A Web-Based Interface to Design Information Visualization

Romain Vuillemot, Béatrice Rumpler
Université de Lyon, CNRS
INSA-Lyon, LIRIS, UMR5205
F-69621, France
romain.vuillemot@insa-lyon.fr

ABSTRACT

Information Visualization is a challenging field, enabling a better use of humans' visual and cognitive system, to make sense of very large datasets. This paper aims at improving the current Information Visualizations design workflow, by enabling a better cooperation among *programmers*, *designers* and *users*, in a *one-to-one* and *community* oriented fashion. Our contribution is a web-based interface, to create visualization flows that can be edited and shared, between actors within communities. We detail a real case study where *programmers*, *designers* and *users* successfully worked together to quickly design and improve an interactive image visualization interface, based on images similarities.

Categories and Subject Descriptors

H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Web-based interaction*; D.1.7 [**Software**]: Programming Techniques—*Visual Programming*

General Terms

Visual Programming, Information Visualization, Service Oriented Architecture

Keywords

Information Visualization, Web Services

1. INTRODUCTION

Information Visualization is a challenging field, enabling a better use of humans' visual and cognitive system, to make sense of very large datasets. The Information Visualization design process involves *every single* data transformations steps, from raw data, up to images. Those steps can be considered as a data flow, and are said to form a *pipeline* [5]. This pipeline gives a graphical view to better understand transformation steps and classify them according to their functions or data structures. It can also assist programmers in their early development steps [8]. The pipeline's output can be images (or other media type) that are included in interactive environments to let users zoom, rotate, or filter, to better perform an analytical process [26]. Thus, users are given new insights on data to assist them in solving complex tasks, such as pattern detection or monitoring. Those tasks solving process could not have been done otherwise, such as with regular tools, and in a reasonable time span and cognitive load.

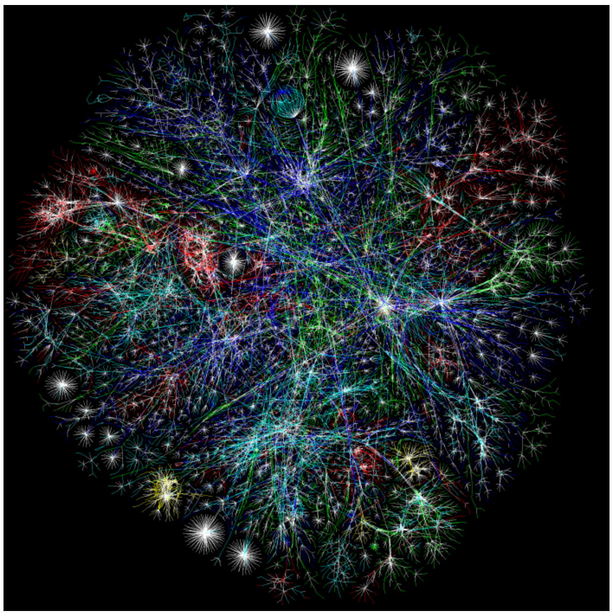


Figure 1: A graph visualization issued from the Opte Project [17], a tentative cartography of the Internet. This is an example of a successful indirect collaboration between actors in computer networks and bioinformatics. In this paper we introduce an interface to assist and reproduce this successful cooperation.

Our work started by looking closely at the Opte Project [17]. Opte was started in October 2003 by Barrett Lyon, and aimed at making a visual representation of the Internet, to detect distributed denial of service attack (DDoS). The underlying dataset is a connected graph, made of network routes over millions of IP addresses. Routes are issued from a traceroute program that determines the route taken by packets over a IP network, and results in a connected graph. A flagship visual result is reproduced figure 1, using a forcedirected graph layout which is the Large Graph Layout [1] library, originally used to make the visualization and exploration of large biological networks. Image on figure 1 is dedicated to analysts to detect patterns and get a better understanding of the Internet structure.

Successful results were also obtained in other application fields, such as in astronomy, advanced network analysis or even artistic purpose. Regarding that last application field, it surprisingly triggered many metaphysical question. Quoting the project's author *Today the image has been used free of charge across the globe and is part of the permanent collection at The Museum of Modern Art (MoMA) and the Boston Museum of Science. It has been used in countless books, media, and even movies* [18].

But from a scientific point of view, the project shortcomings are multiples. For *programmers*, scripts and programs that generate the visualization lack of software optimizations, to scale up to million of edges. *Designers* require more design alternatives and comparisons to understand choices made in the visualization. Finally, *users* need labels and captions to understand nodes and edges meanings. Also, an interactive environment is needed to smoothly zoom in, rotate, and get details on-demand [24] from the dataset. While open-source software were used, there is no easy way to reproduce experiments with the same conditions (such as using the very exact color mapping), especially for people with no technical knowledge. Nonetheless, we think that indirect collaboration between all the actors led to an innovation that just needs a better technical support to be applied to other applications.

In this paper, we introduce an interface to reproduce and generalize the Opte Project to other fields of applications. Our global approach [28] is to consider the Information Visualization model as a data flow, where transformation steps are separated and published across a network through a Uniform Resource Location (URL). This is a simple way of using Web Services that has already been proven efficient [30], contrary to other approaches [32] using SOAP messages, which are complex and sensitive to proxies. Our focus is on the visualization part (and not on the interaction one), which then require to be coupled with an interactive applications controller (to select or filter data). The outline of the paper is as follow: 1) we identify actors (programmers, designers and users) involved in designing Information Visualization interfaces and how they interact between each others (in a one-to-one and community fashion) 2) we introduce *mashviz*, a web-based interface allowing those actors to cooperate and 3) we describe a real-case scenario where those actors successfully used mashviz to quickly design and improve an interactive image visualization interface, based on images similarities. Finally we discuss our results and give our future works.

2. ACTORS AND INTERACTIONS

It is widely acknowledged that interfaces design must be done by pluri-disciplinary work [21, 7]. Nonetheless, actors are currently scattered all over many communities, locations, and their roles and interactions have not been clearly defined, yet. Furthermore, as we mentioned in describing the Opte Project, the Internet fosters new opportunities due to its open and decentralized structure. Also, users tend to become active actors too, as long as they are given an efficient support of sharing and evaluation.

In this section we identify three main actors: *programmers*, *designers* and *users*, and then we focus on two common way they use to exchange information and knowledge: *one-to-one* and within *communities*. Note that other actors and interactions exist, according to usage, culture or context: we will discuss them in section 5.

2.1 Actors

Actors involved in the Information Visualization design workflow are numerous, since the data transformation flow is long, complex, and pluri-discplinary [21]. We focus on three actors that are in the heart of the process:

Programmers: they are individuals who write, test, debug, and maintain computer software. They use programming languages to encode users' need, by means of functions and logical operations that are transformed by compilers to machine understandable binary code. Programmers' outputs are mainly programs, that are executable on a client side (Executable binary) or remotely (Web Services). Programs can be meant to be reusable such as API (Application Programming Interface) or libraries. Also programmers can express recommendations by means of book or design patterns.

Designers: they are individuals who pick up the best solutions among existing ones. Designers' outputs are less formal than programmers, since they may have a less scientific way of work. Outputs are choices among existing architectures and their know-how is not easy to communicate and share, since they are based on experiences and habits. Knowledge sharing is organized in books and best practices exchanges within groups.

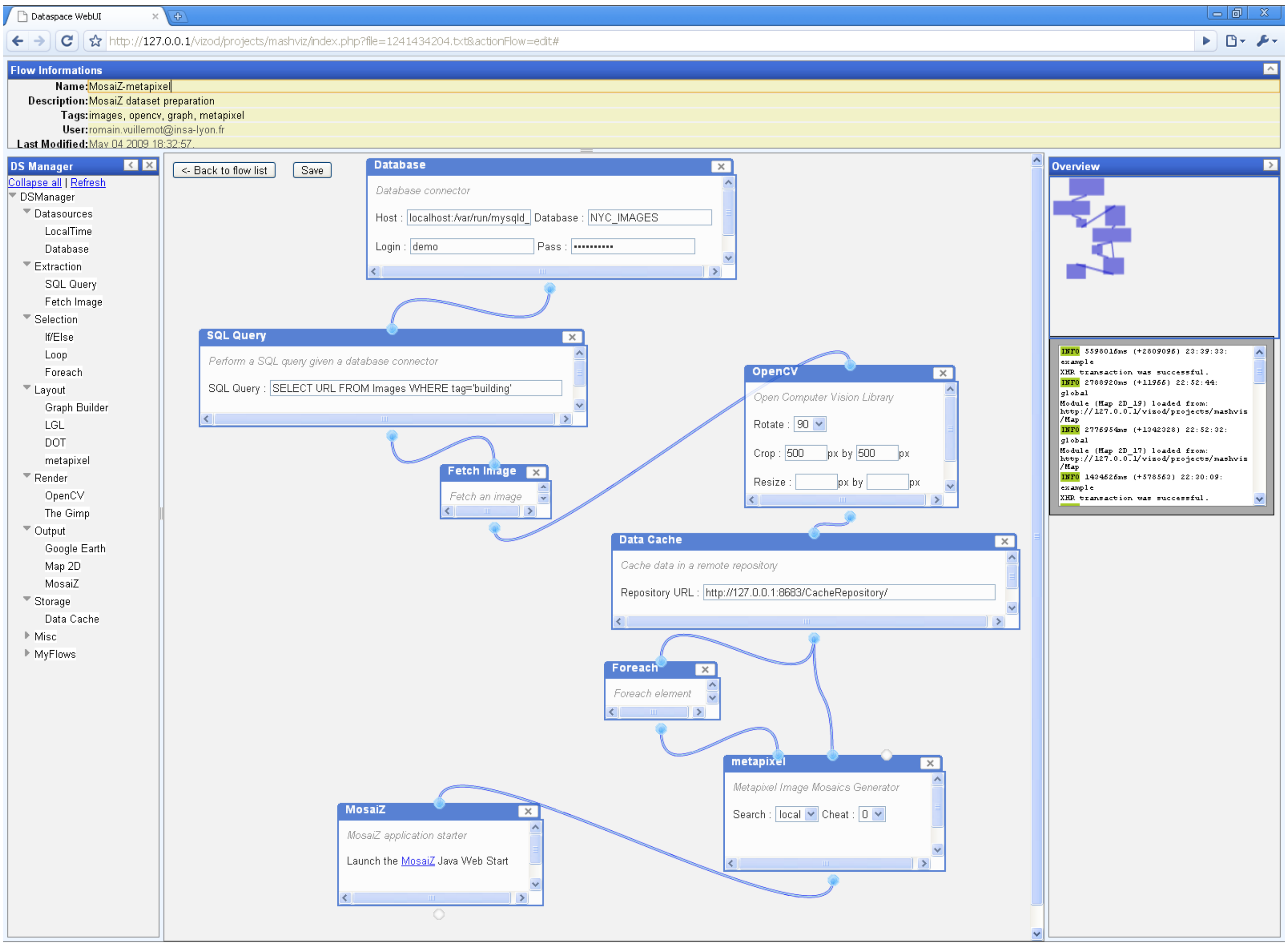
Users: they are individuals who use a machine without complete computer expertise. Users want to achieve a task that could not have been done manually or in a limited amount of time and cognitive load. Users are broad in age and skills, but can be segmented as follow: the most active ones are innovators and early adopters (power users), who will quickly adopt new systems, even at a beta stage. At the opposite laggards are reluctant to use such systems, especially if they have not been proven stables. In between those two extremes, the vast majority (about 80%) user computers with an average learning curve. All users will produce usage statistics and can participate to the system evaluation in a direct (forums, comments) and indirect (usage count, activity logs) way.

2.2 Interactions

Interactions are active behaviors that actors engage to communicate and exchange information. We focus on two interactions types:

One to one: is the simplest way to exchange information between two single individuals, through a sing communication channel. It can be synchronized or not, and using different medium types but it has to share a same communication vocabulary or signs.

Community: is a way to gather individual based on location, organization, faith or believes. It has been proper to humans to build those groups and they tend to be



http://127.0.0.1/vizod/projects/mashviz/index.php?file=1241434204.txt&actionFlow=edit#

Name: MosaiZ-metapixel
Description: MosaiZ dataset preparation
Tags: images, opencv, graph, metapixel
User: romain.vuillemot@insa-lyon.fr
Last Modified: May 04 2009 18:32:57

Database — Database connector
Host : localhost:/var/run/mysqld Database : NYC_IMAGES
Login : demo Pass : ••••••••••

SQL Query — Perform a SQL query given a database connector
SQL Query : SELECT URL FROM Images WHERE tag='building'

Fetch Image — Fetch an image

OpenCV — Open Computer Vision Library
Rotate : 90
Crop : 500 px by 500 px
Resize : px by px

Data Cache — Cache data in a remote repository
Repository URL : http://127.0.0.1:8683/CacheRepository/

Foreach — Foreach element

metapixel — Metapixel Image Mosaics Generator
Search : local Cheat : 0

MosaiZ — MosaiZ application starter
Launch the MosaiZ Java Web Start

```
INFO 5598016ms (+2809096) 23:39:33:
example
XHR transaction was successful.
INFO 2788920ms (+11966) 22:52:44:
global
Module (Map 2D_19) loaded from:
http://127.0.0.1/vizod/projects/mashviz
/Map
INFO 2776954ms (+1342328) 22:52:32:
global
Module (Map 2D_17) loaded from:
http://127.0.0.1/vizod/projects/mashviz
/Map
INFO 1434626ms (+578563) 22:30:09:
example
XHR transaction was successful.
```

Figure 2: *masvhiz* flow creation interface. Middle layout shows composition space (where boxes are linked together to compose a flow). The top strip gathers information about currently edited flow. The left vertical strip shows the service list organized in categories. Finally, right vertical strip shows flows overview and informative logs (errors, warning, etc.).

more and more virtual with broadly available digital device and Internet access. Communities also tend to be decentralized and let anyone know everyone about their production, regardless time or location.

Note that even if we limited our scope to a few actors and interaction schemes only, it is already heavy to support each actor's communications since the combinatory explodes.

2.3 Supporting Actors Interactions

Our focus is now to assist actors' interactions by means of a technical structure. A structure (we will use the term *platform*) helps to uniform communications and workflow processes, to make actors working in a cooperative way. Many collaborative platforms have spread in recent years, and have been proven efficient coupled with a visual language when data can be modeled as flows [15]. Visual languages aimed at reducing the complexity on the client side, to leave every technical and low level operations on the application or server side.

For instance, in the signal processing field, Analog Box [12] is a modular sound synthesizer that lets users to produce complex sounds, without any specific signal processing background. Similarly, advanced image analysis processing can be done with a few mouse click, using a visual language such as VIVA [16]. In the data mining field, V4Miner [6] provides EJB (Enterprise JavaBeans) which assembly produces visualizations that can assist users to solve complex tasks. Also, Knime [3] is an application based in Eclipse to create data flows on which to perform data mining tasks.

Nonetheless, those interfaces are only aimed at programmers or designers, since they -among other many technical issues- require client installation and configuration. Access to regular users is limited, while they could review data flows and could be included in a feedback loop. Also, those interfaces don't provide a global share of usages (saved files, templates and statistics) which could be another useful feedback, assuming a critical mass of users exist. Finally, as far as we know, none have been dedicated to Information Visualization.

3. MASHVIZ INTERFACE

Our contribution is called *masvhiz*[1], a web-based interface that allows *programmers*, *designers* and *users* to collaborate in an asynchronous way, either *one-to-one* or within *communities*, in order to design Information Visualization. Mashviz aims at making ordinary "knowledge buffers" (recommendations, programs) fully reusable, regardless programming language and executing environment, by making them available as web-services, on a unique platform. The figure 3 shows the global workflow of mashviz.

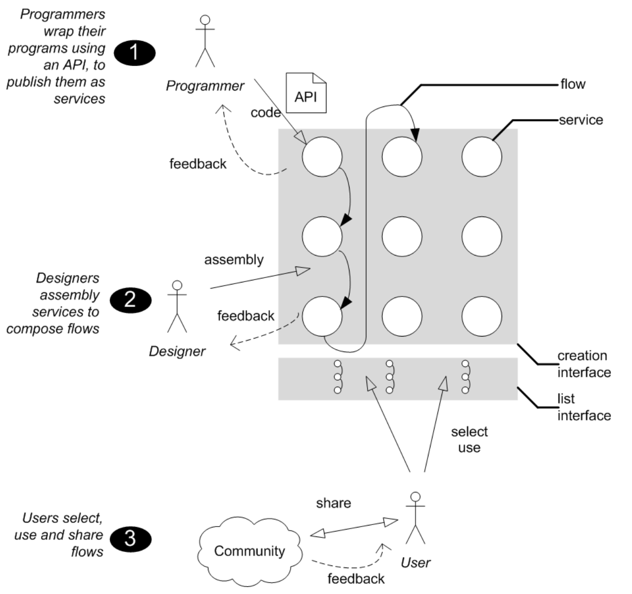

Programmers wrap their programs using an API, to publish them as services
1
Programmer
API
code
feedback
flow
service
assembly
Designers assembly services to compose flows
2
Designer
feedback
creation interface
list interface
select use
Users select, use and share flows
3
Community
share
User
feedback

Figure 3: Masvhiz workflow makes actors to cooperate in an asynchronous way: 1) *programmers* wrap their codes using an API [23] to publish them as web services, 2) *designers* assembly services to compose flows, and 3) *users* select, use and share flows. Feedback is always available using a monitoring interface (not shown here).

The design workflows is described as follow, and is threefold:

1. **Service wrapping** for programmers
2. **Flow composition** for designers
3. **Flow listing** and usage for users

3.1 Service wrapping

Programmers use a dedicated API [23], to wrap programs (i.e. every transformation steps) and make them available as web services. Then, only the interface (inputs and outputs) become visible. Thus, there remain no implementation and execution constraints anymore: only services IP, ports and inputs/outputs are required to use them. Technically, the API is turning programs into stand alone web servers, which read inputs and produces outputs. Off-the-shelf libraries or customized scripts can be wrapped to become web services. According to feedbacks we got from programmers during our case study in section 4, it remains easy and quick to perform. Here are the main primitives of the API:

[1]`http://vizod.liris.cnrs.fr/projects/mashviz/`

- `CheckInput`: checks current service input compatibly with another service.
- `SetInput`: sets the output stream from a service, as input stream for the current service.
- `GetCommands`: gets available commands from the current service.
- `GetOutputPush`: gets the current service output values in a continuous stream.

The API enables flows composition (i.e. connecting services to each others) and performs complex operation such as aggregation, and synchronizations. Once wrapped, services will be categorized by programmers using the pipeline steps [5] and published (visible on the left vertical strip on figure 5).

3.2 Flow Composition Interface

Designers compose flows based on services, using mashviz flow creation interface (figure 2). The interface implements *syntactic* and *semantic* rules [19]. Those rules are invisible for designers: all the rule checking complexity is implemented within the application [15]. Here are details on the two sets of rules:

Syntactic rules: hold low level compatibility checking between services. For instance, service A output can only be connected to service B input, if their types match. In the current implementation we used `MIME` (Multipurpose Internet Mail Extensions) inspired types. To check if types are compatible the web-based interface triggers a `CheckInput` call on both services.

Semantic rules: hold higher checking levels, such as if the flow features navigation history [31], consistency [25] and any global constraint. For instance, Quality of Service (i.e. achieving a certain level of quality) is implemented to guaranty a flow execution time or reliability.

To compose a flow, designers select, drag and drop services (Figure 5) that have been previously published by programmers. Then customize services (i.e. boxes) using regular lists, text fields and check boxes. Services can be connected to each other if they follow both *syntactic* and *semantic* rules.

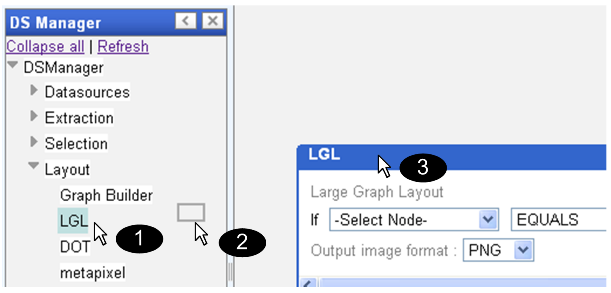
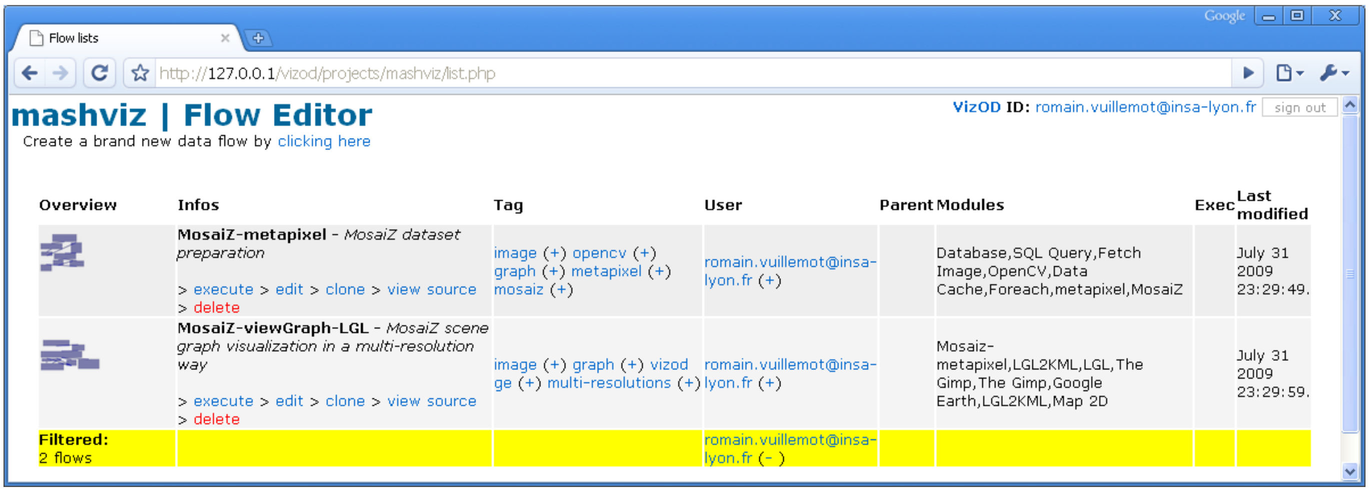


Figure 5: Designers (1) select services from the DS Manager, (2) then drag it to the data space (3) drop and custom it.

In case a new service is published, the DS Manager (datasource manager that provides the list of available services,


Flow lists

http://127.0.0.1/vizod/projects/mashviz/list.php

mashviz | Flow Editor

Create a brand new data flow by clicking here

VizOD ID: romain.vuillemot@insa-lyon.fr sign out

Overview	Infos	Tag	User	Parent	Modules	Exec	Last modified
	MosaiZ-metapixel - *MosaiZ dataset preparation* > execute > edit > clone > view source > delete	image (+) opencv (+) graph (+) metapixel (+) mosaiz (+)	romain.vuillemot@insa-lyon.fr (+)		Database,SQL Query,Fetch Image,OpenCV,Data Cache,Foreach,metapixel,MosaiZ		July 31 2009 23:29:49.
	MosaiZ-viewGraph-LGL - *MosaiZ scene graph visualization in a multi-resolution way* > execute > edit > clone > view source > delete	image (+) graph (+) vizodge (+) multi-resolutions (+)	romain.vuillemot@insa-lyon.fr (+)		Mosaiz-metapixel,LGL2KML,LGL,The Gimp,The Gimp,Google Earth,LGL2KML,Map 2D		July 31 2009 23:29:59.
Filtered: 2 flows			romain.vuillemot@insa-lyon.fr (-)				

Figure 4: *mashviz* flow list interface shows already created flows, which users can edit or clone. User selection can be performed to filter the flow list by users, tags, module used, count of use, etc.

as a tree) is refreshed and designers are notified. Designers can save and load flows, with annotations and tags that will appear in the flow list interface (Figure 4).

3.3 Flow List Interface

Users can use existing flows that appears in the mashviz flow list interface (figure 4). Information on flows, such as textual description or overview pictures, are provided to give users information about the role and complexity of the flow. Users can edit or clone existing flows, already composed by designers. Flows can be used in interactive applications by means of the provided URL by clicking on the `execute` link. Users are identified, so their activity is recorded to provide statistics to both programmers and designers. Also, a programmer can filter flows to the one using his services only, to get usages being made with it by the users community. A monitoring interface (not shown here) gives more details about run counts, execution time and error rates of services. At every steps raw data are available, to let programmers, designers and users to export them and use third party applications to analyze and communicate them more deeply.

3.4 Implementation notes

We implemented mashviz as a web based interface, requiring zero install on the client side. The look and feel is based on Yahoo! Pipes [9], which is a visual programming interface to perform web service mashups. We used Yahoo! User Interface Library (YUI) [11] and WireIT [10] making the interface consistent across web browsers. As back end, we used Apache and a MySql database to store flows and usage statistics, and allow users to perform queries to filter and retrieve flows. The API [23] to wrap programs into web service is written in C# but it can easily be ported into other languages.

4. CASE STUDY

We are now interested in using mashviz to improve an existing interactive application (described in section 4.1). We show how mashviz successfully resulted in an asynchronous work between programmers who implemented the whole data preparation flow as services (section 4.2), designers who created and colored a graph (section 4.3), and users testing the MosaiZ application and improving it by selecting other flows (section 4.4).

4.1 Scenario description

We previously worked on Interactive Image Mosaics[2] [27], which are Image Mosaics (regular images composed from image miniatures) made pan-able and zoom-able, and called *MosaiZ*, to explore large non-structured image collections. Navigation in a MosaiZ allows users to smoothly jump from a *source* image, to reach a *target* image based on a similarity criteria. This criteria is implemented in a mosaic generator [13], which integrates many parameters such as gradient, colors, dataset repartition (e.g. not putting the same images too close). A MosaiZ can be seen as a graph connecting images. This graph is called a scene graph (G_{scene}) which edges are target images, linked to source images, that will become target if selected by users. Building G_{scene} is the crucial part, and the MosaiZ experience can vary grandly according to the way images are extracted and the mosaic generator parameters set: a way to let users to find the best trade off themselves is required, since the options combinatory explodes, and as every visual interface it can't be done automated since users have to evaluate the resulting MosaiZ themselves.

4.2 Dataset preparation

The figure 2 shows a fully complete data preparation flow. The first step is to extract and select images. Images are stored in a SQL database and extracted from via a SQL query (selecting images with tag "building"). Each result is an URL pointing to the image. Then, for each image url, a fetch service retrieves images and provides it to the OpenCV [4] service, which performs resize and rotation. Since original images are modified, and since they will be called multiple times, a temporary cache will store the modified images. The temporary cache is located on a remote server, so when the flow will be executed, the client doesn't need to have

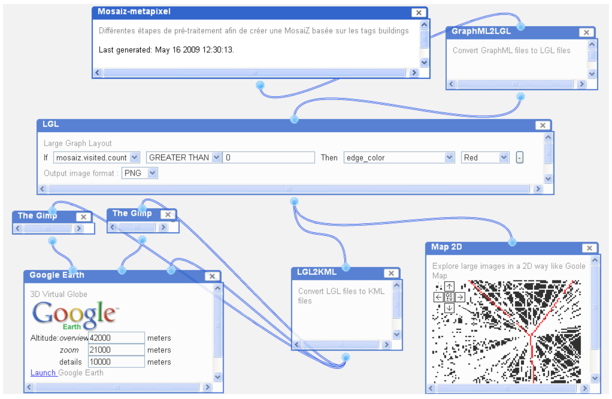
[2] `http://vizod.liris.cnrs.fr/projects/mosaiz/`

any storage facility.

The second preparation step is to extract image characteristics and generate a scene graph. We used the metapixel library [13] as a mosaic generator, which for each images as input, produces an image mosaic as output, based on images features. The metapixel library output is a URL pointing to the graph (GraphML encoded) which holds the scene graph structure and refers to images stored in the cache. This URL is the input of the MosaiZ Zoomable User Interface (based on the Java library Piccolo2D [2]) that can be launched as a Java Web start by users.

4.3 Scene Graph visualization

While the scene graph is originally internal to the Java application, to connect images and construct a space/scale diagram, it could be visualized to give users an overview of the dataset: a graph visualization process must be set up.


Mosaiz-metapixel
Différentes étapes de pré-traitement afin de créer une MosaiZ basée sur les tags buildings
Last generated: May 16 2009 12:30:13.

GraphML2LGL
Convert GraphML files to LGL files

LGL
Large Graph Layout
If mosaiz.visited.count GREATER THAN 0 Then edge_color Red
Output image format : PNG

The Gimp

The Gimp

Google Earth
3D Virtual Globe
Altitude: overview 42000 meters
zoom 21000 meters
details 10000 meters
Launch Google Earth

LGL2KML
Convert LGL files to KML files

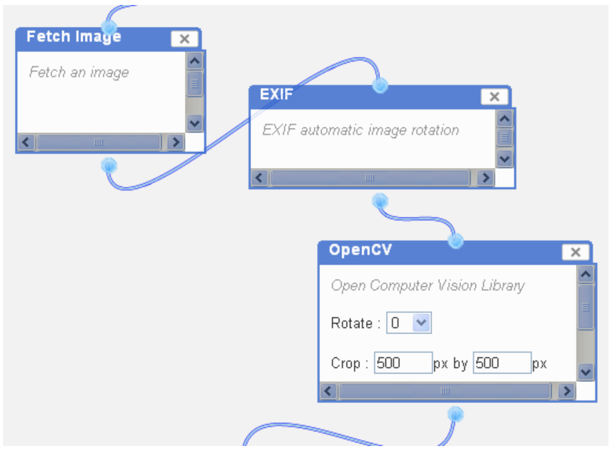
Map 2D
Explore large images in a 2D way like Goole Map

Figure 6: Graph building and rules to color and visualize *MosaiZ* scene graph. The graph that was initially internal to a Java application and is now visualized using LGL [1]. The visualization can be plugged into a 3D multi-resolution environment using Google Earth [29] or a 2D interface (similar to Google Map).

To visualize the scene graph in a meaningful way, designers composed a scene graph visualization flow (figure 6) by including an already wrapped graph layout service, by programmers. Designers used LGL [1], which is the very same library used for the Opte project and which is an quick auto-organizing graph layout. The service can be customized to color the graph. The red color has been associated to the count number: a graph vertex will be red if visited at least once, by users in the MosaiZ application. The LGL output is an image that needs an interactive environment to zoom and pan. We connected it to two modules. The first one is making a KML (Keyhole Markup Language) map that can be executed by Google Earth [29]. The image resolution will change regarding user's altitude on the virtual globe: the higher his altitude is, the more abstract the image will be by making an edge detection to highlight main parts of the graph using The Gimp [14]. Then a Gaussian blur will be added the more the user zooms in. Finally, on the ground the original image is displayed to show details. Note that Google Earth won't show any geographical information, only data from our graph will be shown and filtered through the interface. The second module will display original image as on a regular 2D map such as Google Map, using the very same graph generated at the ground level on Google Earth.

4.4 New service required by users

We already emphasized that the dataset preparation step is crucial, since it will make the MosaiZ relevant or not. During our experiments, we noted that users complained about a substantial amount of images needed to be rotated. The figure 2 shows that in the current dataset preparation, the OpenCV library was automatically rotating 90 degrees clockwise, which leads sometimes to a disappointing result.


Fetch Image
Fetch an image

EXIF
EXIF automatic image rotation

OpenCV
Open Computer Vision Library
Rotate : 0
Crop : 500 px by 500 px

Figure 7: An automatic rotation service is added (based on EXIF metadata), to improve the image dataset preparation flow. If EXIF data are not available, no operation is performed.

So, users decided to use another flow that takes into account EXIF (Exchangeable Image File Format) data (metadata from digital camera) from pictures, to perform an efficient automatic rotation (figure 7). This other flow was picked up because it was more successful in terms of run count. If that flow didn't exist, users could have cloned the basic flow and added an automatic rotation module (assuming that automatic rotation module was implemented by programmers). This operation would have only required a few clicks for users, with absolutely no programming skills. Note that if a clone is made, the link between the two flows will be kept, and users will be able to see the connection. This means that users can pick up better flows just by looking at clones and related flows (based on similar tagging or users). Also, programmers and designers will be notified that a flow has been cloned or a service has been used in a new flow (it can be done since all actors are identified in mashviz). And if the EXIF rotation is still not satisfying, some requests can be made to get a module with automatic rotation (based on content, not on metadata) to the programmers' community. Once a flow is cloned, it is automatically published in the flow list interface, and users can comment and tag it.

5. DISCUSSION AND CONCLUSION

In this paper we introduced a web based interface, mashviz, to support collaboration between programmers, designers and users. We detailed workflows for each actor, and how to apply it to a real case scenario to improve a current interactive application. Regarding our motivating example (the Opte Project) we managed to provide a complete environment that can reproduce it by supporting actors' cooperation in an asynchronous way. Syntactic and semantic rules have been introduced to unload actors from repetitive checking tasks, and focus on more important tasks, such as improving flows design.

We limited our scope of study to three main actors: *programmers*, *designers* and *users*, as well as two interactions types: *one-to-one* and *communities*. But other actors are indirectly involved, such as artists that can give inspirational perspectives through exhibitions or books. For instance, Informative Art [22] aims at integrating data through art in homes, without cognitive overload. If those visualizations were based on mashviz (i.e. every transformation step is published as services and then composed into flows) a broader audience and application range could be reached, fostering innovation.

Regarding our experiments, we noticed that roles are not exclusive. For instance, we noticed after a short training time only, some users were able to fully use mashviz flow creation interface. Also some actors can be "complete" such as *power users* who are also *designers* and *programmers* since they may have multiple abilities.

One of the major shortcoming of the Opte Project was the difficulty to plug it with another dataset. Such a possibility is strongly advised, especially to benchmark a visualization among others [20]. Using mashviz it becomes easy to use any dataset, as long as it matches flows inputs. Working with programmers, we realized that mashviz facilitates *vertical* assembly to fully experiment individual contributions, in a complete existing data flow, up to images and integration to applications. For instance if a new database query system or a graph layout is developed, it can quickly be tested with an existing data flow (after being wrapped using the API). Finally, mashviz allows comparisons with concurrent solutions, which will be an *horizontal* analysis, such as a benchmarks would do.

Our next step is to make mashviz available at a wide scale, as a complete platform with numerous data sources, transformers and logical operators. Also, we plan to include usage statistics directly in both edition and list interfaces (using color coding or symbols). So that designers and users will be aware of execution time (e.g. to identify bottlenecks) to be able to predict flows behavior and pick up the adequate flow regarding the dataset, users involved and the task to achieve.

6. REFERENCES

[1] A. T. Adai, S. V. Date, S. Wieland, and E. M. Marcotte. Lgl: creating a map of protein function with an algorithm for visualizing very large biological networks. *J Mol Biol*, 340(1):179–190, June 2004.

[2] Benjamin B. Bederson, Jesse Grosjean, and Jon Meyer. Toolkit design for interactive structured graphics. *IEEE Trans. Softw. Eng.*, 30(8):535–546, 2004.

[3] M.R. Berthold, N. Cebron, F. Dill, G. Di Fatta, T.R. Gabriel, F. Georg, T. Meinl, P. Ohl, C. Sieb, and B. Wiswedel. KNIME: The Konstanz information miner. In *Proceedings of the 4th annual industrial simulation conference, Workshop on multi-agent systems and simulations, Palermo*. Springer, 2006.

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Ed H. Chi. A taxonomy of visualization techniques using the data state reference model. In *INFOVIS*, pages 69–76, 2000.

[6] Thanh-Nghi Do and Jean-Daniel Fekete. V4miner pour la fouille de données. *numéro spécial de la revue RIA, Revue d'Intelligence Artificielle*, 2008.

[7] J. Forlizzi, J. Zimmerman, and S. Evenson. Crafting a Place for Interaction Design Research in HCI. *Design Issues*, 24(3):19–29, 2008.

[8] Jeffrey Heer, Stuart K. Card, and James A. Landay. prefuse: a toolkit for interactive information visualization. In *CHI*, pages 421–430, 2005.

[9] http://developer.yahoo.com/yui/. Yahoo! Pipes. 2009.

[10] http://javascript.neyric.com/wireit/. WireIt. 2009.

[11] http://pipes.yahoo.com. The Yahoo! User Interface Library (YUI). 2009.

[12] http://www.andyware.com/abox2/. Analog Box 2. 2009.

[13] http://www.complang.tuwien.ac.at/schani/metapixel/. Metapixel - A Photomosaic Generator. 2009.

[14] http://www.gimp.org/. Gnu image manipulation program. 2008.

[15] Wesley M. Johnston, J. R. Paul Hanna, and Richard J. Millar. Advances in dataflow programming languages. *ACM Comput. Surv.*, 36(1):1–34, 2004.

[16] Dennis Koelma and Arnold Smeulders. A visual programming interface for an image processing environment.

[17] B. Lyon. The opte project. *The Opte Project*, 2003.

[18] B. Lyon. Opte as an Aesthetic Experience. *The Opte Project*, 2005.

[19] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, 1986.

[20] Catherine Plaisant, Jean-Daniel Fekete, and Georges Grinstein. Promoting insight-based evaluation of visualizations: From contest to benchmark repository. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):120–134, 2008.

[21] J. Preece, Y. Rogers, and H. Sharp. *Beyond Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons, Inc. New York, NY, USA, 2001.

[22] Johan Redström, Tobias Skog, and Lars Hallnäs. Informative art: using amplified artworks as information displays. In *DARE '00: Proceedings of DARE 2000 on Designing augmented reality environments*, pages 103–114, New York, NY, USA, 2000. ACM.

[23] Marian Scuturici. Dataspace API. Technical report, LIRIS, 2009.

[24] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *VL '96: Proceedings of the 1996 IEEE Symposium on Visual Languages*, page 336, Washington, DC, USA, 1996. IEEE Computer Society.

[25] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[26] James J. Thomas and Kristin A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics.* National Visualization and Analytics Ctr, 2005.

[27] R. Vuillemot and B. Rumpler. MosaiZ: Interactive Image Mosaics. Technical report, LIRIS, 2009.

[28] Romain Vuillemot, Béatrice Rumpler, and Jean-Marie Pinon. *Enterprise Information Systems*, chapter Dissection of a Visualization On-Demand Server, pages 348–360. LNBIP. Springer Berlin Heidelberg, April 2009.

[29] Romain Vuillemot and Béatrice Rumpler. Mapping visualization on-demand onto a virtual globe: an appealing complement to browser-based navigation. In *HT '08*, pages 249–250, New York, NY, USA, 2008. ACM.

[30] Martin Wattenberg, Jesse Kriss, and Matt McKeon. Manyeyes: a site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1121–1128, 2007.

[31] Alan Wexelblat. History-based tools for navigation. In *HICSS*, 1999.

[32] J. Wood, K. Brodlie, J. Seo, D. Duke, and J. Walton. A web services architecture for visualization. In *Proceedings of the IEEE Fourth International Conference on eScience, 2008.*, pages 1–7. IEEE Computer Society Press, 2008.