however, that the visualisation system itself is not dependent in any specific user interface and such interfaces may be constructed by external parties.

Collaborative visualisations can be achieved, for example, by connecting the pipelines of several users. This would mean, for example, that user X would create a pipeline where user Y would use or connect to and continue with his own pipeline. Collaboration can naturally happen also via sharing data sources, components, visualisations themselves.

3.2.4 Architecture advantages and limitations

Wrapping components to common pipeline architecture and configuring them to a specific pipeline enables the creation of visualisation pipelines constructed of pre-existing software components independent of their implementation details. A common component parameterisation interface through the wrapper enables a creation of lightweight pipeline definitions connecting the components. Although wrapping own components or modules as part of a larger system to include customised behaviour has been used in visualisation systems (e.g., in Bavoil et al. 2005), this often requires integration at source code level which limits the possibilities for including third party software.

The design enables flexible integration of pre-processing pipelines and visualisations, which has been seen as an enabler for exploration of large information spaces (Kreuseler & Schumann 2002). The pre-processing stage enables the support for connecting to different types of data sources and adding data transformations which have been identified as challenges in visualisation system design (Bosch et al. 2000).

The design also enables supporting various user needs and community visualisation systems by enabling the integration of existing visualisation or data processing software to the pipelines in a defined manner. As the amount of pre-existing visualisation software is very large, creation of a visualisation software repository has been proposed as one solution in managing them (Borner & Zhou 2001). The presented visualisation system architecture provides such a repository and also the means of connecting the software together. Software components in the pipeline can also be run individually either with or without the wrapper which makes it possible for the components to work in "pipelined" or "standalone" mode. The wrapper interface also enables the creation of common graphical user interface for specifying both the pipeline design and individual component configuration in the pipeline.

The pipeline definitions and component parameters can be saved and shared to enable collaboration and distribution of the pipeline. Distributing the components into a P2P network would enable a distributed processing system that could provide more performance from visualisation in CPU-intensive visualisations. However, mapping the visualisation pipeline onto network requires suitable models (e.g. Mengxia et al. 2004; Schonhage & Ellens 2000) and compatible P2P network interfaces to be defined and available.

The architectural design also has its limitations. The loose coupling of the components via the wrapper interface means limited direct inter-component communication, pipeline-awareness, and reduced control on the component behaviour. The lack of direct communication between components means that functions such as demand-driven processing or data streaming need to be implemented in the wrapper interface if such functions are desired. Pipeline-aware components can, of course, be constructed integrating more tightly in the pipeline which resembles more traditional visualisation pipeline design.

3.3 Relation to the visualisation reference model

In information visualisation, the work by Card, Mackinlay, and Shneiderman (1999) is often referred as an information visualisation reference model. The model presents separation of data and visualisation models to enable multiple visualisations of the data. In similar fashion visualisation models are proposed to be separated from the display or visualisation representation to enable multiple different views to the visualisation. Finally, modular controllers are presented as a way to manage interaction with the users in a flexible way.

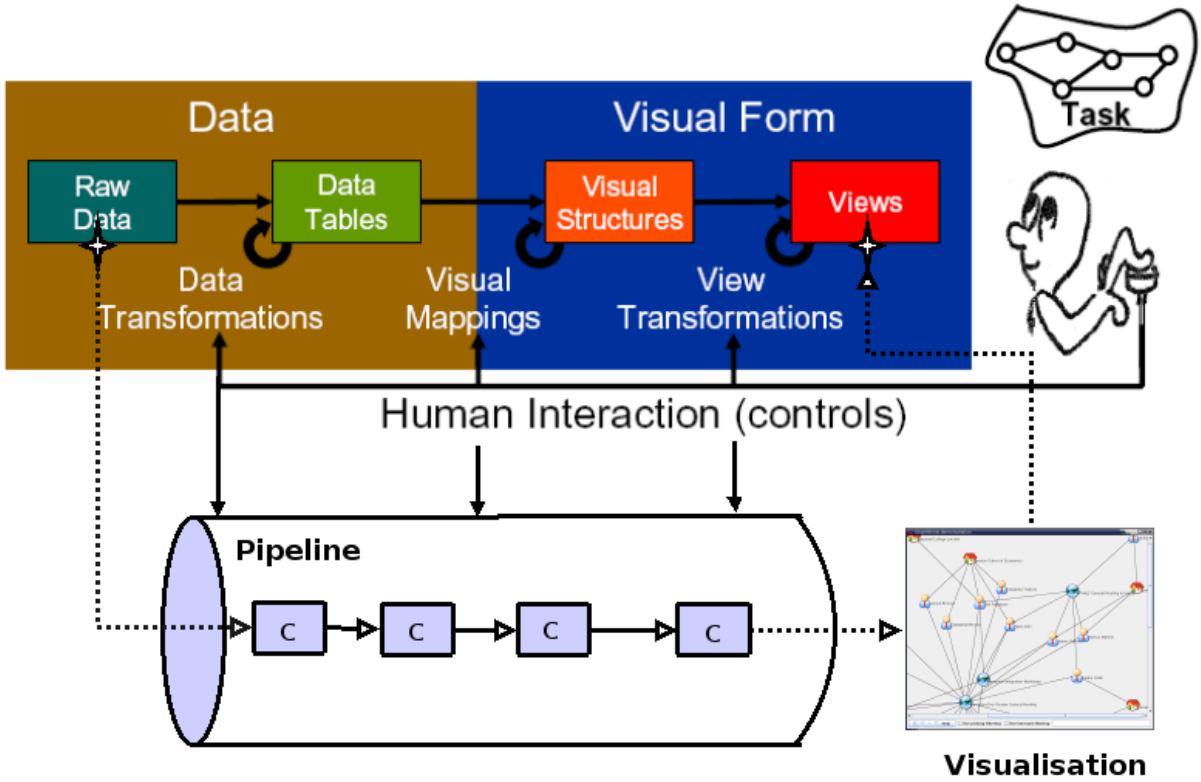

Illustration 3: Pipeline architecture relation to the visualisation reference model (adapted from Miksch (2005)).

In the pipelined architecture the separation of the different parts can be naturally done using the component-based structure which is shown in Illustration 3. The transformations from different stages to the next in the reference model can be managed with separate components in the pipeline. The controller providing interaction can be the final, viewer component in the pipeline together with the visualisation environment interface. By changing, for example, the viewer component different interactions with the visualisation can be provided. Other components in the pipeline can be replaced in similar fashion to enable flexible modification of the visualisation. Changing the visualisation properties via the viewer component and the pipeline structure via the visualisation environment offers a flexible tool for visual data mining (see, e.g. Keim 2002) and exploration. With the process of selecting and configuring suitable components in the data processing pipeline and the viewer component, the guidelines presented by the famous "visual information seeking mantra" (Shneiderman 1996) can be adhered to.

3.4 Interfacing with the OKS

The OKS is an evolving knowledge space based on various technologies and interfaces that is described in more detail, for example, in OKS design documents (Briscoe & Iqani 2007).

Based on the current understanding of the OKS, the visualisation system will eventually be setup in a P2P distributed network environment. The network will contain a distributed datastore(s) with several clients to access the datastores. The visualisations are mainly created from the data and models stored in the distributed datastore(s).

3.4.1 Modes for OKS and visualisation system integration

In Illustration 4 there are three different architectural modes for visualisation system described with letters A, B, and C. The objective of the visualisation system of the OKS is to support each of these cases by being able to display visualisations based on the data. However, the supported functionalities may be different in each of the cases depending on the design of the OKS GUI and the P2P network.

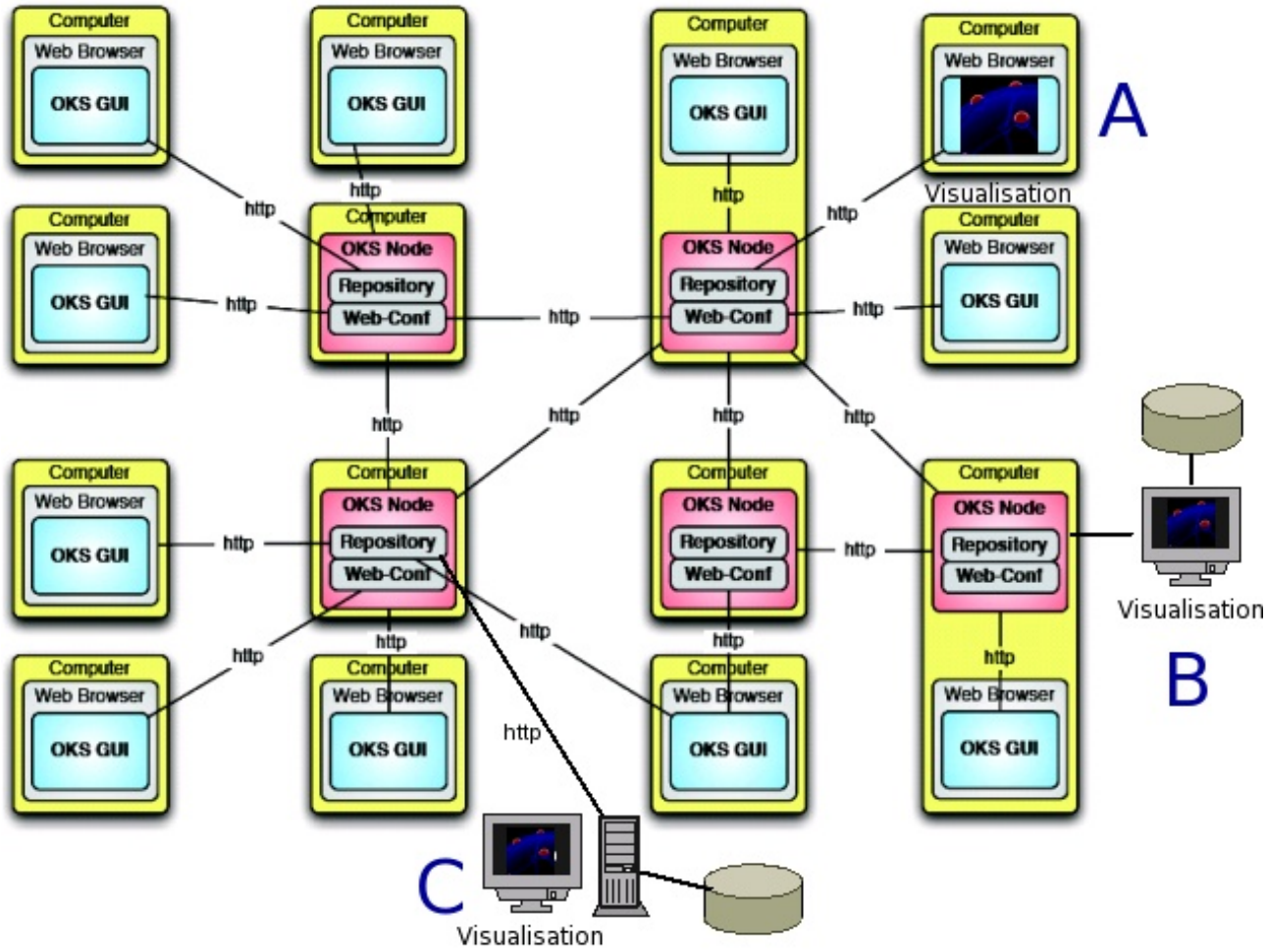

Illustration 4: Vision of the OPAALS OKS environment with visualisations. Adapted from Briscoe and Iqani (2007).

Mode **A** is a case where the visualisations are a part of a P2P network client node connecting to the actual OKS Node. In this case the visualisation system is situated in a different node that than the GUI. The visualisations will be shown as part of the OKS GUI displayed within a web browser whereas the system is running as a server in the OKS Node. Before reaching the target state of distributed P2P network, there will probably be a centralized phase with the network topology is different and case **A** will be dominant.

Mode **B** is a case where the visualisations are shown in the same (OKS) node where the data repository is located. In this case the system itself is located in the same node where the visualisations are accessed. This enables visualisations to be displayed as part of the OKS GUI or as separate applications. In this illustration the visualisation system is also accessing an external

datastore in addition to the OKS data repository, which is an optional but often a necessary scenario to support.

Mode **C** describes a case where the visualisation system and display are in an external (to the OKS) computer and accessing the OKS repository together with an external datastore. In this case the OKS infrastructure is treated as any other data source that the visualisation system can access and produce visualisations about.

3.4.2 Requirements to the OKS

In order to integrate a pipelined visualisation system to the OKS, the minimum requirement is to be able to interface with the data contained in the knowledge space. This provides the possibility to access the actual content to visualise. In the pipelined architecture this can be done by adding suitable data access components (or adapters) to the pipeline enabling access to various types of data sources. This means that the architecture does not constraint the types of data sources that can be accessed but rather enables the use of multitude of data sources by integrating suitable components to the pipeline. The visualisation system also needs to be able to process the data model(s) used in the OKS to effectively visualise the data. The models can be in the form of ontologies or schemas that describe the data. The pipelined visualisation architecture supports processing of the OKS models by allowing the integration of suitable pipeline components for processing and transforming the OKS models to visualisation models.

Another requirement enabling tighter integration of the visualisation system to the OKS is the access to the various services provided by the OKS for the systems interfacing with it. These services include user authentication and naming of accessible resources in the OKS context. This is not a mandatory requirement but without access to the OKS common services, the visualisation system needs to implement, for example, its own schema for user identification and access. This will result in duplicate work and prevent seamless integration of the OKS and visualisation system from the user point of view. In the pipelined architecture the integration of the services can be done in both environment (E) and individual pipeline level. At the environment level common user identification schema and naming can be provided whereas individual pipeline components can access whatever services needed for the specific pipeline. A service of accessing the authoring tools and policies of the OKS is another way of ensuring the integration of the content creation and visualisation which would enable possibilities for launching content editors from the visualisations.

Finally, in order to have a seamless integration of the visualisation system to the OKS from the user perspective, a user interface level integration is also desired. The architecture supports this by enabling the use of several user interfaces for the system. These include, for example, a local command-line based interface or a remote web service interface. The integration to the OKS can be done for both web services and local installations of the OKS node.

4 System specification

This chapter describes a visualisation system design based on the pipelined architecture introduced in Chapter 3. Several views to the visualisation system are taken in order to create a necessary specification of the full system.

First, a user view is given describing the roles and high level activities to be performed with the system.

Second, a data view is described including the conceptual data layers and types of data that are needed to be processed by the system.

Third, a view to the component structure describing the functional parts that the visualisation system must include to be able to fulfil the essential requirements is given.

Fourth, a physical view describing the actual setup of the functional components to make a real-world system is presented.

4.1 User view

This section defines the necessary user roles that are needed to specify and understand the visualisation system. It does not provide a detailed description and guidance for the processes that the users are performing while using the system. Detailed information on what tasks exist and how they are performed can be found in the visualisation system user guide (Wille visualisation environment 2007). However, Illustration 5 shows an example flow of events that clarifies the tasks of the common roles in typical visualisation system use cases.

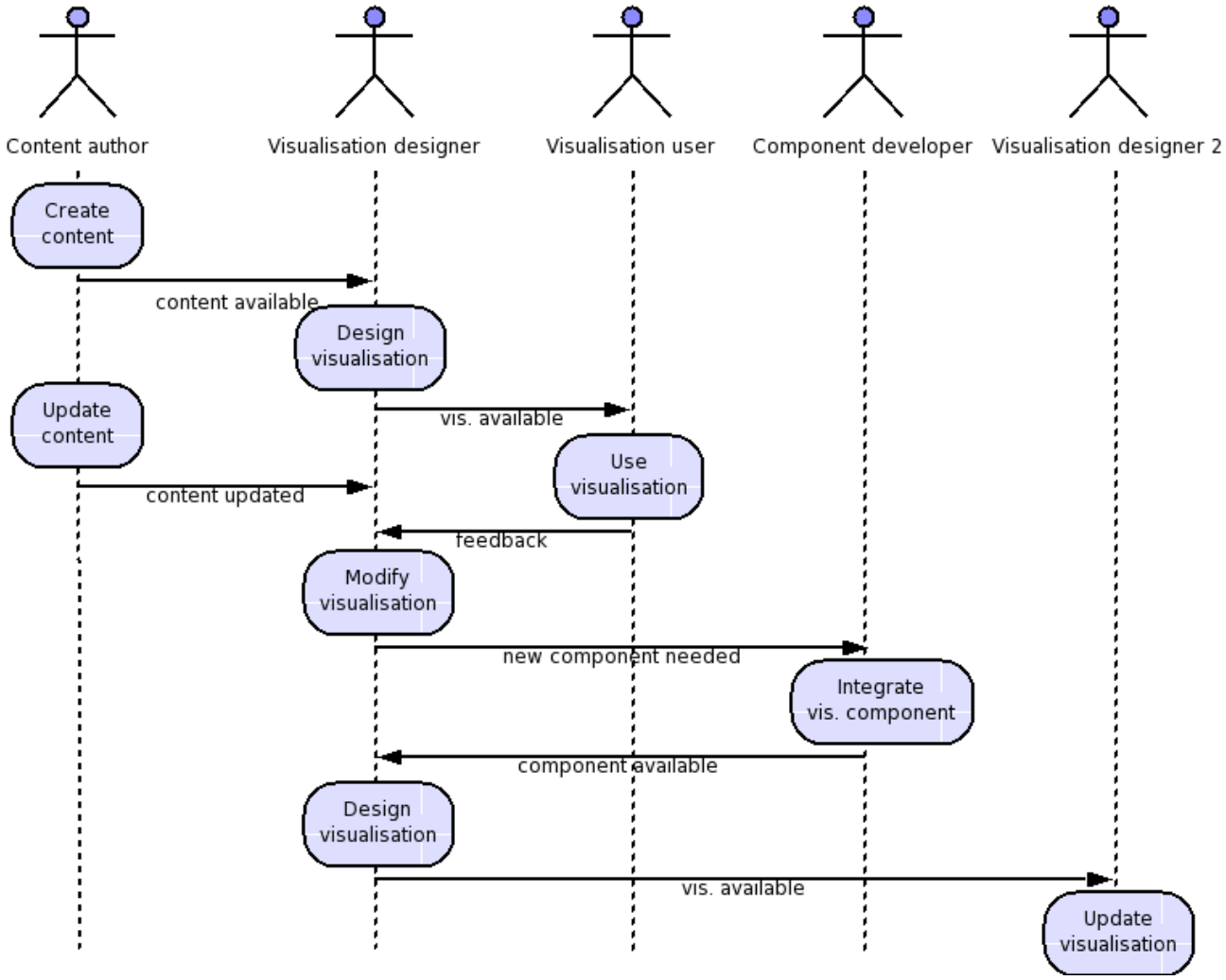

Illustration 5: Typical use cases for different roles in the visualisation system.

Based on the requirements listed in Chapter 2 the following user roles can be identified related to the visualisation system.

4.1.1 Visitor

The visitor is the viewer of the visualisations. The visitor can also interact with the visualisations within the limits of the implemented interaction in them. For example, if the visualisation contains controls for navigation then the visitor can navigate and change the visualisation using those controls. The visitor does not have any interaction with the visualisation system itself (and thus not shown in Illustration 5) but only the visualisation instances which are the output from the visualisation system.

4.1.2 User

The user interacts with the visualisation system and the existing pipelines. Typical tasks for the user are searching, parameterising, and executing existing visualisation pipelines. The user does not design new pipelines or introduce new components to the visualisation system but rather uses the pre-existing components and pipelines.

4.1.3 Designer

The designer interacts with the visualisation system by specifying and creating new pipelines. Typically the designer searches for existing components, designs a pipeline, and publishes it for users. The designer does not have the capability to create new components or wrappers to the environment.

4.1.4 Developer

The developer can introduce new components to the visualisation system to be used by the designers in creating new pipelines. Introducing new components means also developing wrappers for them to enable integration to pipelines. This role typically needs some software engineering skills.

4.1.5 Administrator

The administrator can alter any properties of the visualisation system itself to create an environment for the developers, designers, and users. This means, for example, adding support for totally new types of pipeline interactions or otherwise modifying the system. This role typically needs software engineering skills.

4.1.6 Content author

The content author does not have direct interaction to the visualisation system but it is listed here for completeness. Typically the content author role can be added to any of the roles above where a person is working as the content creator using the content editing tools available to the OKS.

4.2 Data view

This section describes the conceptual data model used in the system as a layered data model and as examples of the types of data processed at different layers. The data types are presented as examples since the actual data models used in the visualisations are not dependent on the visualisation architecture but rather the integrated components. Each data type can be supported if suitable components exist in the pipeline for its processing.

4.2.1 Layered data model

A reference model *Information Visualization Data State Reference Model* (Chi 2000) is used which describes information visualisation techniques using data stages and transformations. The reference model has four data stages between which the data is transformed with three different data transformations. In addition, each data stage has operators which operate with the data within the data stage. The reference model is shown in the left side of Illustration 6 as an activity diagram where transformations are shown as activities and the data stages as output objects from the transformations. The data state reference model has close resemblance to the visualisation reference model presented by Card, Mackinlay, and Shneiderman (1999).

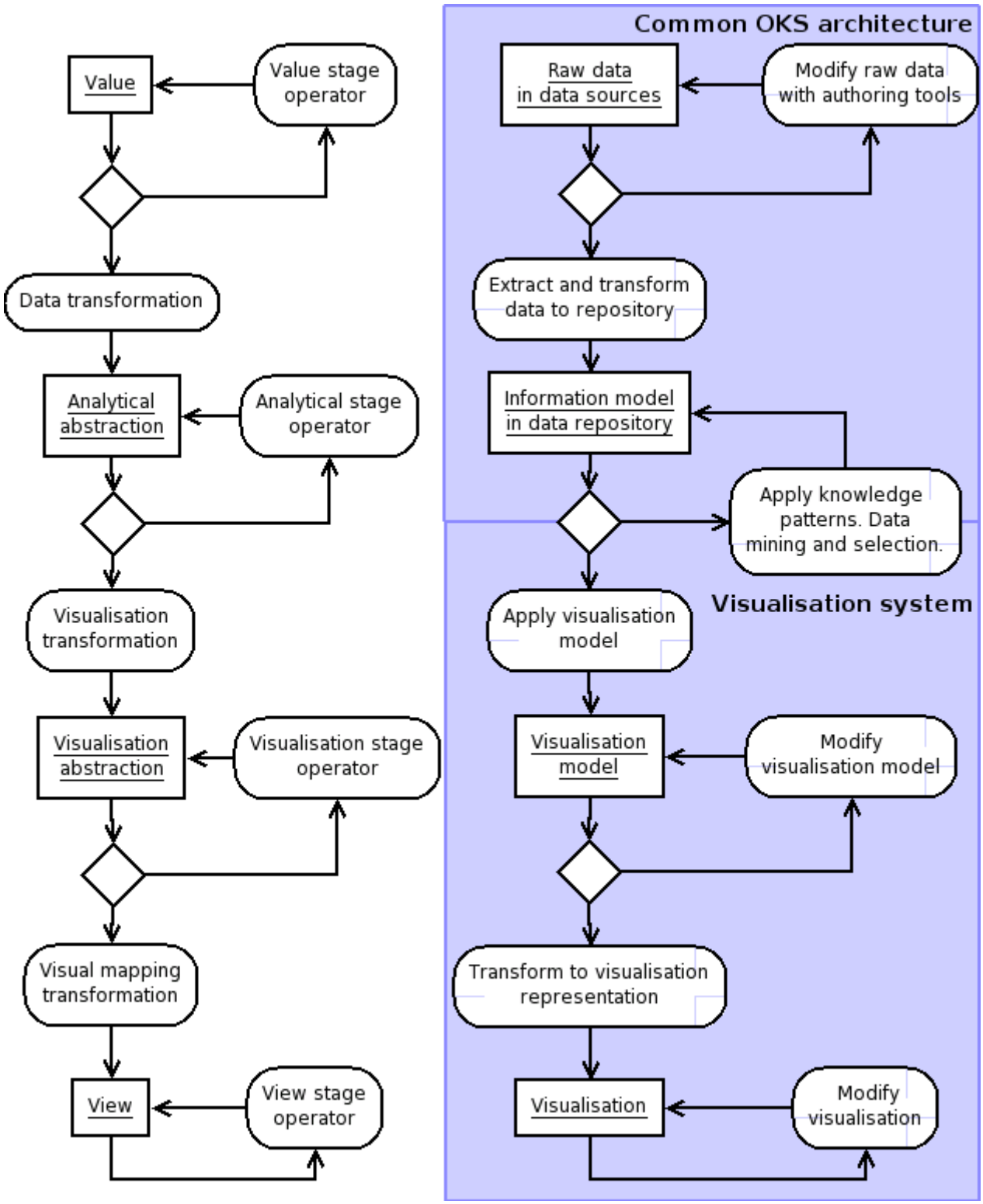

Illustration 6: Left diagram -- Information visualisation data state reference model (Chi 2000). Right diagram – OKS visualisation system data model adapted from Chi (2000).

The data state model is used as a reference model in OPAALS context since all the stages are present in the OKS and highlighted in the requirements of the first chapter where different needs were introduced to the visualisation system and the visualisation models.

The right side of Illustration 6 presents an actual OKS layered data model based on the visualisation reference model. The parts from the model belonging to the visualisation system are from the *Data repository* level onwards whereas the *Data sources* and *Data repository* levels are part of the common OKS architecture. However, the data processing pipeline architecture of the visualisation system can be used to accommodate processing also for these levels with the addition of suitable components. This processing could include, for example, semantic integration (Nykänen 2007) of the OKS data. In addition to the OKS data repository, the visualisation system architecture enables connecting directly to other data sources and repositories thus not limiting only to OKS specific data.

In the pipelined architecture the components in the visualisation pipelines take input from one data layer and typically produce an output to the next layer, for example, taking input from the *Data repository* and producing a *Visualisation model*. However, the components may also produce output to the same or even previous layers. Output to the same layer as input is typically made in cases where a model is transformed to another model more suitable to the task at hand. This happens, for example, when transforming a model to another due to a need for another view to a data presented already in an existing visualisation model. Producing output to previous layers may happen in cases where the data has been enriched during the pipeline process and the results are taken back to lower levels for storage or for some other usage. In practice also caching between the different data levels needs to be done to increase performance of the system. In the pipelined architecture this means storing each of the components output for future reference.

The parts of the layered visualisation system data model shown in Illustration 6 are detailed in subsequent sections in a top-down fashion.

4.2.2 Common OKS architecture data model

This section gives a brief overview on the parts of the data model that are not part of the visualisation system itself but in practice need to exist in the OKS for visualisation and also other purposes (e.g. information searching, processing, and browsing). This section is closely related to Chapter 3 where the architectural requirements to the OKS were described. The overview given to the parts is focused on the visualisation aspects of the system and not intended to describe the OKS architecture in detail.

Raw data in data sources is the first data stage from the top which illustrates the raw data to be visualised in the various data sources that can be both internal and external to the OKS. In OKS these may be the various OKS tools, for example, the wiki, forums, blogs, OKS journal, etc. External data sources may include, for example, a geographic data or an ontology repository. The modification of this raw data is typically made with some external (to visualisation system) tools by the content author role. This could mean, for example, editing wiki content with wiki editors. Although this activity is not part of the visualisation system, the system should enable providing necessary links from the visualisations to the authoring tools to enable, for example, easy transition from visual navigation to content creation.

The raw data should be extracted and transformed to a repository to enable accessing the data for visualisation and other uses. This extracting activity consists of the location and description of the data from various sources to the OKS *Information model in data repository*. This activity may include selection of the data sources, selection, restructuring, transforming, and modelling the data. The information model and data repository contain the data from various sources in some (semi-) formal format. The repository can be an OPAALS wide "OKS Repository" containing data according to the OKS information models using the schemas and vocabularies designed for the OKS. The repository should support querying and reasoning about data and could be, for example,

a RDF datastore with SPARQL interface. Physically the repository can be a distributed one and accessed via P2P network. In a much simpler form the repository can be a collection of files in local disk in suitable format for further processing.

Applying knowledge patterns for the data repository consists of the steps where the data in the repository is refined to the level needed for visualisation and other possible uses. This can be supported by using predefined knowledge patterns represented, for example, with schemas in the data repository. Subactivities contained in this activity are, for example, selecting pattern, selecting data, designing query, etc. Although this specification is for the OKS visualisation system, the data and knowledge processing components in the pipelines are usable for general knowledge management and mining purposes which is shown in Illustration 6 by the positioning of the *Apply knowledge patterns* activity between the OKS common parts and the visualisation system.

4.2.3 Visualisation system data model

This section gives a top-down description of the visualisation system parts in the right side diagram of Illustration 6.

Apply visualisation model is the first activity consisting of the selection of the general visualisation type (e.g. graph, map, chart) and describing the selected data from the repository to appropriate visualisation model. Several visualisation models can be described to one set of data resulting in several types of visualisations for one data set. This could mean, for example, transforming a SPARQL query XML-result to GraphML for graph representation. The activity results in creation of a *Visualisation model* which is the format where the selected data is described in appropriate way for certain type of visualisation and several visualisation models can exist for one data set. The data format used here describes the data in suitable format for the visualisation but not the actual representation. These models can be transformed to other models to enable visualising the selected data in another format without new processing in the data repository (e.g. making a 3D abstraction of a graph). The data in the visualisation model (and the visualisation) layer should preferably use standardised formats to enable standard interfaces to the components. This would enable using the visualisation processing and viewer components interchangeably so that the most suitable processing and viewer components can be selected from existing open source applications. Examples of the data formats used here could include GraphML, TreeML, XTM, RSS, etc. The visualisation model parameters can be changed to modify the model. This can mean, for example, deciding the style of tree (e.g. cone tree, disk tree) to be visualised or enriching the model with additional elements.

Transform to visualisation representation is the activity where the visualisation model is transformed to the actual visualisation. This means transforming the visualisation model to the actual representation format for the visualisation viewer component. One visualisation model (e.g. graph) can be transformed to several visual representations which enable using multiple visualisation viewers with one visualisation model. The representation data format should preferably be standard so that multiple viewers can be used for the visualisation, for example, SVG, bitmap graphics, VRML, or X3D. The end result of the transformation is the visual representation that can be presented to the user with a viewer component. *Visualisation* is thus the end result shown to the user using the selected data (Applying knowledge patterns), visualisation model (Apply visualisation model), representation (Transform to visualisation representation) with the selected viewer pipeline component. The visualisation representation can be modified with the *Modify visualisation* activity. This means, for example, changing in the colours, panning, zooming, and filtering using the interactive controls contained in the viewer component. The visualisation can enable changes also to the lower level data states by allowing modification of the visualisation model (changing the visualisation type) and data repository (changing the data to be visualised).

These links to previous levels are not shown in Illustration 6 but in practice need to exist and correspond to the modification of the pipeline structure or parameters.

4.3 Software component view

This section gives a view to the actual software components needed in the design of the architecture introduced in Chapter 3. This section also describes technologies and software that are used for the actual implementation of the pipelined visualisation system. This section is referring to the architecture description in Chapter 3 in the pipeline architectural concepts.

4.3.1 Execution environment

As described in Chapter 3 and Illustration 2, the execution environment (E) is responsible for running the pipelines and producing the actual visualisation instances as an end result. The main responsibility of the execution environment is to be able to execute the pipeline definitions (D) to produce the visualisation instances.

Considering the requirements for the execution environment (outlined in Chapter 2), a decision to use Apache Ant (*http://ant.apache.org/*) for implementation was made. Essential features of Ant are, from the pipeline design point-of-view, the possibility to define dependencies between different components for defining pipeline structure and the ability to control component execution easily for wrapper creation. Ant also offers a variety of ready-made and community created tasks for various purposes. There is a solid user and developer base with good documentation and Ant is being used in various open source and commercial products. In addition, Ant is Java-based enabling multi-platform functionality and comes preinstalled in many systems.

As shown in Illustration 2, the system architecture is based on data processing pipelines consisting of individual components which can be connected as a pipeline to achieve the desired result. In the Ant environment the pipeline definitions (D) are represented by Ant XML build files. As discussed in Chapter 3, the components (C) are accompanied by the respective wrappers (W) and parameterisation (P). In the Ant execution environment the components can be any executable applications while wrappers are Ant XML scripts adapting the component to the pipeline. The pipeline definition Ant script includes and executes the necessary component wrapper Ant scripts. Parameters are represented by Ant properties which can be thought as static variables with the initial value being static.

Lower level languages such as Java or Python would also be possible alternative languages for the execution environment implementation but their usage would require more development and maintenance work.

4.3.2 Open source components

In the pipelined system architecture many of the requirements listed in Chapter 2 can be reduced to the act of including a specific component as part of the visualisation system. Thus, a study was performed to chart suitable components for visualisation pipelines and the results of study are summarised in Appendix E. Due to the characteristics of the OKS and the visualisation system, only components with suitable open source style licenses are included. Furthermore, the focus lies on tools that have support for typical operating system configurations (Windows, Linux, and Mac OSX).

Information in the appendix is focused on the input and output formats of each component since

they define the interface to other components in a pipeline. *Yes* and *No* indicate whether the tool supports each of the input/output formats and *N/A* is used when the support (or the lack of it) cannot easily be confirmed. The classification separates *Clients* which have a display capability and *Libraries* which do not. Libraries can typically be used as a transformation or filtering component in a pipeline whereas Clients are used as viewer components at the end of a pipeline.

The use cases information in the table illustrates the suitability of the component to some specific tasks. In this case the following tasks were chosen based on the OPAALS context:

1. General (e.g. viewing visualisations)
2. Social networks
3. Data networks
4. Statistical
5. Conversions (e.g. from one format to another)

The appendix is a summary and more detailed information was also collected about each tool including information about software component website, license, operating system support, dependencies, available interfaces, main functionalities, and typical use cases. This information is needed to understand the suitability of the component for visualisation system integration.

The tool study is focused on tools directly related to visualisation tasks. As the visualisation pipelines can also be thought as general data processing pipelines and visualisations often include complex data processing from the source data system, also other data processing components may be and have already been integrated in the environment. Appendix F contains a list of the external software components that have currently been integrated as part of the system. Integrating the components as part of the visualisation system can typically be done with small effort and has currently been done based on the needs of the actual visualisation cases. Thus, most of the components from the study summarised in Appendix E could be added as part of the system if necessary.

In addition to the open source tools, information was collected also about some commercial tools to enable comparison with the available open source components. The work for collecting information about the components is still continuing.

4.3.3 Developed wrappers and components

The previous section focused on open source components that could be integrated as part of the environment. As outlined in the architecture description in Chapter 3, integration of external components requires creation of wrapper(s) to enable component functioning as part of a pipeline. Additionally, for some specific cases ready-made open source components cannot be found or the tasks are so trivial that custom implementation is justified.

Appendix G provides a list of the wrappers and components that have been implemented and included as part of the visualisation system. The list includes also items listed that do not have a unique wrapper associated to them which means that the components are used by wrappers utilising multiple components.

Current focus with the integrated and developed wrappers and components is on semantic and other web technologies (e.g. XML, RDF, Ajax) due to the OPAALS research and OKS evolution direction. However, the environment itself does not mandate the usage of any specific technologies and the focus could be altered or extended by integrating components from some other domains.

4.3.4 User interface

The system architecture supports multiple user interfaces which can be added to the system. Due to the selection of Ant as the execution environment, the native user interface of the visualisation system is command-line based. All features of the environment can be managed by editing the parameterisation files and running the execution environment in command-line.

In addition to the command-line interface, many Integrated Development Environments (IDE) have support for the Ant execution environment and thus provide a user interface to the system. This enables integration of the system, for example, to Eclipse (*http://www.eclipse.org/*) based environments.

To support all users and cases presented in Chapter 3 (in Illustration 4), a web-based interface has been developed to the system. The interface is based on Django web development framework (*http://www.djangoproject.com/*) and provides web browser interface either to local or remote visualisation system installation presented in Chapter 4 section about physical view to the system. An example screenshot of the web interface is presented in Illustration 7.

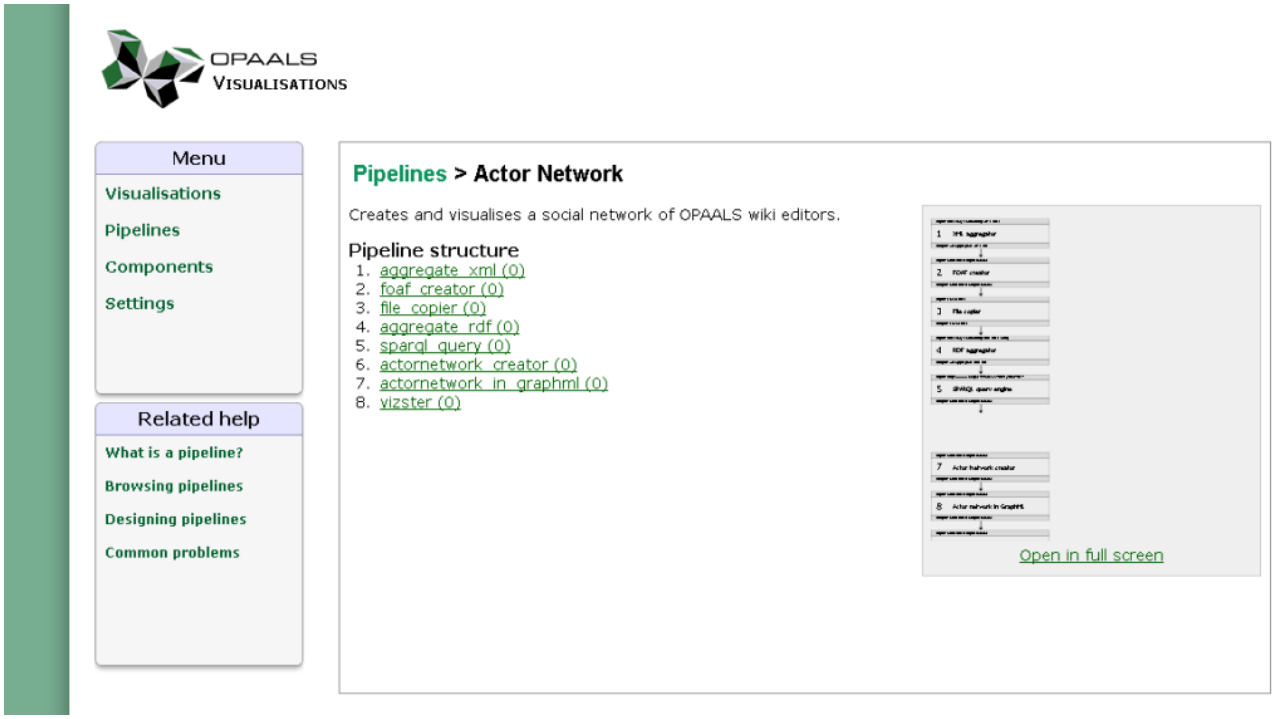

Illustration 7: Example screenshot from the visualisation system web interface.

4.4 Physical view

This part specifies the physical setup consisting of the deployment and integration aspects of the visualisation system. Three different configurations supported by the system are presented. At this phase there are dependencies to external components especially in the distributed case which make detailed specification impossible. These will be updated in later phases.

4.4.1 Local installation

Local installation describes a system installation that is done to the user's computer which means that the user is directly and locally interacting with the system and the visualisations. The system is

self-contained and can be used in off-line mode. This setup corresponds to case C described in the architectural OKS integration in Illustration 4. The user interface to the system is file based and can be used with text editors or development environments such as Eclipse.

Optionally, a graphical user interface can be installed which requires installation of some external components. These optional components are highlighted with lighter edges in Illustration 8 which shows the components and their dependencies in the local installation.

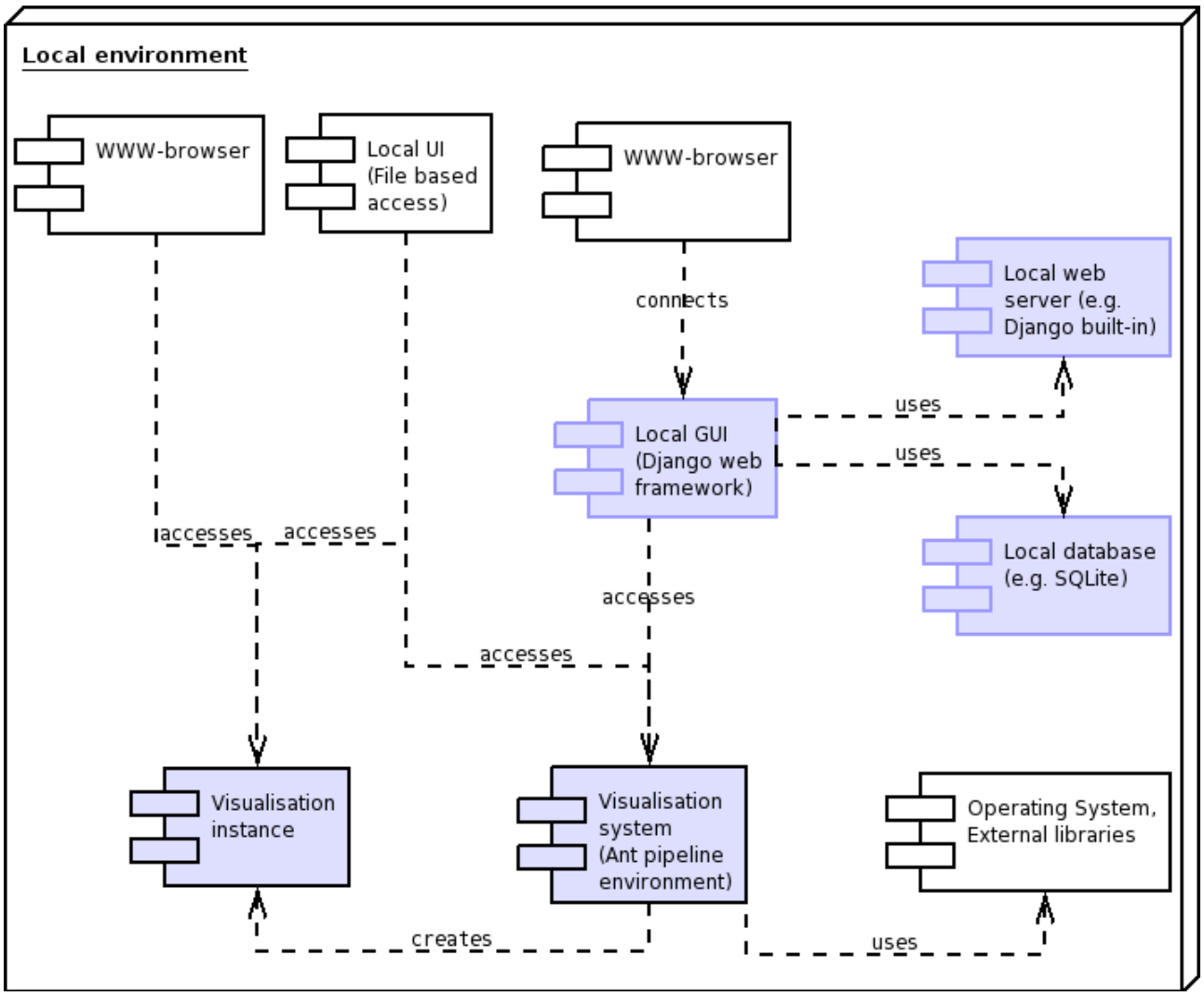

Illustration 8: Local installation of the visualisation system.

If the optional graphical user interface is installed a web browser can be used to interact with the system as shown in Illustration 7. In any case the command line interface interacting with the visualisation system files is available. In this setup all the features and roles (in section 4.1) can be accessed from the local computer.

4.4.2 Server installation

Server installation is a configuration where the visualisation system resides in another computer than the user accesses. This means that the access to the system is via a network and the server must offer a network service interface for the system. In this configuration no software installations are required at the user computer except a standard WWW-browser to access the network service. This

is the setup used in case A in the architectural OKS integration shown in Illustration 4. Illustration 9 shows the conceptual integration of the visualisation system to the OKS.

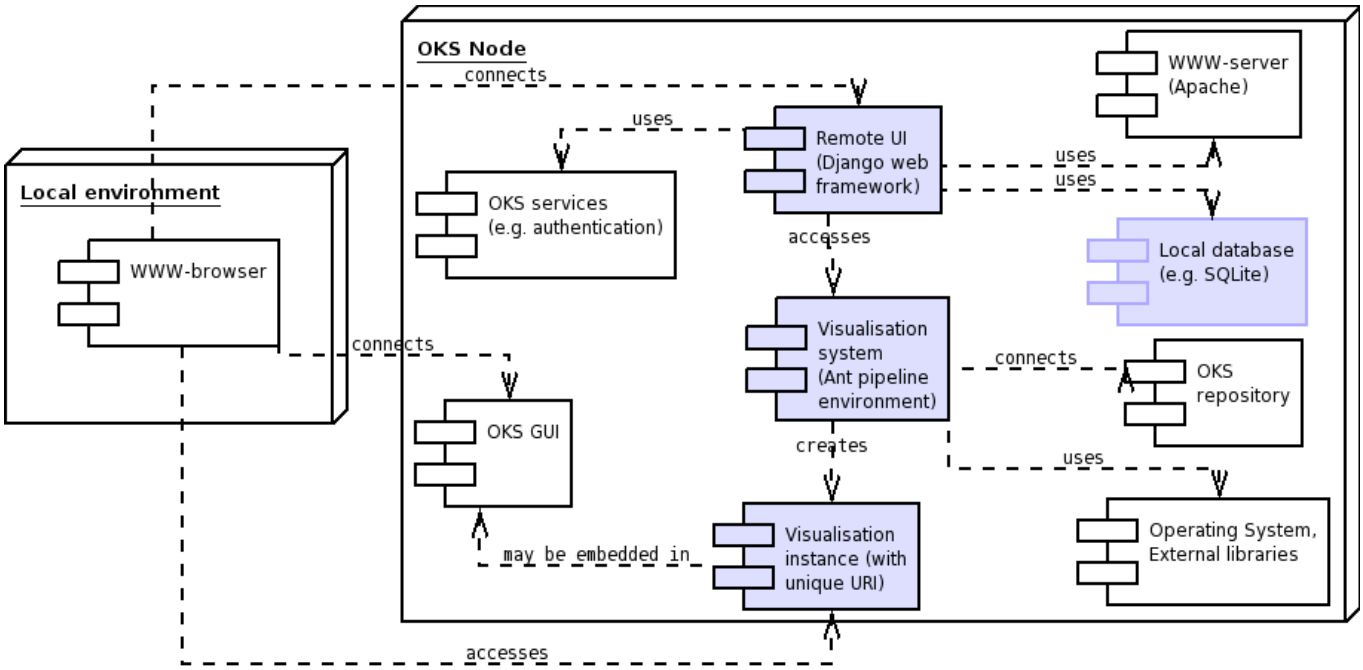

Illustration 9: Conceptual integration of the visualisation system to the OKS Node.

Components with darker (blue) background are part of the visualisation system where the ones with strong (black) outline are essential to it and the ones with lighter (blue) outline can be replaced with other similar components. The components with white background are external to the visualisation system and are not delivered as part of it. The external components include both general components expected to exist in a web server and OKS specific components based on the Open Knowledge Space specification (Briscoe & Iqani 2007). The image shows only dependencies essential from the visualisation system perspective and does not include all dependencies between other components. For example, there is a natural dependency between the OKS GUI and the repository but it is not shown because it is not directly related to the visualisation system.

The visualisation system is accessed via its own web interface for designing and executing the visualisation pipelines. The visualisation interface needs to access OKS services for user access and authentication to enable personalised user activities. The OKS services can also provide the interface to the planned distributed data storage. In addition to a standard web server (Apache) installation, the web interface utilises a local user information SQL database that can be delivered as part of the visualisation system. The web interface interacts with the underlying visualisation system which connects to data repositories that can be both internal and external to the OKS. To utilise all components available for the visualisation system, the system may take advantage of optional libraries present at the server.

As an end result the system produces the actual visualisation instances which can be integrated as part of the OKS web GUI. The instances are provided with unique URIs in the web server to ease integration to other OKS web services. The users may access the instances also directly without using the OKS web GUI by referring directly to the URI given by the system.

When compared to the local installation the client-server architecture cannot display all types of visualisations due to the limitations of the client. Modern WWW-browsers have features and plug-

ins available for displaying variety of content but there are cases when support does not exist. In practice, this affects mostly non-Java custom visualisation viewer components. This setup does not necessarily offer all features for all roles (in section 4.1) to be available remotely due to the limitations of the browser interfaces. This concerns mainly administrator and developer role tasks which can be fully performed only in the OKS Node.

4.4.3 Distributed system

The OKS is planned to evolve towards a distributed knowledge space where the data is stored in a P2P network without centralized control. The visualisation system can be adapted to the OKS data distribution by using the data access service offered by the OKS framework. In this scenario only the data is distributed but the visualisation system itself is not.

An extended possibility is to distribute also the pipeline definition and parameters of the pipeline components in the P2P network. The parameters can be considered as any other data in the distributed network without any special processing required. This scenario would enable collaboration via sharing of the pipeline configurations across the network.

Taking the distribution further, a P2P network with nodes acting as individual service providers (as in DBE) offers the possibility of distributing the visualisation system itself into P2P network nodes. In the pipelined architecture this means that the pipeline components would be distributed in the network as services receiving input and providing output via the service interface. In this scenario the visualisation components and pipeline itself are distributed across the network as described Illustration 10.

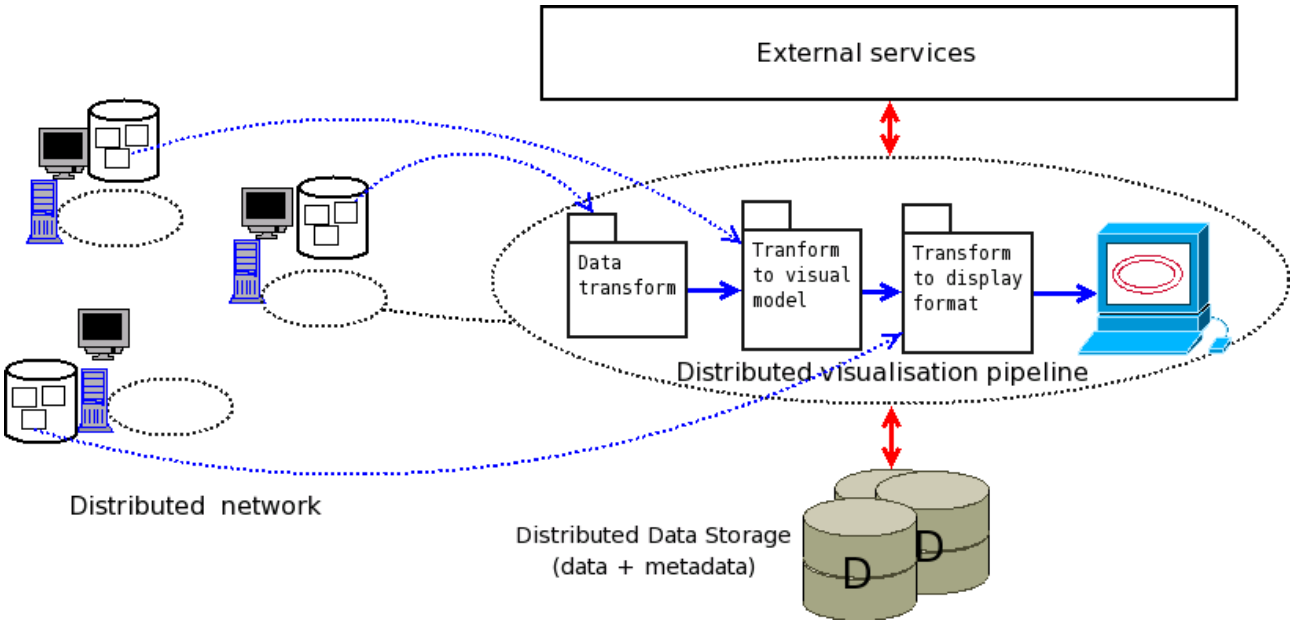

Illustration 10: Distributed visualisation system.

This setup would enable searching and utilising in visualisations whatever components are available not only locally but across the P2P network. It also distributes the processing across several computers offering possibilities for performance optimisation. However, the distributed setup also adds complexity to the system, adds overhead to the processing, and is dependent on the services offered by the network (e.g. for available service location). This scenario can be considered something the visualisation architecture enables but is heavily dependent on the underlying P2P architecture and interface.

5 Summary

In this document requirements, architecture, and design of a pipelined visualisation system were presented. The architecture is based on pipeline paradigm where individual components can be inserted to form a data processing pipeline and visualisations are considered as series of data transformations from data sources to the actual visualisations. The architecture can be integrated to the established information visualisation reference model (Card & Mackinlay & Shneiderman 1999) and data state reference model (Chi 2000).

The architecture enables easy integration of existing domain or open source software components as part of the visualisation pipelines while keeping the pipeline definitions lightweight which may enable considerable savings in resources. Data processing in the pipelines can happen at several levels enabling the pipelines to function as general data processing pipelines in addition to visualisation purposes which provides support for visual data exploration and mining of large information spaces. The component-based approach enables support for different types of data sources, caching intermediate results, and integration of existing data processing software making the system capable of adapting to various user needs.

The system supports multiple user roles including visualisation users, designers, and developers. Collaboration between different users is supported via cooperatively working on a pipeline and sharing data sources, components, or visualisations between users. The system enables a visualisation process that can integrate to other community processes, such as content creation, publication, and knowledge space navigation. Content creation can be supported by integration of the visualisations to the content creation tools as data sources or embedding the visualisations to the tool interfaces. Additionally, visualisations themselves are OKS content as the system enables community members to design and publish their own visualisations. Processes for visual data exploration, navigation, and mining are supported via interactive visualisation and pipeline parameterisation together with the integration of suitable data processing components.

A proof-of-concept implementation of the system has been done based using operating system independent technologies (Apache Ant). A number of open source components have been integrated as part of the system and a study on existing open source visualisation components has been performed to chart the potential software to be integrated to the system in future. Currently integrated components include, for example, adapters for current OKS tools in use (e.g. wiki, blogs) and various (semantic) web related components.

The system implementation may be deployed as local software or as a server whereas possibilities for P2P distribution of the system have been initially charted. A web interface to support the server installation has been designed for remote use of the system. Integration and dependencies to the OKS has been planned in several modes depending on the technical implementation of the OKS. These modes include deployment as part of an OKS node or as independent system connecting to the OKS.

Work is still needed with the system especially in detailing the P2P implementation, tighter integration to the OKS, and further enhancing the user interface to more easily support collaboration and visualisation designer tasks. Appendix H presents an overview to the current status of the system requirements.

6 References

Bavoil L., Callahan S.P., Crossno P.J., Freire J., Scheidegger C.E., Silva C.T., Vo H.T. 2005. *VisTrails: enabling interactive multiple-view visualizations*. In Proceedings of IEEE Visualization 2005.

Borner K., Zhou Y. 2001. *A software repository for education and research in information visualization*. In Proceedings of the Fifth International Conference on Information Visualisation, 2001.

Bosch R., Stolte C., Tang D., Gerth J., Rosenblum M., Hanrahan P. 2000. *Rivet: a flexible environment for computer systems visualization*. ACM SIGGRAPH Computer Graphics, vol. 34, no. 1, pp. 68-73, February 2000.

Briscoe G., Iqani M. 2007. *Open Knowledge Space*. Accessed 7.9.2007. http://www.iis.ee.ic.ac.uk/~g.briscoe/paolo/oks/OKS.pdf

Card S. K., Mackinlay J. D., Shneiderman B. (eds.). 1999. *Readings in Information Visualization: Using Vision To Think*. Morgan-Kaufman, 1999.

Chi E. H. 2000. *A Taxonomy of Visualization Techniques using the Data State Reference Model*. In Proceedings of the IEEE Symposium on Information Visualization.

Fitzgerald B., Gaughan G., Ågerfalk P. 2007. *OPAALS Deliverable 8.1: Papers on basic characterisation of the OSS 2.0 phenomenon*. Accessed 12.10.2007. http://files.opaals.org/OPAALS/Year_1_Deliverables/WP08%20/D08.1-Papers_on_OSS_2.0_phenomenon.pdf

Haeberli P. 1988. *ConMan: A Visual Programming Language for Interactive Graphics*. In Proceedings of SIGGRAPH'88, pp. 103-111. ACM Press, 1988.

Keim D. A. 2002. *Information visualization and visual data mining*. IEEE Transactions on visualization and computer graphics, vol. 7, no. 1, Jan-Mar 2002.

Kreuseler M., Schumann H. 2002. *A Flexible Approach for Visual Data Mining*. IEEE Transactions on visualization and computer graphics, vol. 8, no. 1, pp. 39-51, Jan-Mar 2002.

Mengxia Zhu, Qishi Wu, Rao N.S.V., Iyengar S. 2004. *Adaptive visualization pipeline decomposition and mapping onto computer networks*. Proceedings of the Third International Conference on Image and Graphics (ICIG'04).

Miksch S. 2005. *Impulsvortrag zu Informationsvisualisierung*. Accessed 11.11.2007. http://www.ifs.tuwien.ac.at/~silvia/wien/gwa/ws05/InfoVis.pdf

Nykänen O. 2007. *Implementing context-specific views to distributed (rule) databases with fuzzy logic*. In Proceedings of IADIS International Conference WWW/Internet 2007.

Nykänen O., Mannio M., Huhtamäki J., Salonen J. 2007. *A socio-technical framework for visualising an open knowledge space*. In Proceedings of IADIS International Conference WWW/Internet 2007.

OPAALS Contract. 2006. *Annex 1: Description of Work*. http://wiki.opaals.org/WP0_Project_Management

Salonen Jaakko. 2007. *Knowledge management for visualising social online services*. Master of Science Thesis. http://matriisi.ee.tut.fi/hypermedia/julkaisut/di_jaakko_salonen.pdf.

Schonhage B., Ellens A. 2000. *Information exchange in a distributed visualization architecture: the shared concept space*. In Proceedings of International Symposium on Distributed Objects and Applications, 2000.

Shneiderman B. 1996. *The eyes have it: A task by data type taxonomy for information visualizations*. In Proceedings of the IEEE Symposium on Visual Languages, pages 336-343, Washington. IEEE Computer Society Press, 1996.

Upson C., Faulhaber Jr. T., Kamins D., Laidlaw D., Schlegel D., Vroom J., Gurwitz R., van Dam A. 1989. *The Application Visualization System: A Computational Environment for Scientific Visualization*. IEEE Computer Graphics and Applications, vol. 9, no. 4, pp. 30-42, 1989.

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

Wille visualisation environment. User Guide. 2007.

Xproc: An XML Pipeline Language. W3C Working Draft 17 November 2006. http://www.w3.org/TR/2006/WD-xproc-20061117/.

Appendices

Appendix A: User needs collected from the community

Name	Short description
Navigation aid	Visualisations to help navigating the OKS
OPAALS project visualisation	Visualise work, project, and people relations
Visualizing OPAALS partner network	Way to show OPAALS partner organizations and the people to understand the relationship of persons and organizations
Research topic visualisation	Visualise different research topics in the OPAALS community
Geographical map	A geographical map showing the locations of partners
OKS sitemap	View the structure of the entire OKS for finding sections that users previously did not know about
Activity trend	See topics as an activity trend where the changes for a topic can be seen as a timeline
Describing and prioritising links	Prioritising and characterising links between research and project areas
Links to OPAALS external world	Visualise OPAALS to the general public and media
Manipulation of data	Data needs to be edited and manipulated via visualisations
SBVR vocabulary visualisation	SBVR vocabulary visualisation and extension by the community

Appendix B: Visualisation model needs collected from the community

Name	Short description
Argument map visualisations	Visualisation of logical argumentation as a argument map diagram
Associative map visualisations	Visualising data as an associative map (e.g. SOM)
Concept map visualisations	Support for graph-like (e.g. concept map, mind map) visualisations
Geographical visualisations	Support for showing data within a geographical map
Timeline visualisations	Visualisations where data is presented as a timeline
Topic map visualisations	Visualisations with topic maps
Virtual landscape visualisations	The system must support Benediktine Space type of visualisations

Appendix C: Snapshot of functional and non-functional requirements

Name	Short description or related use cases
Enable animated visualisations	Add animation to visualisations
Changing visualisation properties	Save visualisation customised with interactive capabilities, Use visualisation interactive capabilities, Add interactive zooming capability to visualisation, Add interactive panning capability to visualisation, Add custom interactive capabilities to visualisations, Add interactive graphical objects modification capability to visualisation
Collaborative visualisations	Share visualisation to other user, Load existing visualisation, Extend existing visualisation
Enable data modification through visualisation	View data from visualisation, Modify data from visualisation
Hyperlinking in visualisations	Hyperlinking in visualisations must be supported.
Inclusion of external data sources	Authenticate to data sources, Select data sources for visualisation
Launching applications from visualisation	View data from visualisation, Modify data from visualisation, Add interactive application launch capability to visualisations
Linking visualisations	Refresh visualisation, Connect visualisations
Managing accounts	Authenticate to data sources
Multiple visualisation clients	Support several viewers for the visualisation system.
Saving visualisations	Save visualisation, Share visualisation to other user, Save visualisation customised with interactive capabilities, Create new visualisation
Selecting visualisation data	Filter data for visualisation, Add interactive data selection capability to visualisation

Sharing visualisations	Share visualisation to other user, Load existing visualisation, Extend existing visualisation
Support for multiple source data systems	Authenticate to data sources, Select data sources for visualisation
Support for user profiles	Save personal preferences, Modify personal preferences, Load personal preferences
Support multiple visualisation models	Search visualisation components, Select visualisation component, Configure visualisation component, Connect visualisation components
Utilising source system metadata	Source system metadata should be usable when accessible
Extending visualisation environment	Add new visualisation component, Create new visualisation component
GUI for visualisation viewing	View visualisation

Appendix D: System architectural requirements

Name	Short description
Integration of existing components	The system must support the integration of existing open source or domain specific visualisation components.
Technology support	The system must support various visualisation development technologies and programming languages.
Operating system support	The visualisation system must be executable in various operating system and hardware configurations (Windows, Mac OSX, Linux/Unix).
Support OKS distributed architecture	The visualisation system must integrate to the OKS distributed architecture.
System performance	The system must consume reasonable amount of resources to be usable in a desktop and server environments.

Appendix E: Survey on open source visualisation tools

Tool	Input / Import							Output / Export									Classification	Use cases
	XML	RDF	GraphML	dot	SVG	Bitmap	X3D	XML	RDF	GraphML	dot	SVG	Bitmap	X3D	Web	Display		
Adobe SVG Viewer	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No	No	Yes	Client	1
Ant2dot	No	No	No	No	No	No	No	No	No	No	Yes	No	No	No	No	No	Library	5
Argumentative	Yes	No	No	No	No	No	No	Yes	No	No	No	No	Yes	No	No	Yes	Client	3
Batik	No	No	No	No	Yes	No	No	No	No	No	No	Yes	No	No	No	Yes	Library	1, 5
Demotride	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No	No	Yes	Client	1
Dia	Yes	No	No	No	No	No	No	Yes	No	No	No	Yes	Yes	No	No	Yes	Client	1, 2, 3
Digital Graph	Yes	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No	No	Library	1, 3
Dynagraph	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Library	2, 3
EdiTTm	Yes	No	No	No	No	No	No	Yes	No	No	No	No	No	No	Yes	Yes	Client	2, 3
ESOM Tools	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	4
Tool	XML	RDF	GraphML	dot	SVG	Bitmap	X3D	XML	RDF	GraphML	dot	SVG	Bitmap	X3D	Web	Display	Classification	Use cases
Extrema	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	4
Filters	No	No	No	No	No	Yes	No	No	No	No	No	No	Yes	No	No	No	Library	5
Flux Player	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No	No	Yes	Client	1
FreeImage	No	No	No	No	No	Yes	No	No	No	No	No	No	Yes	No	No	No	Library	5
FreeWRL	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No	No	Yes	Client	1
Gapminder World	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	1, 4
Geomview	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	1, 4
GGobi	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	3, 4
Grand	No	No	No	No	No	No	No	No	No	No	Yes	No	No	No	No	No	Client & Library	3
graph2svg	Yes	No	No	No	No	No	No	No	No	No	No	Yes	No	No	No	No	Library	4, 5
Tool	XML	RDF	GraphML	dot	SVG	Bitmap	X3D	XML	RDF	GraphML	dot	SVG	Bitmap	X3D	Web	Display	Classification	Use cases
Graphviz	No	No	No	Yes	No	No	No	No	No	No	Yes	Yes	Yes	No	No	Yes	Client & Library	1, 2, 3
Graph::Easy	No	No	No	No	No	No	No	No	No	No	No	Yes	No	No	Yes	No	Library	3
GUESS	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes	No	No	Yes	Client & Library	3
gViz	No	Yes	N/A	N/A	No	No	No	No	No	N/A	N/A	No	No	No	No	Yes	Library	3
IBM Many Eyes	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	1, 4
IHMC CmapTools	Yes	No	No	No	No	No	No	Yes	No	No	No	Yes	Yes	No	Yes	Yes	Client	3
InfoVis CyberInfrastructure	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	3
InfoVis Toolkit	Yes	No	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	Yes	Client & Library	3
Inkscape	No	No	No	No	Yes	Yes	No	No	No	No	No	Yes	Yes	No	No	Yes	Client	1, 2, 3, 4
IsaViz	No	Yes	No	No	No	No	No	No	Yes	No	No	Yes	Yes	No	No	Yes	Client	3
Tool	XML	RDF	GraphML	dot	SVG	Bitmap	X3D	XML	RDF	GraphML	dot	SVG	Bitmap	X3D	Web	Display	Classification	Use cases
Jambalaya	No	Yes	No	No	No	No	No	No	Yes	No	No	No	No	No	No	Yes	Library	3
JavaSOM	Yes	No	No	No	No	No	No	No	No	No <;!-- dot Output -->	No	Yes	No	No	No	Yes	Client	4
JFreeChart	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Library	4
JGraph	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes	No	No	Yes	Library	2, 3, 4
JSynoptic	No	No	No	No	No	No	No	No	No	No	No	No	Yes	No	Yes	Yes	Client	3, 4
JUNG	No	No	Yes	No	No	No	No	No	No	N/A	N/A	Yes	Yes	No	No	No	Library	2, 3, 4
KraftPlot	No	No	N/A	N/A	No	No	No	No	No	N/A	N/A	No	No	No	No	Yes	Client	2, 3
Longwell	No	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	3
MathGL	No	No	No	No	No	No	No	No	No	No	No	No	Yes	No	No	No	Client & Library	1, 3
Mondrian	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	4
Tool	XML	RDF	GraphML	dot	SVG	Bitmap	X3D	XML	RDF	GraphML	dot	SVG	Bitmap	X3D	Web	Display	Classification	Use cases
Octave	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	4
OntoViz	No	Yes	No	No	No	No	No	No	Yes	N/A	N/A	N/A	Yes	No	N/A	Yes	Client	3
OpenDX	No	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No	Yes	Client & Library	2, 3, 4
OpenOffice.org Draw	No	No	No	No	No	Yes	No	No	No	No	No	Yes	Yes	No	Yes	Yes	Client	1
Owl Interactive	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	1, 3
Pajek	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes	No	Yes	Client	2, 3
phpGMap2	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Library	3
PHPlot	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Library	1, 4
Piccolo	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client & Library	3
Ploticus	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes	No	No	Yes	Client & Library	1

Tool	XML	RDF	GraphML	dot	SVG	Bitmap	X3D	XML	RDF	GraphML	dot	SVG	Bitmap	X3D	Web	Display	Classification	Use cases
PLplot	No	No	No	No	No	No	No	No	No	No	No	Yes	No	No	No	No	Library	3
Prefuse	Yes	No	Yes	No	No	No	No	Yes	No	Yes	No	No	Yes	No	No	Yes	Client & Library	2, 3
Processing	Yes	No	No	No	Yes	No	No	No	No	No	No	Yes	No	No	No	Yes	Client & Library	3
Protégé	No	Yes	No	No	No	No	No	No	Yes	No	No	No	No	No	Yes	Yes	Client	2, 3
R	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes	Client & Library	4
Relo	No	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	3
RLPlot	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Client	3
SocNetV	No	No	No	Yes	No	No	No	No	No	No	N/A	No	Yes	No	No	Yes	Client	2
SoNIA	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	2
Spin3D	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	1, 3, 4
Tool	XML	RDF	GraphML	dot	SVG	Bitmap	X3D	XML	RDF	GraphML	dot	SVG	Bitmap	X3D	Web	Display	Classification	Use cases
Topic Map Tools	Yes	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No	Yes	Client	2, 3
Tulip	No	No	No	Yes	No	No	No	No	No	No	No	Yes	Yes	No	No	Yes	Client	2, 3
Welkin	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	2, 3
VisIt	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Client	3
Visone	No	No	Yes	N/A	No	No	No	No	No	Yes	N/A	Yes	Yes	No	No	Yes	Client	2
Vizant	No	No	No	No	No	No	No	No	No	No	Yes	No	No	No	No	No	Library	3
VTK	No	No	No	No	No	Yes	No	No	No	No	No	No	Yes	No	No	Yes	Client & Library	1, 4
X3D Tool Kit	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No	No	Yes	Library	3, 5
ZGRViewer	No	No	No	Yes	No	No	No	No	No	No	N/A	Yes	No	No	No	Yes	Client	1, 2, 3

Appendix F: Currently included external components

Component	Web location
AntelopeTasks_3.4.1	http://antelope.tigris.org/
ant-contrib-1.0b3	http://ant-contrib.sourceforge.net/
ant-jnlp-war-0.9	http://ant-jnlp-war.sourceforge.net/
antform-bin-2.0beta2	http://antforms.sourceforge.net/
arq-1.4	http://jena.sourceforge.net/ARQ/
batik-1.6	http://xmlgraphics.apache.org/batik/
colt-1.2	http://dsd.lbl.gov/~hoschek/colt/
commons-collection-3.2	http://commons.apache.org/
commons-logging-1.1	http://commons.apache.org/
grand-1.8	http://www.ggtools.net/grand/
graphl-0_2	http://sourceforge.net/projects/graphl
graph2svg	http://code.google.com/p/graph2svg/
htmltidy	http://tidy.sourceforge.net/
informa-0.7.0-alpha2	http://informa.sourceforge.net/
jdom-1.0	http://www.jdom.org/
jena-2.5.2	http://jena.sourceforge.net/
jung-1.7.6	http://jung.sourceforge.net/
pellet-1.4	http://pellet.owldl.com/
prefuse-beta	http://prefuse.org/
saxon	http://saxon.sourceforge.net/
vizster	http://jheer.org/vizster/
xmltask-v1.14	http://www.oopsconsultancy.com/software/xmltask/
Google Maps	http://maps.google.com/
Simile Timeline	http://simile.mit.edu/timeline/
zgrviewer-0_7_2	http://zvtm.sourceforge.net/zgrviewer.html

Appendix G: Developed pipeline wrappers and components

Name	Type	Short description
3DForest	Component and wrapper	Depicts XML information sources as 3D trees.
actornetwork	Component and wrapper	Describes a network of actors as a graph.
aggregate_rdf	Wrapper	Aggregates RDF resources.
aggregate_xml	Wrapper	Aggregates XML resources.
ant-build-visualisation	Component	
convert_rss	Component and wrapper	Converts RSS feeds to specific RSS version format.
create_audioscrobbler_foaf	Wrapper	Creates a FOAF description of Audioscrobbler data.
filecopier	Wrapper	Copies files in local directory hierarchy.
filefetcher	Component	Fetching resources through cookie authentication.
foaf_creator	Component and wrapper	Transforms FactsAbout data as FOAF.
foaf-o-matic	Component	
get_moinmoin_data	Component and wrapper	Fetches MoinMoin wiki contents a local repository.
get_URL	Wrapper	Fetches resources with basic authentication mechanism based on URI.
get_wiki_attachments	Wrapper	Fetches MoinMoin wiki attachments.
google_maps	Wrapper	Creates a geographic map.
graph2svg	Wrapper	Creates SVG graphs from XML.
graphl	Wrapper	Displays RDF as a graph.
graphshow	Component and wrapper	Displays GraphML as a graph.
histogram2graph	Component and wrapper	Transforms SOM histogram to XML.
interestcommunity	Component and wrapper	Describes a network of actors and their interests as a graph.
local-browser	Wrapper	Launches a local web browser.
matrix2mathml	Wrapper	Creates a MathML matrix from Octave save files.
pagerank	Component	PageRank algorithm implementation with Octave.
prefuse-beta-treeview	Wrapper	TreeML viewer.
rdfxml2tree	Component and wrapper	Converts RDF/XML of SPARQL results to XML-tree.
run_octave	Wrapper	Executes an Octave (MatLab) script.
run_xquery	Wrapper	Executes an XQuery.
run_xslt	Wrapper	Executes an XSL transformation.
simile_timeline	Wrapper	Displays a timeline.
somviz	Component and	Self-organising map library.

	wrapper	
sparql_query	Wrapper	SPARQL query engine.
sparql2vector	Component	
sparqlxml2graph	Component and wrapper	Transforms SPARQL query results to graph.
sparqlxml2map	Component	Transforms SPARQL query results to map data.
spqr1lxml2timeline_data	Component	Transforms SPARQL query results to timeline data.
squiggle	Wrapper	Displays SVG.
swdesk2	Component and wrapper	Outputs RDF/XML, GraphML, and XML serialisations of SPARQL SELECT queries.
texttools	Component and wrapper	Tools for textual data mining.
tpl_engine	Wrapper	HTML templating engine.
validate_xml	Wrapper	XML validator.
vizster	Component and wrapper	Community visualisation player.
webdav_publisher	Wrapper	Publishes files to remote WebDAV repository.
webstart_creator	Wrapper	Creates a Java Web Start package.
zgrviewer	Wrapper	Visualises SVG or GraphViz graphs.
zgrviewer_applet	Wrapper	Visualises SVG or GraphViz graphs as Java applet.

Appendix H: Current requirement status

Name	Status
Enable animated visualisations	This is a component specific requirement. Can be fulfilled with suitable component integration (e.g. with SVG).
Changing visualisation properties	This is a component specific requirement. Components supporting this are integrated.
Collaborative visualisations	Supported.
Enable data modification through visualisation	This is a component specific requirement. Supported via adding links to authoring tools to visualisation.
Hyperlinking in visualisations	This is a component specific requirement. Components supporting this are integrated.
Inclusion of external data sources	This is a component specific requirement. Components supporting this are integrated.
Launching applications from visualisation	This is a component specific requirement. Components supporting this for certain types of (web) applications are integrated.
Linking visualisations	This is a component specific requirement. Components supporting this for certain types of (web) applications are integrated.
Managing accounts	Supported partially. Dependent on the OKS wide account management.
Multiple visualisation clients	Supported.
Saving visualisations	Supported.

Selecting visualisation data	This is a component specific requirement. Components supporting this are integrated.
Sharing visualisations	Supported.
Support for multiple source data systems	This is a component specific requirement. Components supporting this are integrated.
Support for user profiles	Supported partially. Dependent on the OKS wide user profiles.
Support multiple visualisation models	Supported.
Utilising source system metadata	This is a component specific requirement. Components supporting this are integrated.
Extending visualisation environment	Supported.
GUI for visualisation viewing	Supported.