Universiteit Leiden
Opleiding Informatica

Building an administrative system

in a scientific workflow system

Name: Huseyin Sener

Date: 27/08/2015

1st supervisor: Prof. Dr. J.N. Kok

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Over the years scientific workflow systems are becoming more advanced and more commonly available. These tools for implementing and sharing scientific workflows are mainly being used by domains like bioinformatics for complex distributed scientific calculations. In recent years these tools provide an increasingly more friendly user interface and easier to use environments. For this reason we are interested if we can implement common administrative procedures in these scientific workflow systems and make it usable in a practical environment. For our research we will take an existing administrative procedure as a case study and try to implement it in a scientific workflow system. We will compare the most popular scientific workflow systems and make a reasoned choice in which scientific workflow system we will use. We will then discuss the results as well as our findings of implementing the case study.

Contents

1 Introduction

A workflow, see Figure 1, is a schematic representation of a real-world process consisting of individual steps, where the result of one step is input for the next step. Each step represents an activity or process which needs to be performed.

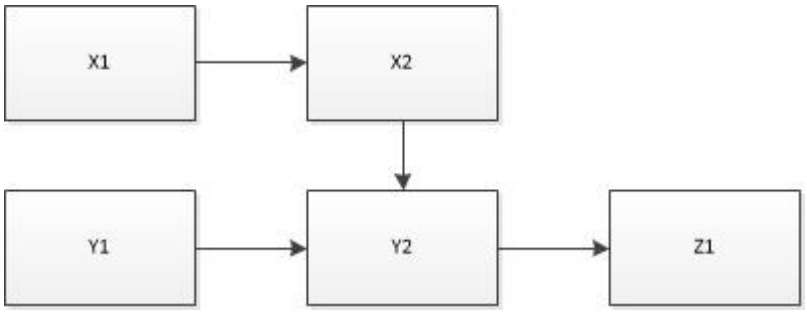


Figure 1: A workflow.

Workflows are most commonly used in business processes and lately more and more in science, especially in bioinformatics and data mining. Scientific workflows, as they are called, are widely recognized as a "useful paradigm to describe, manage, and share complex scientific analyses" [1]. Scientific workflows as one can imagine are useful in managing complex scientific computations. Scientific computations are complex and the workload of these computations are most of the time very heavy, this is where scientific workflows come in the picture. It does not only bring structure to the process of the computations, it also provides the possibility to distribute the computations over multiple processes and hardware. The tools for implementing and sharing these scientific workflows are called scientific workflow systems. These systems provide the building blocks for a scientific workflow. Each process in the scientific workflow system can be a variety of services, ranging from executing a calculation to invoking a web service. These processes are then linked according to the data flow and dependencies among them [2].

The purpose of this research project is to see if we can use these scientific workflow systems to implement an every day administrative task that bears the look of a workflow. There are quite a few scientific workflow systems out there, we will briefly go into the differences of the most popular scientific workflow systems and make a choice in the one we will use for this project. We will then try to implement this workflow and discuss the issues and flaws of the system and the results of our implementation. Furthermore we will discuss

what is needed to make practical use of a scientific workflow system in an administrative environment.

The rest of this thesis is structured as follows. In Section 2 we discuss a few popular scientific workflow systems and make a choice in the system we will use, Section 3 gives a description of the problem and the case study we are going to implement, in Section 4 we discuss the implementation of the case study as a workflow and finally in section 5 we discuss the results, give our conclusions and discuss further work.

2 Scientific Workflow Systems

There are quite a few scientific workflow systems out there that we could use for this project. A few of the most popular systems are Discovery Net, Taverna, Triana and Kepler. Each of these have their advantages and disadvantages, which we will discuss then in this section.

2.1 Popular Scientific Workflow Systems

2.1.1 Triana

Triana is a data driven workflow system. This means that execution of the dataflow proceeds by running all the nodes with no predecessors and continues until there are no more nodes left to run [3]. Triana does not have a seperate data flow and control flow layer, but rather provides a set of control flow components which are on the same level as the dataflow. It provides seperate components for branching, parallelism and looping which can be freely combined with the data flow components. What Triana lacks compared to the other major scientific workflow systems is the ability of a node to have multiple ports that produce different output types. As in some other scientific workflow systems the source code for each node can be modified, which in Triana is pure Java.

2.1.2 Kepler

Kepler is a scientific workflow construction, composition, and orchestration engine, focusing on data analysis and modelling [4]. Kepler is an orchestrated type of workflow system. Instead of trying to provide a generic semantic for all possible types of processes, Kepler separates the execution engine from the workflow model. To do this Kepler works with four core type of "directors" which orchestrate the execution of the workflow. Keplers workflow components

are called actors, which represents an operation with a number of ports that transports tokens. The simpelest interaction of an actor is when it consumes one token on each input port and produces one token on each output port whenever it "fires". Directors are the key component in Kepler. While actors and relations between the actors make up the workflow, the directors make the execution decisions. The directors make up the control flow so to say. The four types of directors are SDF - Synchronous Dataflow, PN - Process Network, CT - Continues Time and DE - Discrete Event. Kepler supports embedding of workflows, depending on compatibility of the directors. The semantic strictness of the directors can be classified as strict, loose and loosest, and embeddings are only allowed when the inner director is at least as strict as the outer director.

2.1.3 Discovery Net

Discovery Net 3.0 provides a 3 layer approach to modelling scientific workflows. The top layer is the control flow layer. It provides control flow operators for the coordination of the dataflow operations. The middle layer is the dataflow layer, which provides data integration, transformation and processing using distributed services. The bottom layer is the grid control layer, which enables access and control of remote Grid computing resources. Discovery Net is a model-driven orchestration, which means that it performs only the operations needed to produce the required result. One of the key features of Discovery Net is the automated mapping of workflows into reusable services. It also has support for embedding of workflows.

2.1.4 Taverna

Like Triana Taverna is a data driven workflow system. It uses SCUFL as its workflow language which is mainly a dataflow language, with some additional control flow like constructs. SCUFL can integrate any Java netbean executable code as a component. The dataflow nodes in SCUFL are called processors. These processors can be connected with two types of links, a "data" link which provides data transfer between processors and a "control" type link which can be used to determine the order of execution. More details about Taverna is provided in 2.2. The main advantages of Taverna is that besides the easy to operate user interface it provides interesting features like implicit iteration, failure mechanisms and incoming link strategies [6].

2.1.5 Choice of Scientific Workflow System

As already described there are some differences between the scientific workflow systems, but also quite some similarities. Any of the scientific workflow systems above can probably be used for the purpose of this research project. After considering the pros and cons we have chosen to use Taverna for this project. The main reason for this is the wide availability, the user friendly interface and ease of use of Taverna. This makes it approachable and accessible for administrative tasks. Also the fact that all processors in Taverna are a netbean based component, where it will accept any netbean executable code as a component, makes it flexible and interesting from a programming point of view. That Taverna is a data-driven workflow system makes it a more challenging choice for an administrative workflow, where most of the time there actually is no data passing between processors.

2.2 More about Taverna

Taverna is a scientific workflow system aimed on the bioinformatics domain, a domain that has mainly the need to build scientific workflows from numerous remote web services. That is why Taverna provides a large collection of web services components. In addition to these web services components, Taverna provides a set of generic components for easy integration and development. The main target audience of Taverna usually does not have a broad knowledge of scripting or programming languages, therefore in order to allow ease of development Taverna provides a window-based, user friendly interface. It provides a library of example workflows and standard or shared custom services which can be used in development.

2.2.1 SCUFL

Taverna uses SCUFL as its workflow language and Freefluo as its enactment engine. SCUFL (Simple Conceptual Unified Language) was developed specifically for the taverna project and is a language for representing workflows as directed acyclic graphs. The execution units in SCUFL are the processors. These processors act as a function that takes zero or more inputs (sources) and generate one or more outputs (sinks). The input and output data are represented as ports in het processor. There are two types of links between the processors, first is a "data" link which provides data transfer between the two processors. The second is a "control" type of link which makes the processor wait until the previous processor is finished without transferring data between the processors. The need for such a link arises because Taverna is a data

driven workflow system, just like Triana. The workflow executes based on the data that is passed through the processors, when no data is passed through to a processor that does expect an input, the processor fails. So to overcome this Taverna provides a "control" link which does not pass data, but does provide the order of execution. The other control component which Taverna provides is an indirect conditional construct, an if/else structure so to say. It is indirect, because the user has to ensure that only one of the nodes succeed and the rest fails. Since Taverna 2.0 it also provides the possibility of looping processors, where the processor loops until the output has met a certain requirement.

2.2.2 Data-agnostic

One of the guiding principles of SCUFL is that it is data-agnostic. This means that the datatype of data passing through does not have to be specified, which gives flexibility. This still leaves the problem of distinguishing between single input/output and a collection of inputs/outputs. To overcome this SCUFL introduces depth of data, which specifies a tree like structure. Data with depth 0 is a single input, depth 1 is a list, depth 2 is a list of lists, etc. SCUFL has a mechanism called configurable iteration, this enables to configure the handling of the processor input . For example if the processor with function f takes one input a, the default output is $f(a)$. If the designer knows that the input can also be a list he then can apply a mapping for the input, so that for an input $[a_1, a_2, ..., a_n]$ the output will be $[f(a_1), f(a_2), ..., f(a_n)]$. If the input for the function is two lists, then the function can be applied as a dot-product or a cross-product. This approach generalizes SCUFL processors so that as little as possible processors need to be used in a design. The example below in Figure 2 you see an example if this. The first concatenation *ColourAnimals* processor is a dot-product, whereas the second concatenation *ShapeAnimals* is a cross-product. They both use the same concatenation component, but produce different results.

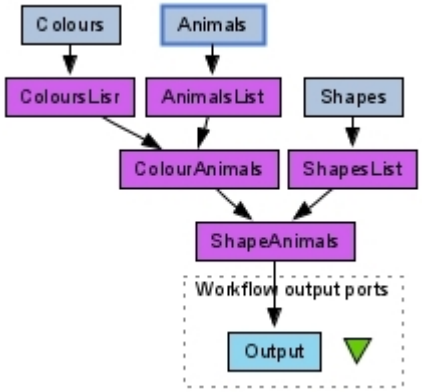


Figure 2: Example of configurable iteration in SCUFL

3 Problem description

As we said before, we are interested in the question if a scientific workflow system can be used to implement an administrative process which has the bearings of a workflow. To do this we will use an existing administrative process as a case study and will try to implement this in Taverna. The case study we have chosen for this is *the procedure towards defense of a PhD* at Leiden University. This process is quite complex and has its own web page [7] which we will use as a guidance in developing the workflow. Before starting the implementation in Taverna we will draw out the process as a workflow, which we then can use as a reference when implementing.

3.1 The procedure

The procedure can roughly be divided in seven sections. The first is **Admission**, the student has to apply for admission to the graduate school and needs to get a supervisor appointed. Next the student has to attend the **PhD training programme** during the course of the PhD. The students need to complete 4 compulsory courses and receive a certificate. After completion of the manuscript the supervisor has to approve it as a **Dissertation**, in this stadium a promotion committee needs to be assembled who will approve the dissertation. After approval the **defense of the dissertation** needs to be scheduled. When the date is set the dissertation needs to be **sent to the printers**. **Towards the defense** the Opposition committee needs to be assembled. Then the dissertation needs to be **distributed** accordingly to specific groups. This is globally what the

procedure looks like, for more details you can address the website [7].

3.2 Workflow

In appendix A we have included the schematic workflow for the procedure towards the defence of a PhD. The fact that we were able to draw out a workflow for the complete procedure means that this procedure was indeed a workflow and that we should be able to implement it in Taverna. There are a few interesting steps in the workflow that need some mention. For example when the PhD student has finished the dissertation a promotion committee needs to be assembled. This committee needs to satisfy some requirements and we will have to build in this check for the requirements in our workflow in Taverna. The same goes for later in the workflow when an opposition committee needs to be assembled. This also needs to satisfy some requirements and the check for this we will also need to implement.

4 Implementation

4.1 Data storage

Before implementing the workflow we had to make a few choices. The first choice to make was how we are going to store the data? Taverna provides a few options here, for example we could use a SQL database. Taverna provides native jdbc services to execute SQL queries on a database. The upside of a SQL database is that the data can be easily stored and accessed, but the downside is that the user needs to set up a database. Since the data for our workflow is going to be restricted in size and complexity we will not be needing a SQL database. Therefore we have chosen for a simple csv file output, where we can read from and write to in the workflow. Taverna also provides native i/o workflow services for this task. We have used the native *Read Text File* service from Taverna, but this task has one big flaw. Taverna tries to generalize the task by also making it able to read files from a URL, but by doing so they fail to catch the error for when there is no file on the disk or at the URL address. To overcome this flaw we have adjusted the native service slightly by adding a try/catch block to the code. We do this at every read file service we use. When writing the file the native*Write Text File* service should suffice for simple workflows, but it does not have native support for replacing data, it can only write new files or overwrite existing files. We have adjusted the write service so that it will look up the student id and it will replace the data of the student if the id exists, if not it will add a new row to the file.

4.2 Data driven

As we said before Taverna is a data driven scientific workflow system. An administrative workflow does not always pass data to the next connection, e.g. a processor that only needs to open a url. Therefore we have chosen for a system that every processor passes data to the next processor. The data we pass consists of two variables, namely a data variable which holds the data for the selected student and a phase variable which holds the progress of the workflow. The data variable which we have simply named *string*, holds the data for the selected or added student. It is a comma separated string which holds the basic data like student number, name, etc. and it also holds execution information for each processor in the workflow. With boolean outputs it keeps track of each step in the workflow. An example *string* looks like:

9999999,Sener,Huseyin,hsener@somemail.com,0611111111,Researchgroup,ProjectTitle,01-01-2014,true,false,true,false,true,false

As you can see, first there is some personal information of the student and then there are the boolean parameters that give information on each step in the process and its status. Some of these steps are mandatory and will finish the workflow when they output false, but others are optional and will not finish the workflow when they output false. This string is passed between processors and the data is adjusted accordingly depending on the output of each process. *False* can be changed to *true* if a process that was not complete before is completed and a new parameter might be added when a new process is reached. This way the data is built up that will be written to the data file at the end of the workflow.

The second variable which is passed through the workflow is *fase* (Dutch for phase). This variable keeps track of the phase the workflow is in. When reading the reference data in the very first step, if an existing student is selected the *fase* variable is determined on the basis of the boolean parameters in the *string* variable. When a new student is selected *fase* parameter is set to 0, which is the phase where you need to insert the student data. If the student exists, but there is no boolean parameter yet, then the phase is set to 1, which is the first step in the procedure.

4.3 Nested workflows

Taverna has support for nested workflows, which makes it possible to use other workflows as part of a larger workflow. Aside from this it also makes it possible to better structure a larger workflow. We have chosen to split our workflow in

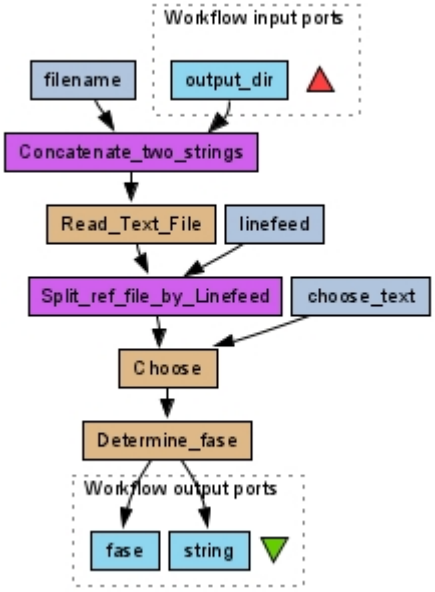


Figure 3: Nested workflow Read_reference

10 smaller workflows which makes it easier to maintain and follow our workflow. The sections we have chosen are: Read_reference (see Figure 3), Student_information, Admission, PhD_Training_Programme, Manuscript_Dissertation, Schedule_defence, Print_dissertation, Towards_defence, Dissertation_distribution, Write_file. Each of these smaller workflows either correspond with the sections in the procedure towards defense of a PhD [7] or is for the i/o part of the workflow.

4.4 Student information form

This is actually not a part of the official procedure towards defense of a PhD. This part of the workflow collects the student information for administrative purposes. To collect the student information we need a form which can be filled in, but Taverna does not have a native service for this. Therefore we can make our own processor in BeanShell. There are alternatives like for example the BIFI (Beautiful Interfaces for Inputs) plug-in [8], which is like the name suggests a GUI definition language to lay out user interface and to define parameters. However for this project we have chosen for a more direct

approach by implementing it in BeanShell. This is one of the things that make Taverna flexible, if you have Java/BeanShell experience you can easily expand or create processors for your own need. The form we have implemented lets you input some student information and generates the output *string* which will serve as a basis for the data we will write as output in the reference file.

4.5 Tell

The most frequently used service by us is an adjusted *Tell* service. The standard *Tell* service takes two inputs, *title* (optional) and *message*, which as the name suggests are the title and message of the dialog that is shown. The standard *Tell* service is a dialog that only shows a message and has a OK input button. The adjustments we have made are that the dialog does a check on the *fase* variable to determine if the service should run, the dialog has a YES/NO option instead of a single OK option so the user can confirm the procedure is finished and the service adds a true/false to the boolean parameters depending on the answer of the user. With these adjustments we lay a basis for foundation of our workflow.

4.6 Open URL

Another service that we use frequently is the *Open web browser at a URL* service. This is a native service which does exactly what it says, it opens a URL in the standard web browser. The only adjustment we have done is that we have added a check on the *fase* parameter to check if the URL needs to be opened or not. To check if the procedure corresponding with the URL is completed we add a *Tell* service right after the URL service where the user needs to confirm the step is completed. For example, when a URL is opened to a web form which needs to be filled in, the user then can confirm that the web form is filled in or not. Like all the *Tell* services this determines if the phase is complete and the workflow can continue to the next phase.

4.7 Committee form

When the dissertation is done the user has to assemble a Promotion Committee, this committee needs to satisfy a certain set of requirements. To implement this part of the procedure we have made a few custom services. Again thanks to Taverna services being based on BeanShell we can easily program the necessary functions ourselves. The *Committee_form* service is a service that shows a dialog for the user where the information about the committee can be

filled in. The information needed about the committee is as folows: Surname, Name, Function, Degree and University. The service checks if there is a csv file with previously filled in information, if the csv file is found it will show the information in the dialog so that the user does not have to fill it in all over again. Aside from the standard *string* and *fase* outputs this service has an extra output named *info* which contains a List with the committee information. This information is needed by the next service to do a check on the committee requirements and the service after that which writes the information to a separate csv file. This file is used for the check mentioned before.

The *Check_Committee* service checks the filled in information about the committee against a few requirements. The committee needs to have at least five members of which one is the scientific director and the (co-)supervisor. Furthermore at least one of the members has to be from a different university and the majority of the committee needs to be professors. This service does these checks and if these conditions are fulfilled the workflow will continue to the next phase. If not the workflow will end after the next service that writes the committee information in a separate csv file.

Later on in the workflow the *Committee_form* service is used again, this time for the Opposition Committee. The basis for this service is the same as before, the only difference being that it writes the information to a different reference file. So per student we will generate two reference files for the committees. Further on in the workflow we again use a *Check_Committee* service, this time with different checks. This committee needs to have at least 7 members of which one should be the Rector Magnificus and one should be the Scientific Director. The majority of the members should be appointed at Leiden University. All members have to hold a doctors degree and the majority should be a professor. These checks are again implemented in BeanShell. Once the check is complete the next service writes a reference file to the disk.

4.8 External dependencies

While coding in BeanShell it happens that the same piece of code occurs in multiple services. While not necessarily a problem it is annoying to copy the same piece of code or class from one service to the other and it does not do well for the readability. To overcome this, Taverna services have built-in support for external dependencies. This makes it possible to include external classes in our BeanShell code. To do this the external classes need to be compiled as a .jar file and the .jar file needs to be put in the libraries directory of Taverna. To make the services lean and readable we have chosen for two classes we want to implement as external dependencies. The first is a SpringUtilities.java class [9]

provided by Oracle. This class facilitates ease of use of the SpringLayout class when formatting the layout of dialogs. We use this class for all our dialogs where the user has to put in some data. For example the *Committee_form* service or the Student information form service, see 4.4. The second class we use is the replacestring class. This class we use in almost all our services. This class provides the possibility to replace a part of a string with a new string. We use this in our services to build the string of data that we write in the reference file, see 4.2. Instead of copy/pasting the code in each service we have chosen for the better solution of making it a external class we just import in each service.

5 Conclusions

The research question we asked ourselves for this thesis was if we could implement an administrative procedure in a scientific workflow system. We were interested in the limitations of a scientific workflow system when doing something it was not entirely designed for. We can say that we have successfully implemented a case study in Taverna and have proven that it is possible to use scientific workflow systems for non-scientific workflows. It appears that scientific workflow systems are designed to be extensive and robust systems, this is so because scientific workflows need many different kinds of services and the workflow systems needs to be able to provide this. Therefore they are built to be flexible, an example of this is that processors in Taverna are based on BeanShell. This way the scientific workflow systems make it possible to implement or adjust services to our own need. One of the problems we faced while implementing our case study in Taverna was the fact that Taverna is a data-driven workflow system. To make the implementation possible we proposed to always send two variables from one service to the other. These two variables determine if the service should run and let Taverna believe that there is always data being passed through. The other choice we had to make was making the workflow a pipeline, this means that there is no branching in our workflow. This makes it easier to control the workflow through the variables we are sending and does not add any unnecessary complexity. By successfully implementing our case study in Taverna we have shown that an administrative process can be implemented in a scientific workflow system, but this is possibly not enough to use this implementation in a practical environment. We will discuss the limitations and possible improvements in the next subsection 5.1. In subsection 5.2 we discuss the fact that there has been an update to the procedure and what needs to be done to adapt the workflow.

5.1 Limitations and further work

Although we have successfully implemented our case study in Taverna it has its limitations in a practical environment. The first is that we need to install Taverna Workbench on the client to be able to run the workflow, which basically means that the user always needs the rights to install new programs or needs to have Taverna available. Executing a workflow in Taverna could also put quite a heavy load on the client, especially bigger workflows like the one we have implemented. This might not always be available in an administration or business environment, for example when the user is working on a thin client. Second is that you need a local copy of the workflow file and you need to edit the output directory for the workflow to a directory where you have read/write permissions. The third is that we need to have the external dependency files available and be able to put it in the Taverna libraries folder, this also might not always be possible to do. To overcome these limitations further work based on this thesis could be to explore into the possibilities of Taverna Server [10]. Taverna Server enables the user to set up a dedicated server for executing workflows remotely, which could eliminate some of the limitations by moving the workload of the execution to a dedicated server.

More further work based on this thesis could be the development of some of the services we have built into commonly available services called components, so that every user even without programming skills can develop administrative workflows. Taverna provides the means for this with the Taverna Workflow Components module. This is a system for creating shareble, reusable sub workflows that perform clearly defined tasks [1], built into Taverna Workbench. Building these components makes the workflow also easily adjustable to changes, keeping in mind that these components need to be reusable and generic.

5.2 Update to the procedure

While writing this thesis the case study we have researched has received an update and new regulations apply to the procedure towards defense of a PhD [11]. The old regulation still applies in some cases, but the new regulations are in effect from february 8 2015. This means that our workflow needs to be adjusted to the new regulations if we still would like to use it in the future. The new regulations are the mostly the same in global aspect, but the details seem to be different. For example the former PhD Committee had the following requirements:

- the chairman is the Scientific Director of your Institute

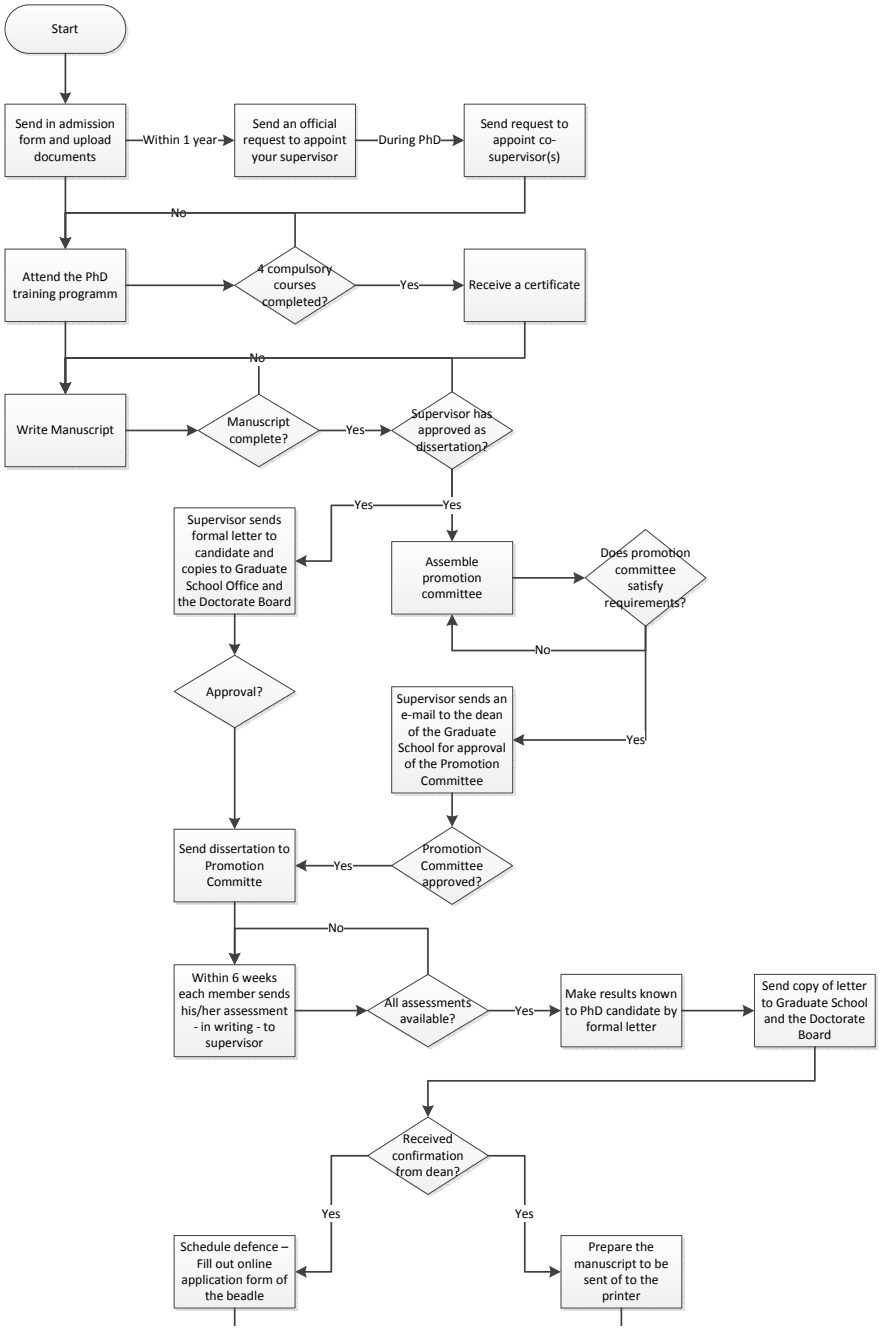
- your (co-)supervisor(s)
- at least three other members with at least one of them not employed at the Leiden University
- the majority of the committee members should be a professor.

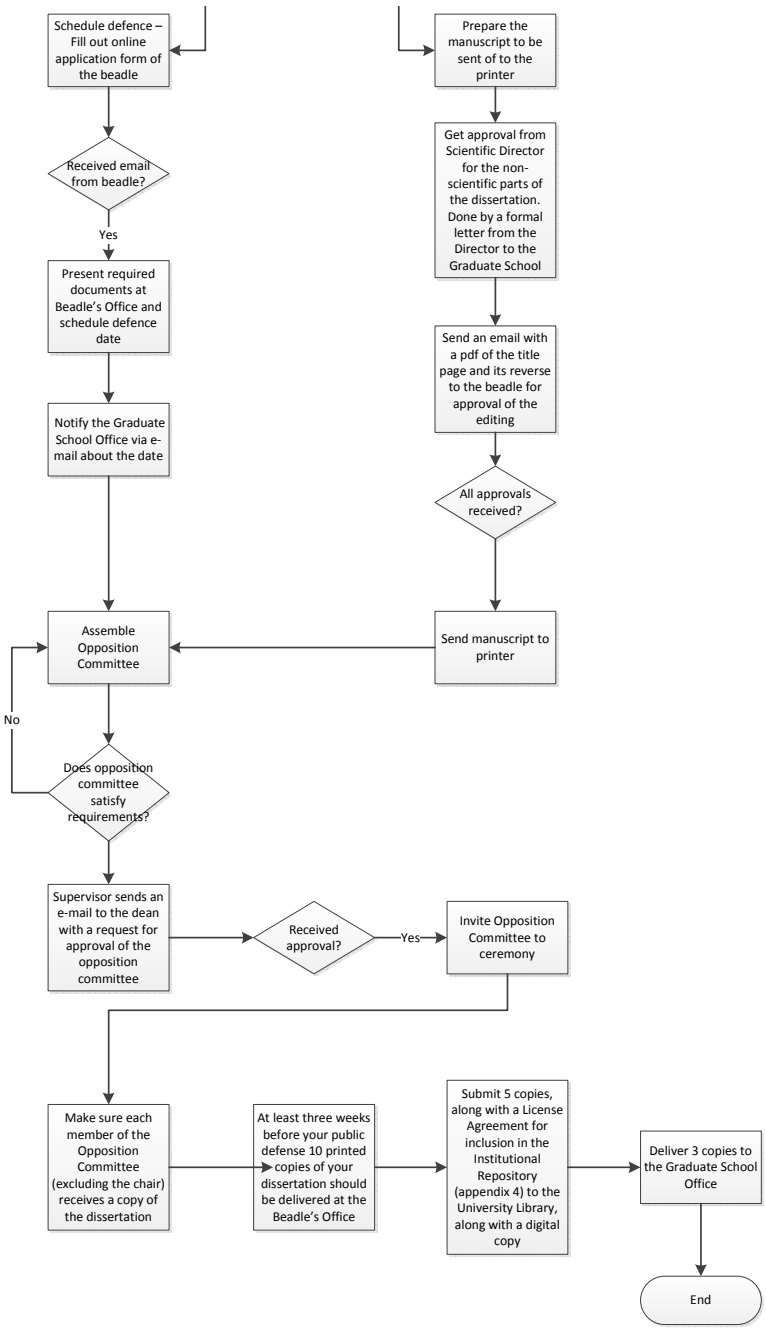
This has been replaced by a Doctorate Committee which has the following requirements:

- the chairman is the Scientific Director of your Institute
- at least three other members where one of them is appointed as Secretary of the Committee by the Scientific Director
- at least two members of the committee should not be involved in the practical realization of the dissertation
- at least two members of the Committee should not be appointed at the Faculty
- as a rule, the committee will include at least one male and at least one female member
- The supervisor and co-supervisor are not e part of the doctorate committee
- the majority of the committee members should be a professor.

Although the service can be reused, it has to be adjusted to the new requirements. To have a complete overview of the scope of the changes one should put the new regulations next to the old and list the changes that have been made. Only then an estimation can be made of the time required to adapt the workflow. This update has exposed a weakness of our system, it is not easily adjustable for future updates. Therefore instead of just adapting the workflow one could also explore the possibilities of creating generic components which can be easily adjusted if something changes again in the future. Further work can be done in this area.

A The schematic workflow




Schedule defence – Fill out online application form of the beadle
Received email from beadle?
Yes
Present required documents at Beadle's Office and schedule defence date
Notify the Graduate School Office via e-mail about the date
Prepare the manuscript to be sent of to the printer
Get approval from Scientific Director for the non-scientific parts of the dissertation. Done by a formal letter from the Director to the Graduate School
Send an email with a pdf of the title page and its reverse to the beadle for approval of the editing
All approvals received?
Send manuscript to printer
Assemble Opposition Committee
No
Does opposition committee satisfy requirements?
Supervisor sends an e-mail to the dean with a request for approval of the opposition committee
Received approval?
Yes
Invite Opposition Committee to ceremony
Make sure each member of the Opposition Committee (excluding the chair) receives a copy of the dissertation
At least three weeks before your public defense 10 printed copies of your dissertation should be delivered at the Beadle's Office
Submit 5 copies, along with a License Agreement for inclusion in the Institutional Repository (appendix 4) to the University Library, along with a digital copy
Deliver 3 copies to the Graduate School Office
End

B User Manual

B.1 Initial setup

The deliverables for this research project contain the following files:

- The Taverna workflow file: Phd_Administration_Vx.t2flow
- The reference file ref_informatie.csv containing the student information reference.
- Two external dependencies layout.jar and replacestring.jar. These contain java code which are used by multiple processes in the dataflow, so instead of repeating the code for each process we have built it into a .jar file and import it in the processes where needed.

Before starting you will need to install Taverna Workbench from the Taverna website [1]. After installing you can open the file Phd_Administration_Vx.t2flow. The file contains the complete workflow. In the upper left corner you will find a Text constant called output_dir, before running the workflow change this to an empty directory of your choice. When this is changed you need to save the reference file ref_informatie.csv to the new directory. After this we need to place the external dependencies in the appropriate folder. The files layout.jar and replacestring.jar need to be places in the libraries folder of Taverna, in windows this is: C:\Users\Username\AppData\Roaming\taverna-core-2.5.0\lib. When this is complete the workflow is ready to be executed.

B.2 Executing the workflow

The workflow can be executed by pressing the "Run the current workflow" button in Taverna Workbench. The first window gives us the possibility to chose an existing student or add a new student. When we add a new student the next window asks for information about the student. This is the information according to the ref_informatie.csv reference file. After this the next steps follow the procedure and appropriate instructions are given with each step. The workflow works as a series of mostly mandatory steps per student, once you answer a mandatory step with "no" or you cancel the action the workflow quits. You need to run the workflow again and chose the same student to continue where you left off. This is done this way because there can be a long periods of time between each step in this procedure.

References

[1] Taverna, http://www.taverna.org.uk/

[2] Gil, Y., "Examining the Challenges of Scientific Workflows", IEEE Computer Society, Dec. 2007

[3] V. Curcin and M. Ghanem, "Scientific workflow systems - can one size fit all?", Biomedical Engineering Conference, 2008. CIBEC 2008. Cairo International

[4] V. Curcin, M. Ghanem, P. Wendel and Y. Guo, "Heterogeneous Workflows in Scientific Workflow Systems", Computational Science - ICCS, 2007.

[5] Y. Zhao, I. Raicu and I. Foster, "Scientific Workflow Systems for 21st Century, New Bottle or New Wine?", IEEE Congress on Services - Part I, 2008.

[6] J. Sroka, J. Hidders, "Towards a Formal Semantics for the Process Model of the Taverna Workbench. Part I", Fundamenta Informaticae, IOS Press 2009

[7] Procedure towards defence: http://www.science.leidenuniv.nl/index.php/english/graduateschool/procedure_towards_defence

[8] A. Yildiz, E. Dilaveroglu, I. Visne, "BIFI: a Taverna plugin for a simplified and user-friendly workflow platform", 2014

[9] SpringUtilities.java, http://docs.oracle.com/javase/tutorial/uiswing/examples/layout/SpringGridProject/src/layout/SpringUtilities.java

[10] Taverna Server, http://www.taverna.org.uk/documentation/taverna-2-x/server/

[11] Amendments as per 8 February 2015: PhD Regulations 2015, http://www.regulations.leiden.edu/research/phd-regulations.html