Fall | 2014

Technical Report

Service Oriented Computing (18-655)

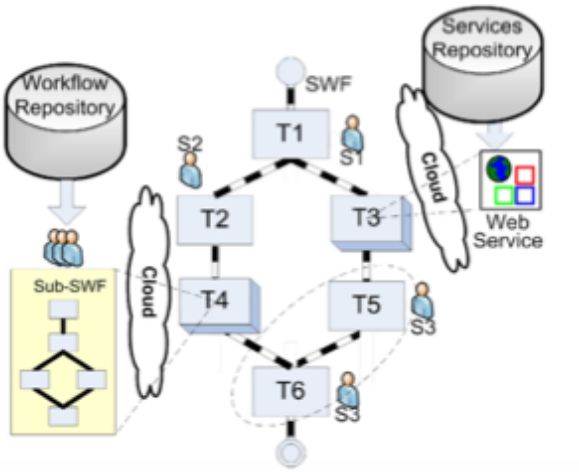
IBM Big Data Team

Enclosed in this document is the technical report of the IBM Big Data project sponsored by Dr. Wei Tan.

Table of Contents

1. Introduction

As illustrated in the following figure, a scientific workflow precisely describes a multistep procedure to streamline a composition of tasks and the dataflow among them.



In these workflows, we can see it has many web services, which help different workflows to accomplish their goals.

In fact, recently emerged services computing technology enables and encourages scientists to expose data and computational resources wrapped as Web services, so that they become publicly available to other researchers through standard interfaces. For example, Biocatalogue has registered over 2,500 life science services. A scientific workflow thus may leverage published Web services as tasks to speed up workflow composition. To facilitate domain scientists in finding available workflows, several domain-specific online repositories have evolved in recent years. For example, MyExperiment stores 2,510 public workflows.

In this project, we aim to analyze and visualize the relationship of workflow and web-service and then build a recommendation engine for scientists who want to build a new workflow.

2. Motivation

To accelerate data-intensive scientific exploration, many disciplines including biology and astronomy have adopted workflows as data-pipeline orchestrators to design scientific applications. As illustrated in the above figure, a scientific workflow precisely describes a multistep procedure to streamline a composition of tasks and the dataflow among them. Through our research work, we try to find the answer of two questions: What implicit knowledge (usage patterns) may be automatically extracted from historical data to help scientists better understand existing artifacts; How to leverage such data to facilitate scientific artifact reuse.

3. Related work

Artifact reuse and recommendation has attracted much attention in the engineering field. VisComplete provides auto-complete suggestions for VisTrails system by mining frequent patterns in existing pipelines. Leake et al. propose a case-based approach to suggest the possible next step(s) aiding re-use of portions of prior workflows. Xiao et al. propose a layered workflow structure that allows users to specify workflows at different levels of abstraction, and a graph matching method to find similar ones.

On the other hand, there is some research work on how to visualize data. It is easier for the user to pick an artifact if they could literally see these graphs and relationship. Harrer et al. transform structured data (e-mail, discussion boards, and bibliography sources) into social network data formats to visualize dynamic communities. Duong et al. study automatic generation and visualization of personal ontology from personal and organizational websites, blogs, and publications. Viégas et al. propose history flow visualization, as an exploratory data analysis tool, to analyze cooperation and conflict of authorships in the wiki context. Vizster system offers an online visualization of social networking, allowing users to discover communities, people and connections. Invenio is a tool for visualizing multi-modal, multi-relational social networks.

Combining artifact reuse and data visualization together, Professor Jia and her team proposed the "Recommend-As-You-Go" approach to support service-oriented scientific workflow reuse in 2011. Their work highlight three parts: first, analyze service usage history; second, build a PSW network: people, service, workflow network; third, get the relationship based on the past usage history.

In our project, we also divide our system into two parts: data collection/analysis and data visualization. We get workflows and web services and their related information. We analyze these workflows and web services based on their usage history. Then we use a Java library JUNG to visualize the results and help us better understand different aspects of the whole network.

4. System design

In order to provide users an efficient and effective way to find an appropriate workflow, we use JUNG (Java Universal Network/Graph Framework) to visualize the relationship between workflows and we services. Before that, we should fetch all data of available workflows and web services and analyze them to get our desired data.

4.1 Data Collection & Analysis

We fetched data set from the web sites and analyze them in these two steps.
Step1: Parse XML file using Biocatalogue and MyExperiment API and get needed data in java.

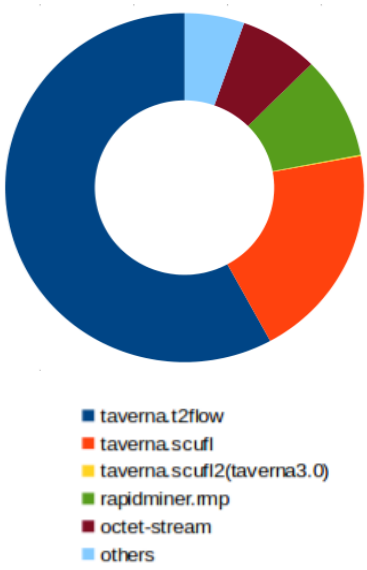
Step2: Analyze the data and obtain the relationship between workflows and web services. Also, get the relationship between different workflows based on the services they called.

4.1.1 Parse XML files

In the parsing step, we used Biocatalogue and MyExperiment public API to get all the raw data of all available workflows and web services. There are many different kinds of workflows that we had to handle them in different methods.

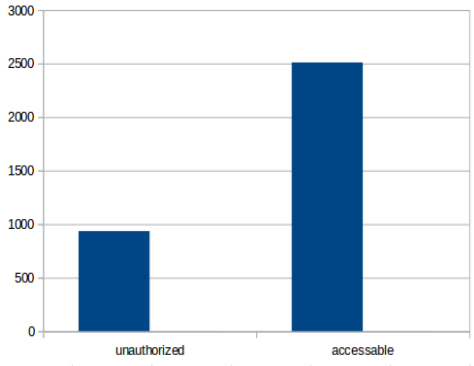
MyExperiment workflows
MyExperiment provides us different types of workflows using different suffixes. This is the category we fetched from MyExperiment workflows.

TYPE	COUNT	PROPORTION
taverna.t2flow	1410	58%
taverna.scufl	501	20%
taverna.scufl2(taverna3.0)	3	00.12%
rapidminer.rmp	232	9.5%
octet-stream	174	7.1%
others	190	7.6%
TOTAL : 2510		



There are some unauthorized workflows that we cannot fetch through the public API, so we didn't include these workflows into our data set. Here is the proportion of the unauthorized workflows.

condition	COUNT	PROPORTION
unauthorized	935	27%
accessable	2510	73%
TOTAL : 3445		



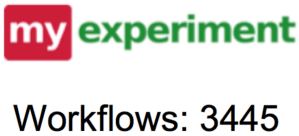
Biocatalogue Web Services

In many data sets, restful apis are dominant. While in scientific domain, soap services are still popular. Scientific services are typically not easy to use – for example, they carry a lot of parameters that need to be understood and properly tuned. Since RESTful APIs lack a way to specify the semantic meanings of such information due to its simplicity, SOAP services can carry significant information in WSDL files.

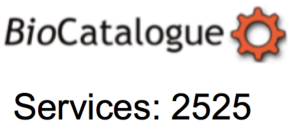
Biocatalogue provides us both REST and SOAP services and we used them in the analysis step. Most of web services are SOAP services, and we identify them using WSDL location. To identify REST services, we used their endpoint.

Data Set

This is our row data set. As you can see in the chart



Workflow Type	public	private
count	2510	935
percentage	72.9%	27.1%



Web Service Type	SOAP	REST
count	2314	212
percentage	91.6%	8.4%

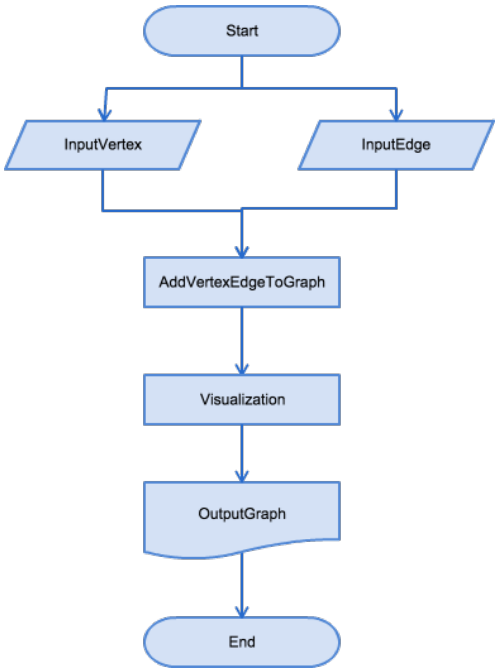
4.1.2 Deal with relationships

Build the relationship between workflows and web services are not difficult. We identified workflows and services with different IDs, and then build the relationship based on the workflows' processes. In one workflow, there are many processes. Some of peocesses would call a web service and it can be parsed from the XML file of the workflow. Here is a segment of one workflow process and you can easily find the web service it calls.

```
<s:processor name="runInterProScan">
    <s:arbitrarywsdl>
        <s:wsdl>http://www.ebi.ac.uk/Tools/webservices/wsdl/WSInterProScan.wsdl</s:wsdl>
        <s:operation>runInterProScan</s:operation>
     </s:arbitrarywsdl>
  </s:processor>
```

When we built the relationship between workflows and web services, we also considered the popularity of one specific service based on how many times it has been called from workflows. Because we could recommend the most popular service to users based on their needs.

Furthermore, we built the relationship between different workflows. If different workflows calling the same web service, they may have some similarities and we may use these similarities in the recommendation engine and we could also build workflow cluster in the future.



4.2 Visualization

We use JUNG (the Java Universal Network/Graph Framework) to visualize the data and results. JUNG is an open source graph modeling and visualization framework written in Java, under the BSD license. The framework comes with a number of layout algorithms built in, as well as analysis algorithms such as graph clustering and metrics for node centrality.

In our project, since we want to build a network of scientific workflows and web services, and then visualize the network, we shall first propose a programming model for the workflows and web services. As shown in the flow chart, there are three main steps in visualization part. First step is retrieving all web services and workflows and their relationship data. Second step is generating the graph with nodes and edges. Last step is visualizing the graph with specific parameters such as defined layout.

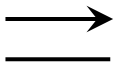
5. System implementation

In system implementation, the most important part is how to visualize the relationship between workflows and web services and represent it in an appropriate way. We use JUNG to do the visualization. First step is to generate the graph. We represent workflows and services as vertices and their relationship as edges. There are two types of web services in our data set, RESTful services and

SOAP services. So altogether, there are three kinds of nodes: workflows, RESTful web services and SOAP web services. In order to differentiate these nodes, we give them different shapes and colors. For a workflow, its shape is a square, and its color is dark yellow; for a SOAP web service, its shape is a circle, and its color is blue; for a RESTful web service, its shape is a rectangle and its color is purple.

Graph: G(V, E) → G(MyNode, MyEdge)
MyNode: Workflow
SOAP Service
REST Service

MyEdge: Directed edge
Undirected edge



The schema of our node class is as follows:
id: String
title: String
popularity(nodes): int

Id is the unique id we add to each node; and we differentiate different kinds of nodes(workflows, RESTful services and SOAP services) via node id.
Title is the title of the workflow or web service. There may not be unique and sometimes can be null.
Popularity is mainly related with web services. The popularity of a web service node represents how many times it is called by workflows in MyExperiment. If a web service is called many times, it is popular, and it becomes bigger in the graph.

The schema of our edge class is as follows:
id: String
title: String
width: int

Id is also the unique identification of the edge. The directed edge is from a workflow to a web service. The undirected edge is from a workflow to a workflow.

```
FRLayout<MyNode, MyLink> layout = new FRLayout<MyNode, MyLink>(g);
layout.setRepulsionMultiplier(0.8);
layout.setAttractionMultiplier(6);
layout.setSize(new Dimension(1000,1000));
```

After creating the graph, we should visualize it. To view a graph with JUNG we need two new concepts: (a) that of a graph layout, and (b) that of a visualization component. In this project, we focused visualization on the layout. JUNG provides many different layout algorithms for positioning the vertices of a graph. The Layout interface provides additional mechanisms to initialize the locations of all vertices in a graph, set the location of a particular vertex, lock or unlock the position of a vertex, etc. We tried three main layout in out graph: KKLayout, FRLayout and SpringLayout. KKLayout based on the Kamada-Kawai algorithm; FRLayout based on the Fruchterman-Reingold algorithm and SpringLayout is a simple force-directed

spring-embedder layout. Finally, we chose FRLayout in visualization part through repeated tests. You could clearly and easily obtain the relationship of each node in the graph.

6. Experiments and analysis

In order to give an overview of all the workflows and web services and their relationship, we build the first class to contain all workflow and web service nodes, whether they have edges with other nodes or not. To be more specific, the first graph contains all the workflows from MyExperiment and all web services from BioCatalogue. So there are many isolated nodes, which is not a very good thing because, for workflows, if the workflow is isolated(it does not call any web services), it means the author of the workflow build everything from scratch; it means he does not leverage any existing work; it means he may spend a lot of time build something already exists.

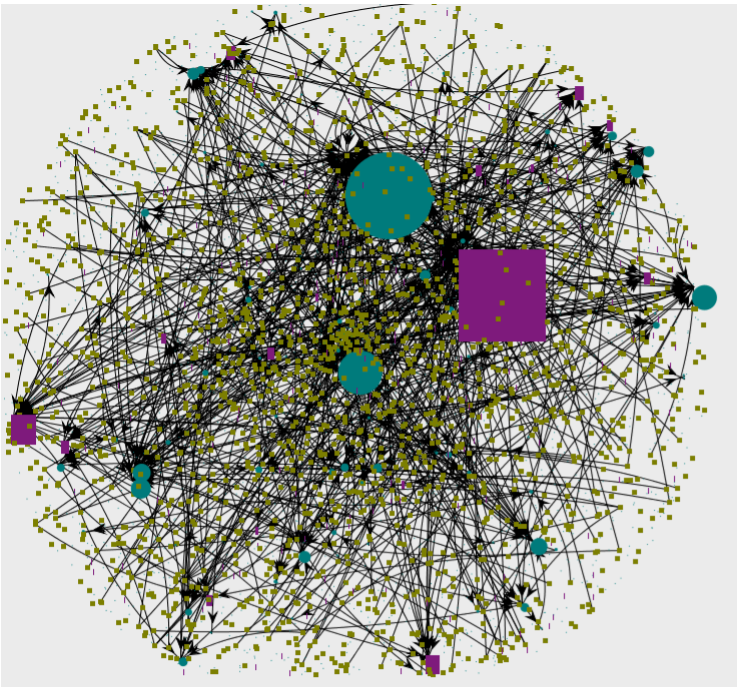


Figure 6.1 All workflow and web services

For web services, if the web service is isolated (not called by any workflow), it means that this web service does not fulfill its purpose: people put a web service in BioCatalogue to let others use them hence saving time; if it has never been used, then it's only a waste of time and resource. Figure 6.1 is the still evolving graph while figure 6.2 is the graph that has become stable. JUNG has different layout options and hence different algorithms to calculate the relationship of nodes and the best position to display the results. The graph will evolve using the algorithm and when it becomes stable, it can reveal more interesting characteristics.

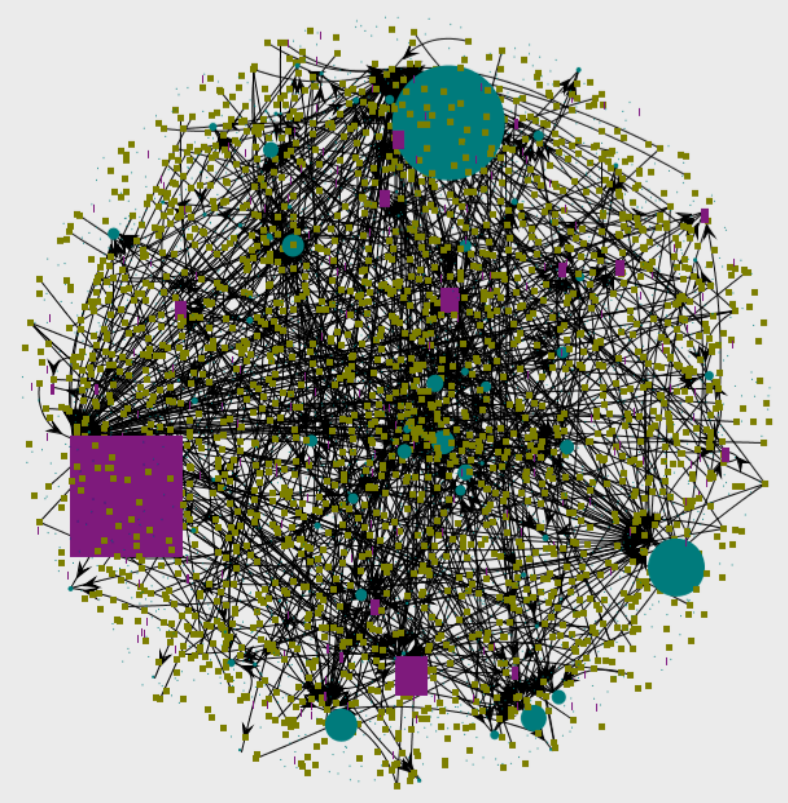


Figure 6.2 All workflows and web services (stable)

As we can see from the picture, there are some nodes that are way bigger, still evolving than other nodes. That means these web services are very popular, and have been called many times, which is a good thing. But the problem is that most of them are very basic web services. They do some basic processing thing. What we like to see is that more customized and complex services are also used by the public, which means scientists are leveraging their colleagues' work. Another thing is that most nodes are of the same size and their size is small. That means scientists still don't really have the habit of leveraging existing services and work. The first graph is a directed graph. The edges are from workflows to services meaning workflows call web services.

Having got a whole feeling of the workflow and service network, we can then go into more details. Since we have SOAP services and REST services, we want to analyze them separately. Also, we can choose other layout options than KKLayout to highlight different aspects of the graph. Figure 6.3 is the workflows to soap service graph, and we apply SpringLayout to this graph. SpringLayout highlights popular nodes. It puts the nodes with more edges in the center of the graph, and other nodes around them. So for users, they can see more clearly the most popular nodes and their calling histories. Usually the most popular nodes are the services that are more reliable and useful.

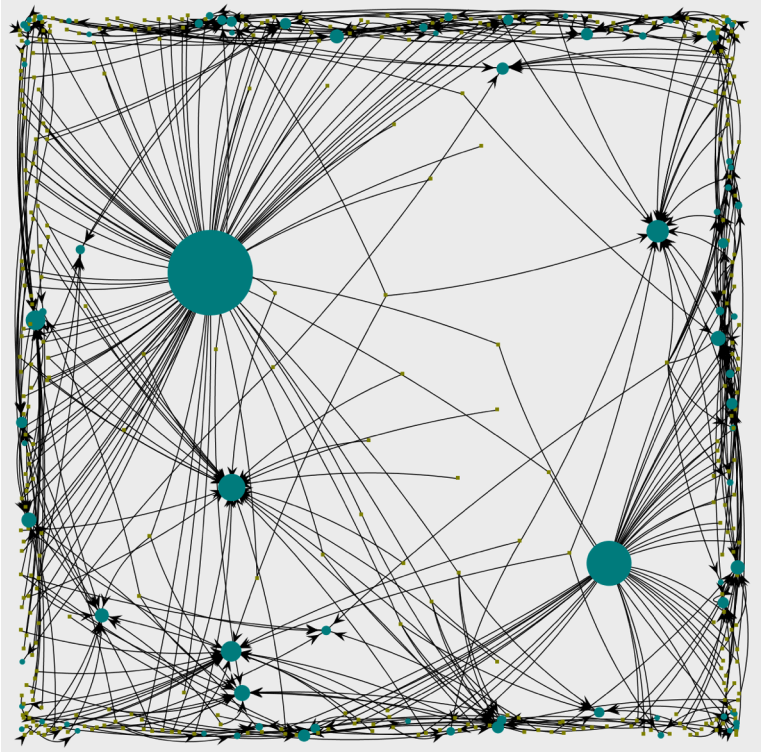


Figure 6.3 Workflows to SOAP service, SpringLayout

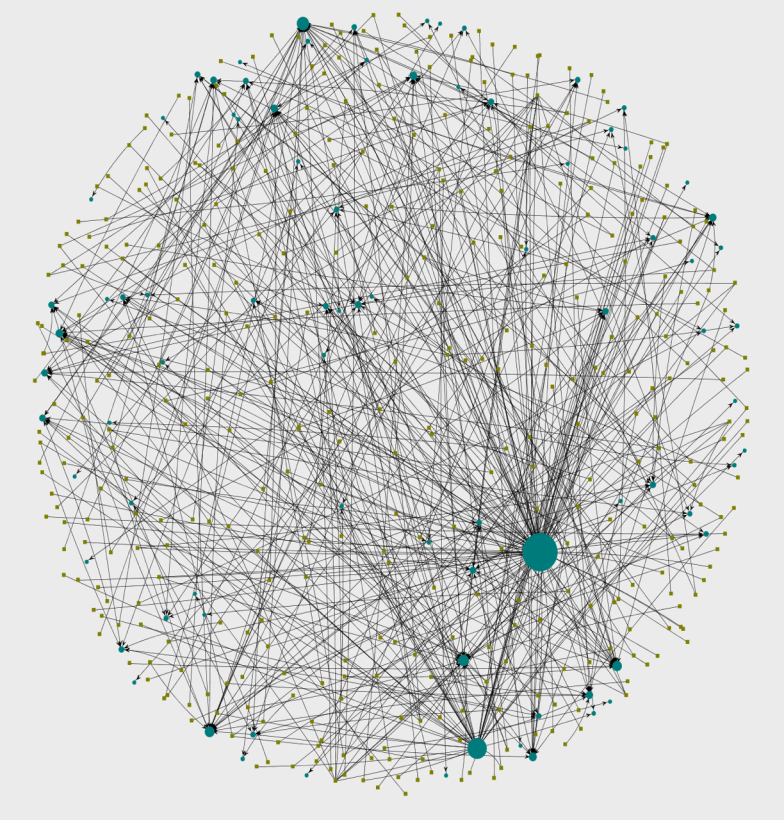


Figure 6.4 Workflows to SOAP services, KKLayout

Figure 6.4 uses KKLayout to visualize the workflows and web services they call. We remove isolated nodes from this graph to make it clearer. Also, to avoid nodes and edges overlapping with each other, we make them smaller so users can have a clearer view. Here we can see the condition of all nodes, whether they are popular or not popular.

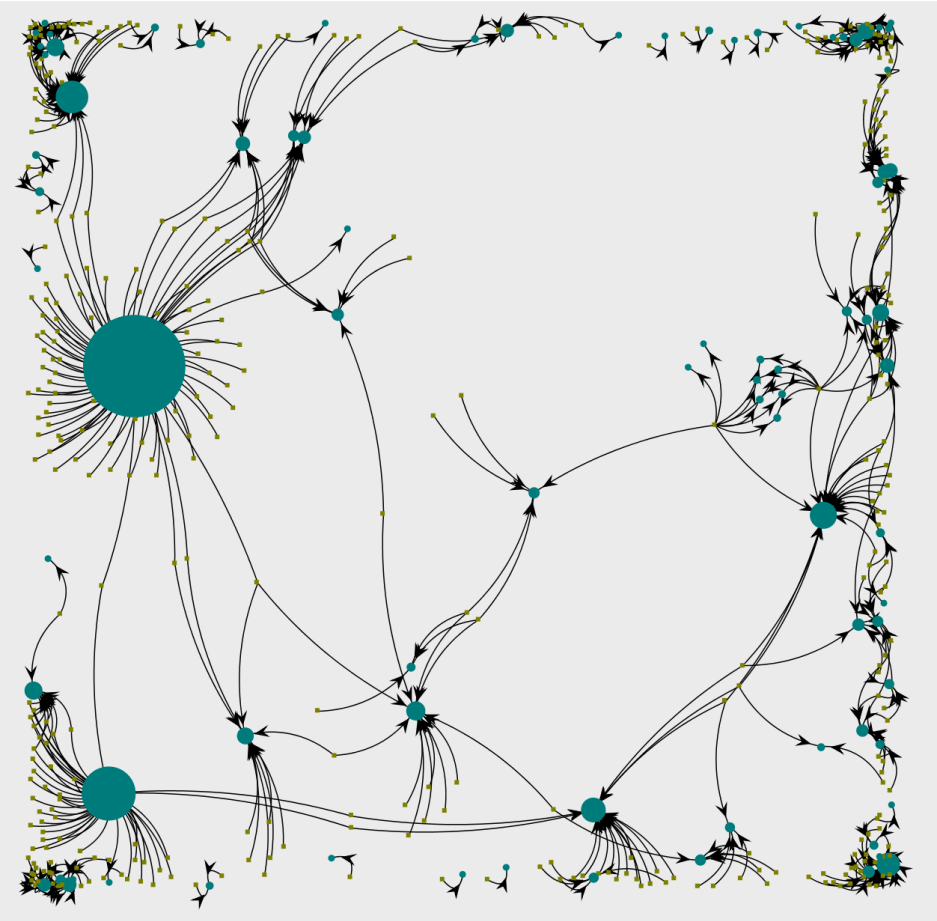


Figure 6.5 Workflows to SOAP services, FRLayout

Now we see that SpringLayout highlights the popular nodes, and KKLayout shows us the structure of the graph. In order to emphasize both the popular nodes and the structure of the graph, we can adopt FRLayout. Figure 6.5 uses FRLayout to show the workflows and SOAP services. We can see clearly from this figure the popular nodes and who called them. Workflows calling the same service are placed around the service. Thus we can build a relationship between workflows using the shared service as a bridge. Workflows calling the same web service are supposed to have some kind of relationship or similarity. User can make use of the services to find related workflows. We have another graph dealing with those related workflows, where removed the services first; but here in this figure we have both the related workflows and the services linked them together.

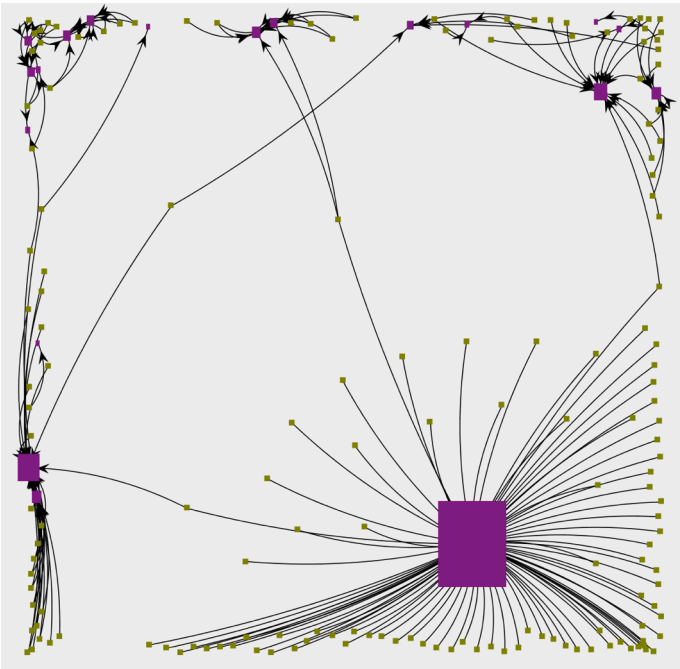


Figure 6.6 Workflows to REST services, FRLayout

Moving to REST services, figure 6.6 uses FRLayout to show the calling situation of REST services. We don't have as many REST services as SOAP services. As in the previous graph, it highlights popular nodes and the structure of the graph. We can also apply KKLayout and SpringLayout to REST services as with SOAP services, but here a FRLayout example would be enough.

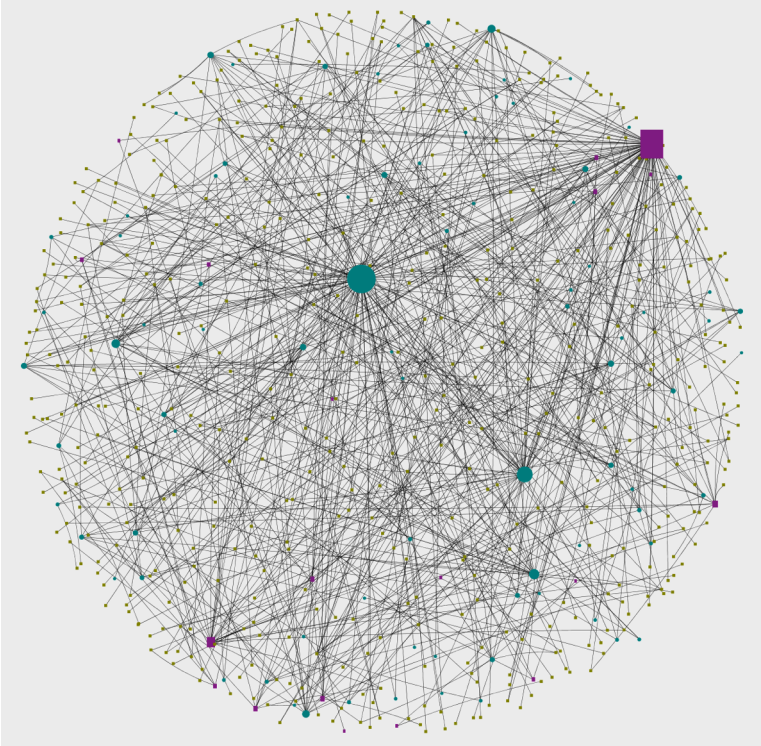


Figure 6.7 Workflows and all types of services, KKLayout

Combining SOAP and REST services together, figure 6.7 uses KKLayout to show three kinds of nodes in different shapes and colors. Figure 6.8 is the FRLayout version. From figure 6.8, we can see that some workflows call both SOAP and REST services. Also, groups of workflows appear in this graph: some workflows call the exact same web services, SOAP or REST. When recommending workflows to users, we can recommend a group of workflows sharing the exact same web services along with the group of web services they call.

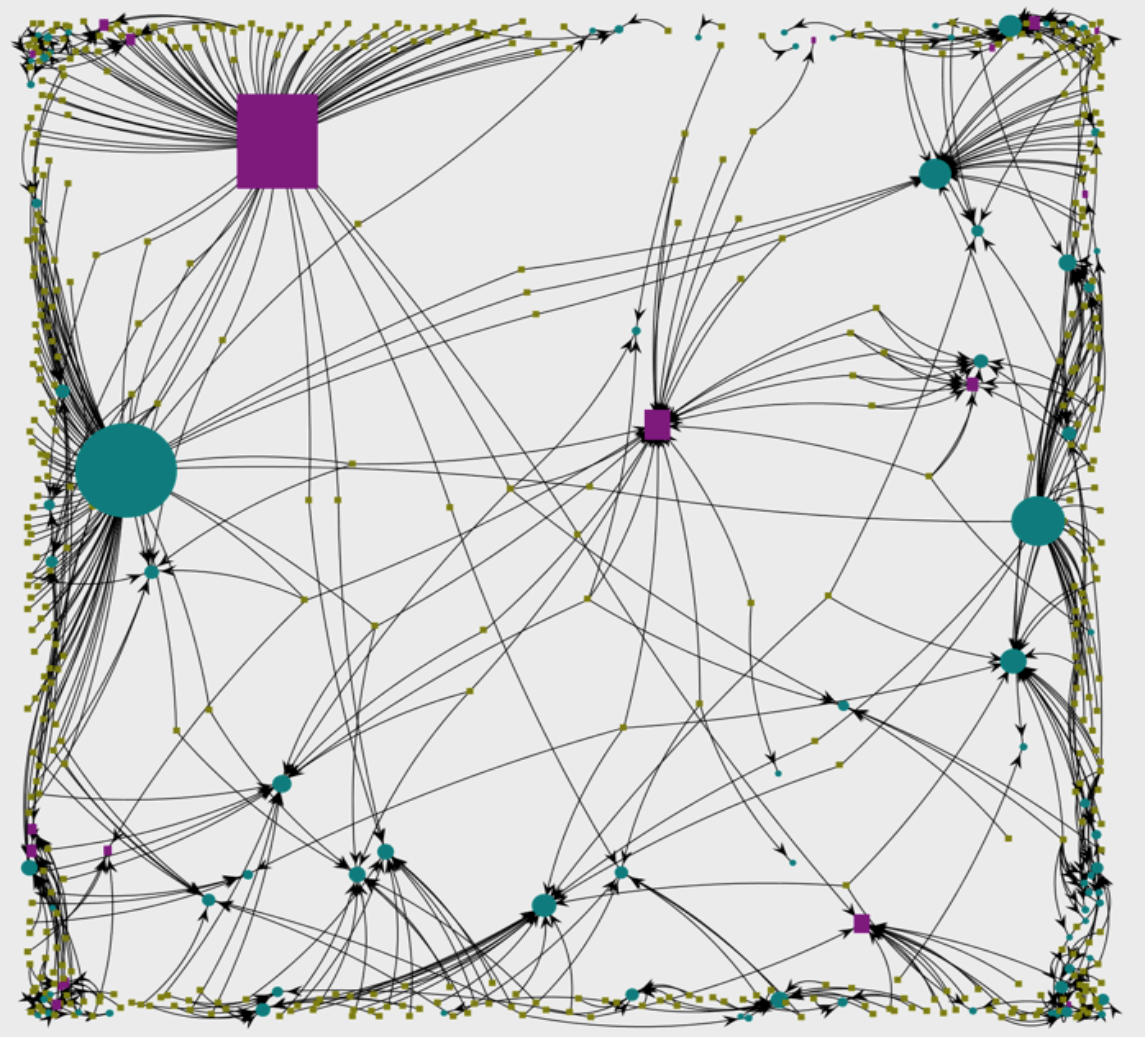


Figure 6.8 Workflows and all types of services, FRLayout

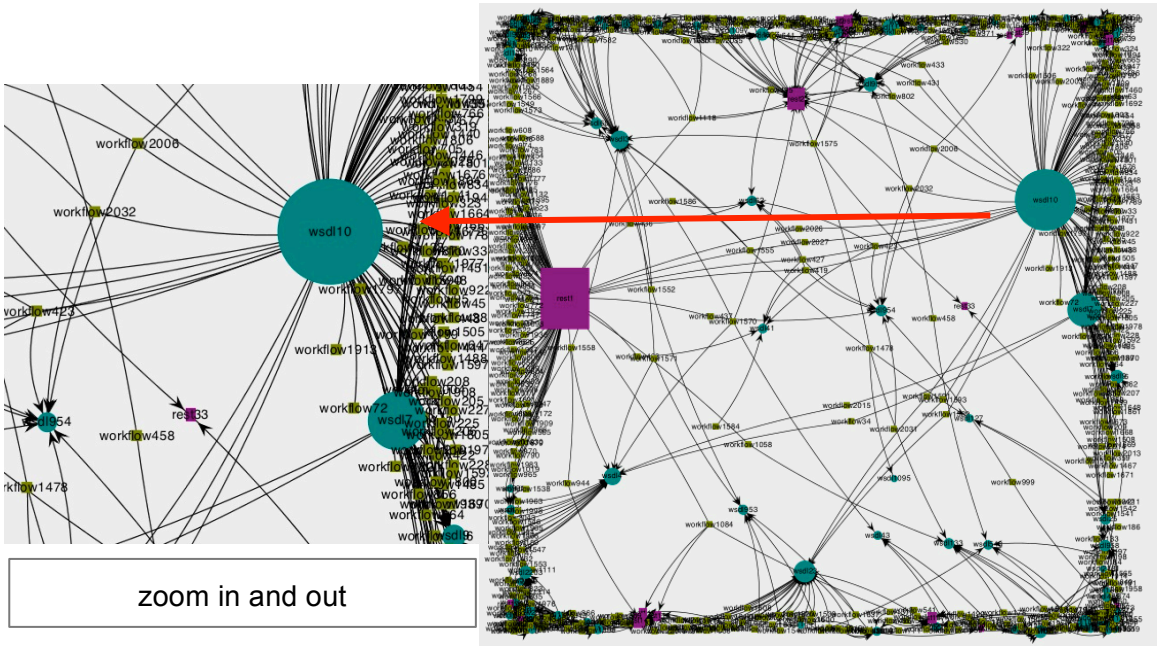


Figure 6.9 Workflows and all types of services, FRLayout with tags

Up to now, the figures we present don't have tags. We can add tags like names or ids to the nodes and edges, as in figure 6.9. However, because we have thousands of workflows and services, adding tags makes it hard to see the nodes clearly. We provide zoom in and zoom out function to address this problem.

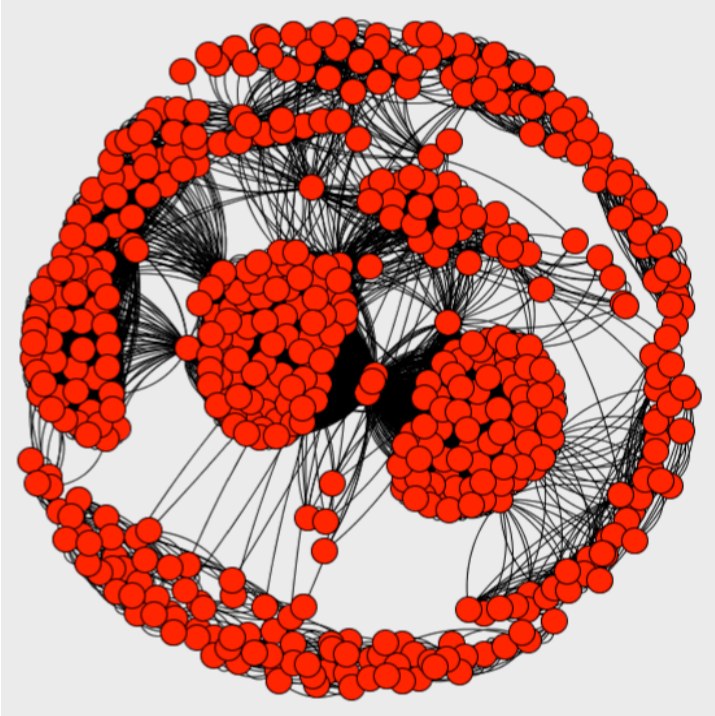


Figure 6.11 Workflows to workflows

So up till now, we have the relationship between workflows and web services. But when people start to build a new workflow, they not only want to call external web services to save time, they might also want to leverage existing workflows. We want to build the bridge between workflows and workflows. And what link them together in our case is the web services they call. If two workflows call the same service, then they must have some kind of relationship, thus we add an edge to the two nodes representing the two workflows. Since they are all callers, the graph is not directed. By doing this, we can group similar workflows from the perspective of web services they call. This is the final result after the graph stabilizes (stops evolving). We can easily see groups of workflows from this graph. Also, it's obvious that some nodes are of high centrality points. For example, if we remove the three nodes at the center of the figure, a group of workflows on the left of them and another group of workflows on the right of them will not be directly related. These high-centrality-point nodes are very important to the overall structure of the graph. For example, there are the nodes in the center of the graph. If we remove these few nodes, then a lot of nodes surrounding them will become isolated.

7. Conclusions and future work

In this project we mainly focus on building a PSW network that can be used to find appropriate artifacts (workflows and services) and obtain advice on their use. During the exploration we find out that much knowledge can be extracted from existing artifacts. By mining historical artifact usage patterns, we show how to answer various queries.
By analyzing the visualized result we got from experiments we found out that despite the rich information embedded in the PSW network only little is utilized by developers and scientist currently. So significant improvements are awaiting to be made. Further exploration can be made includes building a recommendation engine based on our history-based Network analysis techniques and semantics-based discovery ones. Efforts can also be made to accumulate practice data to create benchmarks for the presented approach.

8. Contribution of each team member

Jian Jiao: Processed raw data. Visualize data experimental result.
Ming Qi: Worked on extract web services and cleaning the dataset. Enriching and beautify the visualized results using different layout.
Qiwen Guo: Worked on visualizing experiment results.
Yu Gu & Jin Xi: Worked on extracting workflows and cleaning the dataset collected.

Appendix:

-Github url: https://github.com/JianJiao/IBM_Big_Data.git
-Check in everything onto GitHub under the predefined directory including the following items

-Readme file: Describe briefly the purpose of the project, how to download and install the software, how to use the software

-DataCollectionAndAnalysis: Java project doing the data collection and analysis work

-Visualization: Java project for visualizing data and results

-src (sub-directory): include all source code categorized by packages

-lib (sub-directory): include all related library packages needed to support the project

-contact:

Jian Jiao: 6509605291 jian.jiao@sv.cmu.edu
Ming Qi: 6509605598 ming.qi@sv.cmu.edu
Qiwen Guo: 6509605763 qiwen.guo@sv.cmu.edu
Jin Xi: 6506257840 jin.xi@sv.cmu.edu
Yu Gu: 6506257831 yu.gu@sv.cmu.edu

-Documents (sub-directory): in different WORD files

-presentations (ppt file)

-technical report