The Workflow Synchronizer

Synchronizing event log with workflow definition

Thesis by

Rune André Liland

Department of Information Science and Media Studies
University of Bergen

Spring 2010

Acknowledgment

There are several people I have to thank that have helped me during this master thesis. I would in particular like to thank thank my supervisor Weiqin Chen who has been a great help on this thesis. I would also like to thank Jørn Klungsøyr and Peter Wakholi for all our work together. I also want to thank all of them for making this project possible.

Mathias Hellevang, Christoph Carlson, Thor Møller and Petter Brodin I want to thank for all the breaks together and feedback in various degrees of seriousness. Hans Terje Møller and Stig Solholm I want to thank for sticking together on the trip to Uganda and the openXdata conference.

I want to thank all of my fellow students who has made this year an interesting one, and for providing many interesting discussions and situations, you will all be missed in different ways.

Table of Contents

List of figures

List of tables

Abstract

Workflow Synchronizer was developed in order to support Workflow Management in openXdata which is an open source solution for surveys and studies. The Workflow Synchronizer takes an event log and matches it with the planned activities and sequences in a predefined workflow. It provides the user with what the next tasks are as well as possible errors in the event log. The Workflow Synchronizer is a custom-made workflow engine for YAWL workflow documents. The mechanism itself is generic enough to be used in any Workflow Management Systems. The Workflow Synchronizer makes it possible to automate the synchronization processes and better control the execution of activities.

The Workflow Synchronizer was tested using a set of best practice workflow patterns. Black-box testing was used, and the results for each test run were checked to make sure the results were correct. The results showed the Workflow Synchronizer is capable of finding undone tasks and identifying errors in the event log.

1 Introduction

Vaccine trials[1] are clinical trials that test vaccines prior to public release, and they follow strict guidelines. A clinical trial is a research study that tests how well new medical approaches work on people. A medical approach to treatment can be medicinal, but also objects like mosquito nets. The studies try's to find better ways to screen for, diagnose, prevent or treat a disease. Data that was not collected correctly in the trials have to be discarded, this also includes the previous data collected about that person. This means that it is important to document when the data was collected and that it is done on time.

In western countries, keeping the time frame is not not such a big problem. These countries have relative good infrastructure in regard to roads and communication like telephone and internet connection. There are also many hospitals and medical clinics.

In Uganda[2] much of the population lives scattered on the country side, far apart from each other, connected by poor roads and separated by lack of communication channels like telephone. This makes keeping the vaccine trial time frames difficult. The people gathering the data for the vaccine trial can not always report the results on time even though they collected them in time. A digital system can not synchronize results and get new jobs on most of the country side because there is no internet connection available.

OMEVAC[3] (Open Mobile Electronic Vaccine Trials) is a project that aims to reduce the resource requirements of doing vaccine trials in a setting with mid-low-resources. Its objective is to develop a complete secure, mobile and electronic system for data collection and management in vaccine trials. openXdata[4] (previously EpiHandy) is one of the software solutions that OMEVAC builds upon and is an open source project. The goal is to have paper work moved to the digital platform, one reason is to remove many of the human flaws that occurs when using pen and paper. openXdata is a software solution for data collection and data management and can run on a wide variety of mobile devices, including mobile phones costing less than 50$ (USD). Data collection

1 http://www.nlm.nih.gov/medlineplus/clinicaltrials.html
2 https://www.cia.gov/library/publications/the-world-factbook/geos/ug.html
3 http://www.openxdata.org/Main/OMEVAC
4 http://www.openxdata.org/

is done using electronic forms that are created with the projects software solution. It allows users to create survey schemas, or schemas for vaccine trials for instance. Mobile devices and computers are used when answering the electronic schemas. There are also many extensions for the openXdata system developed by different contributors, for instance an extension that shows a map with markers[5] that contain data for that location about dog bite incidents recorded in a rabies surveillance project. This is generated based on the data collection done with the openXdata system. But openXdata lacks a workflow system for managing and preforming multiple schemas according to a predefined definition.

What's needed is a workflow system for mobile devices that can store the workflows and their results locally on the mobile device. Many of the environments where the workflow system will be used are disconnected, in this case that means there is no Internet available via Ethernet, wireless LAN or mobile phone network. The capability for the workflow system to store necessary things locally in order for the workflow to be executed is crucial.

The system needs to be able to manage the execution of a workflow, log the progress, return what task is next and be integrated with openXdata.

There exists software solutions for handling workflows, some are based on BPEL[6] (Business Process Execution Language). BPEL is an XML based language which describes an execution of a business processes. This description can then be executed by a BPEL execution engine. The BPEL file is usually accompanied by a WSDL file (Thatte, Andrews et al. 2003), it describes the web service, how a client can connect to the BPEL process on the server. One open source workflow software that is based on BPEL is Sliver (Hackmann, Haitjema et al. 2006). It is an open source BPEL & SOAP server and execution engine for mobile devices. It is written in Java and aims to be a light weight implementation.

However, the further development of Sliver presents several problems. An alternative solution is a combination between the YAWL Editor and a custom-made software (The

[5] http://www.ghdonline.org/tech/discussion/national-dog-bite-rabies-surveillance-using-openxd/
[6] http://www.oracle.com/technology/pub/articles/matjaz_bpel1.html

Workflow Synchronizer) for workflow task management. A user should be able to create a workflow in the editor and output what tasks to do next.

The solution is tested with black-box testing. This means testing the application by feeding it the required input, and validating the output. Depending on the input, the application should output certain results. These results are checked against expected results.

The thesis is organized as follows: Chapter two presents different literature related to workflow. Chapter three presents design and development of the Workflow Synchronizer. Chapter four describes the testing of the application. Chapter five presents the findings and possible future work.

2 Literature Review

In this chapter, literature related to the project is presented including Workflow Management, process awareness and mobile workflow.

2.1 Workflow Management

Workflow Management is characterized primarily by its automation of processes involving both human and machine-based activities (Hollingsworth 1995). Workflow Management includes a workflow, and a Workflow Management System (software).

2.1.1 Workflow

Hollingsworth (1995) defines a workflow as:

"*The computerised facilitation or automation of a business process, in whole or part.*"

A workflow illustrates a sequence of procedures, these procedures are automated in the workflow. A procedure is the passing of tasks, documents and information between participants. This is done according to a set of rules to achieve or contribute to an overall business goal (Hollingsworth 1995, p. 6).

A similar definition was given by (Weske 2007, p. 50):

"*Workflow is the automation of a business process, in whole or in part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules.*"

The definition describes a type of document, but not a specific implementation of that workflow document. There are however implementations of these definitions, YAWL[7] being one of them.

2.1.2 Workflow Management System

A workflow needs software, both for creating the actual workflow document, and to handle the executing the workflow described in the workflow document (Hollingsworth

[7] http://www.yawlfoundation.org/

1995). This kind of software is called a Workflow Management System. It is defined by Hollingsworth (1995) as:

"A system that completely defines, manages and executes "workflows" through the execution of software whose order of execution is driven by a computer representation of the workflow logic."

Another definition for Workflow Management System is as follows (Weske 2007, p. 50):

"*A Workflow Management System is a software system that defines, creates, and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants, and, where required, invoke the use of IT tools and application.*"

YAWL is a an example of a Workflow Management System. It contains an editor for creating a workflow, and a server for executing the workflows created via a web interface.

In order for a computer system to be able to manage the business processes, they need to be transformed into a process definition, also called a process model, process template, process meta data or a process definition (Hollingsworth 1995). A Process Model is defined in Hollingsworth (1995) as:

"*The computerised representation of a process that includes the manual definition and workflow definition.*"

The business process is translated from the real world into a computer processable formal definition.

Process models are defined at design time using constructs and the workflows are executed based on them. The constructs includes activity models, event models, and gateway models. Activity models says what specific activity is to be undertaken, event models specifies the current state of the process, and the gateway models defines the routing.

The Workflow Management Coalition (WfMC)[8] is a consortium that creates and contributes to standards related to processes. WfMC is the only standards organization that focuses purely on process. It was founded in 1993 and is a global organization of developers, adopters, consultants, analysts, universities and research groups working on workflow and business process management.

The organization provides a reference model[9] to standardize the implementation of Workflow Management (Hollingsworth 1995). It specifies a framwork for workflow systems, identifying their characteristics, functions and interfaces. The framwork has five categories of interoperability and communication standards. These allow workflow products to exists together and interoperate within a users environment.

There has been created many Workflow Management Systems (WFMS) based on this reference model (Jing, Huff et al. 1999). A WFMS can define, create and manage the execution of workflows that model business applications through a coordinated set of process activities (Jing, Huff et al. 1999). Software clients running on one or more workflow engines is used by WFMS. The workflow engines can interpret process definitions, and the software clients can interact with the workflow participants. When required, the software clients can also invoke the use of IT tools and applications. COSA[10], Oracle BPEL[11], SAP Workflow[12] and YAWL[13] are some examples of WFMS's.

WfMC created the process definitions[14] Wf-XML and XPDL that is used in more than 80 known solutions to store and exchange process models. Wf-XML provides a standardized way to execute and monitor program that might use a long time to complete. XPDL is a process design format, it is not an executable programming language, it stores process syntax and visual diagrams for business process models.

2.1.4 BPEL

BPEL (Business Process Execution Language) is a standard executable language for

[8] http://www.wfmc.org/about-us.html
[9] http://www.wfmc.org/reference-model.html
[10] http://www.cosa.nl/
[11] http://www.oracle.com/technology/products/ias/bpel/index.html
[12] http://help.sap.com/saphelp_46c/helpdata/en/c5/e4a930453d11d189430000e829fbbd/content.htm
[13] http://www.yawlfoundation.org/
[14] http://www.wfmc.org/about-us.html

specifying interactions with web services[15]. This standard was made by OASIS (Organization for the Advancement of Structured Information Standards) and is an organization that works for advancing structured information standards, BPEL is one of them.

There were ten design goals originally for BPEL (Leymann, Roller et al. 2003):

1. *Define business processes that interact with external entities through Web Service operations defined using WSDL 1.1, and that manifest themselves as Web services defined using WSDL 1.1. The interactions are "abstract" in the sense that the dependence is on portType definitions, not on port definitions.*
2. *Define business processes using an XML-based language. Do not define a graphical representation of processes or provide any particular design methodology for processes.*
3. *Define a set of Web service orchestration concepts that are meant to be used by both the external (abstract) and internal (executable) views of a business process. Such a business process defines the behavior of a single autonomous entity, typically operating in interaction with other similar peer entities. It is recognized that each usage pattern (i.e. abstract view and executable view) will require a few specialized extensions, but these extensions are to be kept to a minimum and tested against requirements such as import/export and conformance checking that link the two usage patterns.*
4. *Provide both hierarchical and graph-like control regimes, and allow their use to be blended as seamlessly as possible. This should reduce the fragmentation of the process modeling space.*
5. *Provide data manipulation functions for the simple manipulation of data needed to define process data and control flow.*
6. *Support an identification mechanism for process instances that allows the definition of instance identifiers at the application message level. Instance identifiers should be defined by partners and may change.*
7. *Support the implicit creation and termination of process instances as the basic lifecycle mechanism. Advanced lifecycle operations such as "suspend" and "resume" may be added in future releases for enhanced lifecycle management.*
8. *Define a long-running transaction model that is based on proven techniques like compensation actions and scoping to support failure recovery for parts of long-running business processes.*
9. *Use Web Services as the model for process decomposition and assembly.*
10. *Build on Web services standards (approved and proposed) as much as possible in a composable and modular manner.*

BPEL is an orchestration language and specifies an executable process, it involves

[15] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

exchanging messages with other systems. it uses WSDL and web services for its external communication. BPEL's messaging uses WSDL to describe outgoing and incoming messages.

A BPEL document is an XML file and contains a process definition. The WSDL document is also an XML file, and contains the service definition. The service created with the WSDL file refers to the BPEL file for the actual processes.

A BPEL file contains a process element. Within this element there can be partnerLinks, variables, sequences, assigns and flow elements to name some[16]. Variables are basically the same as in standard programming, sequence has the process definition, assign manipulates the incoming and outgoing variables, and partnerLink types represents the interaction between BPEL and the involved parties such as a service defined by a WSDL file. A BPEL file is however often not logical to many in what things mean, even with programming experience and an understanding of what words in that domain usually mean.

2.1.5 YAWL

YAWL[17] (Yet Another Workflow Language) is a workflow language which also has a Business Process Management and workflow system created for the language. It is an open source software which includes a graphical editor, an execution engine and a worklist editor. YAWL is written in Java and uses XML schemas and XQuery natively. It has a Web Service integration and can handle complex data, transformations and integration with organizational resources.

YAWL is sometimes seen as an alternative to BPEL. But one major difference is that BPEL has a committee behind it that standardizes the language, it has support by several industry players and is therefor supported by many tools, while YAWL on the other hand only has a single implementation at the moment. But since YAWL has a single implementation created by its developers, all the software in it works together, BPEL software has more fragmentation in many ways. Examples of BPEL editors are Eclipse

[16] http://www.oracle.com/technology/pub/articles/matjaz_bpel1.html
[17] http://www.yawlfoundation.org/

BPEL[18], NetBeans SOA[19] and Active VOS[20]. BPEL server/execution engine examples are Sliver, Apache ODE and ActiveBPEL[21]. What creates the fragmentation is how all the different editors and servers does not work more easily with each other. Setting up Eclipse BPEL to execute the BPEL and WSDL files in Apache ODE could be problematic since setting up the settings did not work out as in the guide[22]. A manual deployment requires creating an Apache ODE build XML file something the Eclipse BPEL editor could not do. This raises the threshold for using BPEL software with each other and creates fragmentation.

2.1.6 XProc & XML Pipeline

XML Pipeline[23] is an XML vocabulary for describing the processing relationships between XML resources. Inputs and outputs of the XML processes are specified and a pipeline controller uses this XML document to understand the chain of processing that must be executed in order to get a certain result.

When processes, also sometimes called XML transformations, are connected together, it forms something called an XML Pipeline. For example two transformations T1 and T2, they can be connected together so that an XML document input is transformed by T1, and the output of T1 is then given as an input document to T2. This is a simple linear pipeline where a single document is processed in the same sequence of transformation which produces a single output document.

XProc is an XML transformation language for defining XML Pipelines and is also a W3C Recommendation[24]. XProc is a language designed for describing operations to be preformed in XML documents. A pipeline is a collection of connected steps, where the output of one step flows into the input of another step. The pipeline itself is a step and must satisfy the constraints on steps. "*A step is the basic computational unit of a pipeline.*" (http://www.w3.org/TR/xproc/). It typically has zero or more inputs, from

[18] http://www.eclipse.org/bpel/
[19] http://soa.netbeans.org/soa/
[20] http://www.activevos.com/products-activevos.php
[21] http://www.activevos.com/community-open-source.php
[22] http://www.eclipse.org/bpel/users/pdf/HelloWorld-BPELDesignerAndODE.pdf
[23] http://www.w3.org/TR/xml-pipeline/
[24] http://www.w3.org/TR/xproc/

their inputs, the step receives XML documents to process, the result is zero or more outputs which is sent as XML documents. The output can have option and/or parameters.

The steps come in three kinds: Atomic, compound and multi-container. An atomic step is indivisible and carries out basic XML operations and can do arbitrary amounts of computation. The compound step on the other hand controls and organizes how documents flow through the pipelines using functionality similar to conditionals, iterators and exception handling in programming languages. The compound step contains other steps controls their evaluation. The last type of steps is the multi-container. It contains several alternate subpipelines and may involve an evaluation or partial evaluation of the multi-container step.

XProc is designed to allow composing of XML processes and share these compositions in a standard way. The language is a W3C recommendation as of 11 May 2010, and is relatively new, but it could become very relevant for workflows in the near future.

2.2 Process Awareness and Process Mining

Workflow participants, or users, performing workflow activities is only supported in office environments by most WFMS's (Jing, Huff et al. 1999). Many of the Workflow Management Systems we use today keeps logs of all the events that takes place in the system, and are in that sense “process aware” (Aalst 2005). A workflow systems audit trail is an example of an event log. Information systems are increasingly controlling and/or monitoring business processes, this means that many transactional information systems has “footprints” from these business processes. The events from the business processes are stored in what is often called event logs. Comparing the actual process based on an event log with the desired process to ensure compliance by mining the event log is possible because the logs are kept. To achieve this, techniques have been developed collectively referred to as process mining (Dongen, Medeiros et al. 2005). Mining these event logs for knowledge to diagnose the business processes is the basic idea of process mining. The mining has three different perspectives:

1. The process perspective

2. The organizational perspective
3. The case perspective

The *process perspectives* focus is on the control-flow, ordering of the activities. The aim of this mining perspective is to find a good characterization for all possible paths, it can be expressed as a Petri net or Event-driven Process Chain (EPC). The *organizational perspective* focuses on which performers are involved and how they are related. The goal is either to show relations between the different performers, building a social network in a way, or by classifying people in terms of roles and organizational units and thereby structuring the organization. The *case perspective* focuses on the cases properties. The originators working on a case or their path in the process is what the case can be characterized by. A case can however also be characterized by the values of the corresponding data elements.

Two techniques are proposed by van der Aalst (2005) for process business alignment: conformance testing and delta analysis. In the *delta analysis*, an actual process is compared with a predefined process representing the information system. The actual process is represented with a process model, and is obtained through process mining. The basic idea behind *conformance testing* is to directly compare the event log with the predefined process model.

In delta analysis, the actual process, or discovered model, may not be possible to find because of too few events. This could make it flawed or not representative. This problem is something that the conformance test does not have and is therefor an alternative to delta analysis. The application in the Master project can be considered as a conformance test because we check if the event log matches the pre-defined workflow and what the next task is.

2.3 Using Mobile Devices for workflows

Mobile devices have become increasingly powerful, and a modern smart phone can now rival a computer from the turn of the century. The HTC Desire[25] for instance has a 3,7 inch Touch Screen, 1 GHz CPU, 576 MB Ram, WiFi and 3G. These kind of mobile

[25] http://www.htc.com/no/product/desire/specification.html

devices allow one to run more complex and resource intensive software while also having several connectivity options. The WiFi allows one to connect to a local wireless network and the internet, which makes uploading or downloading large amounts of data affordable and fast. 3G is slower WiFi, but still quite fast and will also be practical for uploading and downloading data. The mobile devices have also large storage capacity, 32 GB for instance the limit on the HTC Desire. It means that data can be collected, saved and the uploaded at a later time when the mobile device can be connected to a LAN or a computer and the data synched.

In openXdata, Windows Mobile smart phones are used to collect data in studies. The data is stored locally on the phone. Schemas can be designed on a computer and transferred to the mobile phone where they can be answered. The answers can come in the form of text, video, sound or checking check boxes for instance.

Since the smart phones are small and have batteries, studies can be preformed in remote areas without a fixed power source. Since the phones can store the data locally, one can also be places without phone coverage. This is necessary for a flexible platform with workflow support.

Because mobile devices are so powerful, lightweight mobile workflow engines can now run on these devices. This would allow for a complete platform for user driven work processes to run of these mobile devices. Mobile devices would support workflows by adding them as one of the environments where activities can be executed. Sliver (Hackmann, Haitjema et al. 2006) is a demonstration of how on mobile devices, groupware applications are being implemented. A wide range of mobile solutions have been developed by many vendors using some kind of workflow technology.

Development of a framework that delivers workflow definitions to mobile devices in a disconnected environment has been attempted (Bahrami, Wang et al. 2006). The IBM FlowMark (Alonso, Günthör et al. 1996) is addressing the constraints of deploying mobile workflows with a meta-model. A mobile device has work loaded onto it, and the activities are now considered locked. A user should hopefully do them. The activities can only be unlocked and reassigned when the user has synchronized. IBM has also proposed to develop a distributed workflow system, it would span between multiple

devices and backend infrastructure. A distributed workflow system called Exotica (Alonso, Günthör et al. 1996) transfers processes to sites, central servers are thereby not needed. This is achieved by having a copy of the process definition maintained at all of the sites or pre-compiling the solution and determining which sites executes a particular process.

Collecting data in clinical trials using mobile workflow in developing countries will require research to find a lightweight workflow engine for mobile devices, such as smart phones, that can execute process definitions, have a log of activities and synchronize with a central server when a server connection is established.

2.3.1 Challenges for mobile workflows

A workflow system is usually a server client setup where it is assumed that both are always connected to a network and can reach each other. With a mobile device, coverage is not guaranteed, so the software will have to handle the situations where it is not able to reach the central server. This also creates challenges for the server side, the server knows that a client has accepted to do some work, but since the client is out of reach the work could be completed even if no results have been returned within a expiration time limit. Should the server trust that the task will be completed, or reassign it to another client?

Another challenge is input limitations on the devices. A mobile device, and smart phones especially, have input options that is not designed for large amounts of text, either because of a small physical keyboard or a virtual one on the screen. Even the iPads virtual keyboard on the 10 inch screen can not compete with a standard full size keyboard. The screen is also a limit, usually they will be smaller than 4 inches on a smart phone. This limits screen real estate and in turn how much one can put on the screen and the visibility of text and symbols on such a small screen. Showing a large workflow diagram on a small screen will cause the text to become unreadable and one could loose the overall perspective on the workflow.

3 Design and Development

The design and development follows the agile software development methodology. The Agile software development methodology has a manifesto that was created by seventeen people[26] early in 2001 (Bell 2005, p. 330-331):

> ***Individuals and interactions*** *over processes and tools*
>
> ***Working software*** *over comprehensive documentation*
>
> ***Customer collaboration*** *over contract negotiation*
>
> ***Responding to change*** *over following a plan*
>
> *That is, while there is value in the items on the right,*
> *we value the items on the left more.*

There are also twelve principles behind the Agile Manifesto. The focus of the agile methodology is to develop software rapidly with minimal planning, but achieving high quality. In the waterfall model, the steps from Requirements to Maintenance are completed without revisiting a previous step after it has been completed (Bell 2005, p. 291). The agile methodology also preforms these steps, but in a different way as expressed in the agile manifesto above. The twelve principles that the manifesto is based on came from an analysis of the *heavyweight* (old methods), and aim to do software development in a more *lightweight* fashion. Agile software development is therefor preformed with several iterations of requirement specification, design, development and tests[27]. During this process requirements are refined and different solutions are tested.

3.1 Scenarios

The purpose of the Workflow Synchronizer is to take a workflow and event log as input and return what tasks to do next. Here we have designed three scenarios:

- Going to the cinema

[26] http://agilemanifesto.org/
[27] http://agilemanifesto.org/principles.html

- A followup study
- A clinical trial process

These scenarios were used during development.

3.1.1 Cinema

In this scenario, a person has decided he wants to go to the cinema, the workflow shows what tasks that person has to do before finally watching the movie.

- The person has to decide if he wants to go alone, or with other people.
  - Going with other people means he has to find friends.
    - Then they have to decide on a movie to watch.
    - Finally they have to pick who buy's the tickets.
  - A person who goes alone to the cinema just makes up his own mind on what movie he/she want's to watch. That person pays for himself.
- Then the movie tickets are bought, going alone or with a group does not change this task.
- When the tickets are bought (in what ever manner is chosen), one must go to the cinema, at the time the movie is screened.
  - One can buy snacks before watching the movie, but this is not mandatory.
- Now the people can watch the movie at the cinema, and the scenario ends.

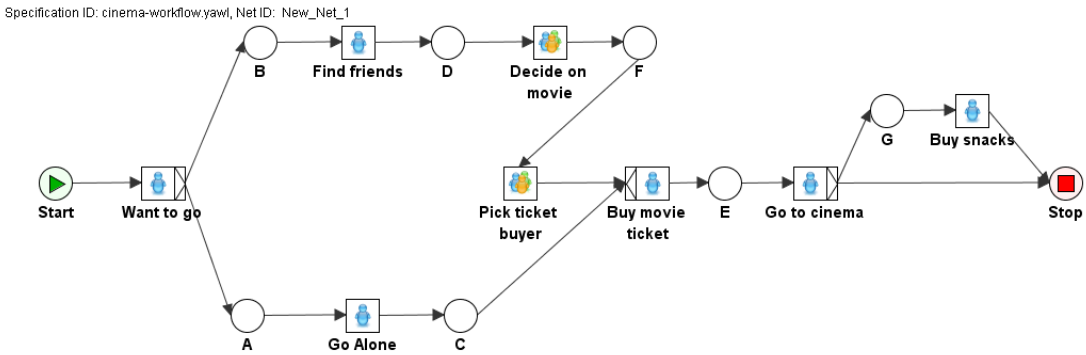

Figure 1: Cinema workflow example

Modeling the scenario described above is shown in Figure 1.

3.1.1 Followup Study

This is a followup study scenario in clinical trials.

Description:

- A candidate is registered.
- The candidate is evaluated.
  - If he/she does not pass the evaluation, the person is rejected and is out of the study.
  - If the person is approved, his/hers full information is registered.
  - The study starts, and after a week a blood test is done which the person has to pass in order to stay in the study, and schema A is answered.
  - When two new weeks have passed, schema B is answered.
  - When three new weeks have passed, a new blood test is preformed which the person has to pass, and schema A and C are answered.
  - When all is finished, the study ends.

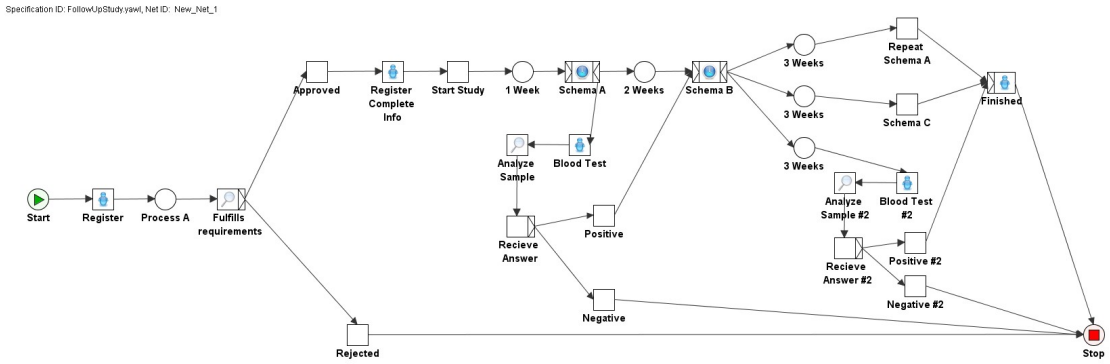

Figure 2: Followup study example

This model of the followup study (Figure 2) is however not correct. There is supposed to be a time delay between some of the schemas. The YAWL Editor has a time function, it is however disabled and it is not clear how to enable it. This is therefor represented in the model as processes with names that tell how long of a delay they represent.

3.1.1 Clinical Trial

This scenario is a model of a clinical trial process, which is a process where data is collected while testing out medicine, the point being to check what the effects of the medicine is[28].

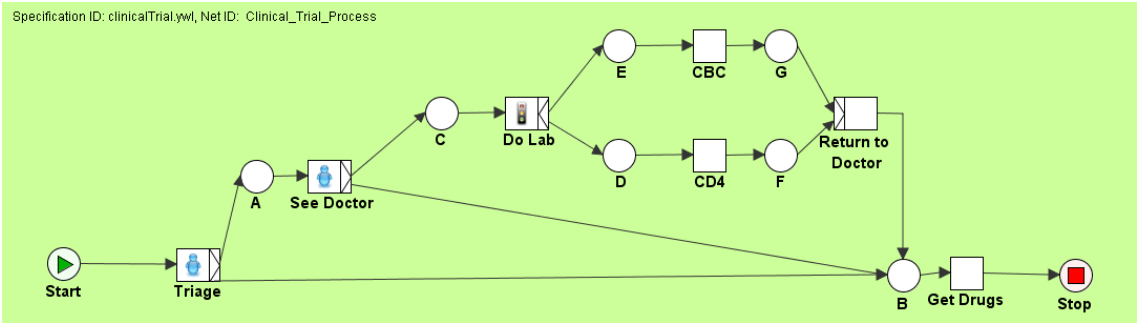

Figure 3: Clinical trial example

As shown in Figure 3, the clinical trial begins with a Triage. The general purpose of a triage is to prioritize the patients based on the severity of their situation, in this case do they need to see the doctor or do they get drugs immediately. If they need to see the doctor, the doctor will either give them a certain drug or do a lab test. If they have to do the lab test, two tests will be preformed, the CBC and CD4 test. When both of these test are complete, one will return to the doctor and then get the drugs. Now the trial process is finished. This is what the Figure 3 has modeled.

3.2 First Iteration

3.2.1 Requirements

The original requirements are gathered through interviewing the end-users of the application. The application should implement a mobile Workflow Management System and be able to integrate with openXdata. More specifically, the application should fulfill the following requirements

- Java SE based application

[28] http://www.nlm.nih.gov/medlineplus/clinicaltrials.html

- Two XML documents
  - A work definition document
  - A work data document. It describes what schemas have been completed
- Should return what tasks to do next in a workflow for each patient, when a certain given tasks have been completed, as stated in the event logs.

3.2.2 Testing different solutions

In the first iteration, we tested Sliver, a mobile workflow engine, and BPEL. Sliver is an open source software created at Washington University and a SOAP/BPEL execution engine for mobile devices. The execution engine is created to execute BPEL and WSDL files, they are the input files for the Sliver application. BPEL is XML document standard for a business process execution language.

3.2.2.1 Testing Sliver solutions

Implementing Sliver proved to be a bigger challenge than anticipated. We started working on the implementation in second half of 2009. Some of the challenges that would force us to reconsider what solution to use was the lack of documentation for Sliver, no example BPEL/WSDL documents, no fully compatible BPEL/WSDL editing software and poor documentation and guides on BPEL and WSDL files for people inexperienced with these standards. A small portion of Sliver was implemented as a server and client. This implementation was however improvised. We contacted one of the developers by email to ask for any documentation on how to implement Sliver. The available information was located in “package-info.java” files, but this documentation seems to have holes, and some example code lines are incomplete and have “...” where code is suppose to be. The implementation that was made also did not match how the info file described how to do it, so the implementation was most likely incorrect in some way. One reason for this conclusion is that further development of the initial limited implementation did not lead anywhere and create the basic functionality that was needed.

Alternative open source software was checked out because of all the issues with Sliver. One piece of software that showed promise was Apache ODE (Orchestration Director Engine)[29]. Apache ODE is an Apache Tomcat application that executes business processes written in the WS-BPEL standard. This software had several examples and instructions on how to execute them. The plan was to try and implement Apache ODE with openXdata, and create BPEL and WSDL files with the process that openXdata needed. But the BPEL and WSDL files created with the Eclipse BPEL editor did not execute successfully with Apache ODE even though it was a software editor suggested for Apache ODE.

3.2.2.2 Testing BPEL solutions

During the work of implementing Sliver, example files were necessary for testing of much of the code, this is something that Sliver does not have available within the code or on its web site. Sliver it self does not include an editor for creating such files in any form, nor does it recommend any other software for this task. But we found a BPEL and WSDL "hello world" example in an Oracle tutorial, a tutorial for one of their Oracle products. These files were then modified so that the implementation of Sliver we had would accept it as input. Attempts at modifying the "hello world" example further were however unsuccessful and Sliver would not accept the manually modified XML files. Software for creating BPEL and WSDL files was needed. Eclipse with a BPEL plugin (BPEL 2010) was the most successful solution found. The plugin gave a graphical editor to Eclipse for creating and editing BPEL and WSDL files. This made creating the files easier, but not flawless. The editor still required some things to be written manually in Xpath for instance. The editor requires knowledge that at least the first time user usually will not have.

3.2.2.3 Sliver implementation example

Even though we never got Sliver to work as intended, work was done to implement it as mentioned previously. The code written started a server, and created a client that connected to that server. The server and client were both created from the same main

[29] http://ode.apache.org/index.html

class.

```
public static void main(String[] args) throws Exception {

        BPELServerConstructor bpel = new BPELServerConstructor(9001,
        "http://jbpm.org/examples/hello", "127.0.0.1", 9000,"hello.bpel");

        bpel.startServer();

        BPELClient client = new BPELClient("hellotest.xml", "127.0.0.1", 9001);
```

As seen in this code example, the server is first created by creating a new *BPELServerConstructor* object, then the client is created by making a *BPELClient* object. The client connects to the localhost (127.0.0.1) ip address since the server is started on the same computer.

The server is started with the following code.

```
public void startServer() throws Exception{
        String namespace = nameSpace;

        Transport transport = new SocketTransport(soapListenPort);

        BPELServer server = new BPELServer(transport);
        server.addProcess(namespace, new FileInputStream(bpelFile));

        server.bindIncomingLink("caller", namespace, "sayHello");

        Binding remoteHost = new SocketBinding(remoteIP, remotePort);
        server.bindOutgoingLink("ExternalSOAPServiceLink", remoteHost);

        server.start();
}
```

The class has a constructor that requires the variables necessary for this method, this code is left out here. When this code is executed, the main method code above starts the client.

```
private void getServerSOAPResponse() throws IOException{
        InputStream fis = new FileInputStream(xmlInputFile);
        BufferedReader bis = new BufferedReader(new InputStreamReader(fis));
        String read = null;
        StringBuffer bf = new StringBuffer();

        while ((read = bis.readLine()) != null)
        {
           bf.append(read).append('\n');
        }

        final Socket socket = new Socket(hostIP, hostPort);

        DataOutputStream out = new DataOutputStream(socket.getOutputStream());
        out.writeUTF(bf.toString());

        DataInputStream din = new DataInputStream(socket.getInputStream());

        SOAPResponse = din.readUTF();

        out.close();
        din.close();
        fis.close();
}
```

This method connects to the server just created, and opens a data stream. The stream is read as a String and saved in the SOAPResponse variable. The BPELClient class that this method resides in has a constructor that requires parameters and corresponding variables that are used by this method. The data stream the method gets from the server contains the BPEL document text from the “hello.bpel” file given to the server in the main method.

Since we were able to retrieve the XML document, an XML parser was implemented on the client side. That way, data could be extracted from the XML elements. This resulted

in code that could find an element by name, and then return the value within it. This approach did not use the existing methods within the Sliver code, and therefor in most ways contradicted with the purpose which was to implement Sliver.

3.3 Second Iteration

In the first iteration, Sliver and BPEL have been proved as not appropriate and did not fit the requirements well. Therefor using BPEL and BPEL based software was considered unsuitable for use with the openXdata project. We had also been unable to implement Sliver, editors were available, but not compatible with other software like Apache ODE for executing the business process file. Documentation for how to create BPEL and WSDL files was thin, and none seemingly existed for how to make tools work together. Even if BPEL could technically do what was needed, it would require too much time from people in the project to understand it. After the first iteration, the requirements were also refined.

3.3.1 Refined requirements

In the second iteration, more details of the requirements are identified.

The work definition document needs to describe a set of events. The events should contain data the following data:

- Template id. An identifier of a template within the existing openXdata system.
- ID. This is an local id used to identify an schema template within the document.
- Start Time. The time from which the schema can be preformed.
- Parent template. The templates that need to be completed in order to preform the current schema template.

These were some of the important requirements. These were the ones that it was decided to try and implement as a start. There were more ideas about what would be needed in the future, but they were put on hold temporarily.

The work data describes schemas and how far in the execution of them one is. Each

schema entry should contain the following data:

- Schema id. An identifier for a schema within the document.
- Work definition Id. A reference number to the corresponding template in the work definition document.
- Status. Is the schema finished or unfinished. Each status update, a schema can contain several, has a time stamp. This creates a history for each schema entry.

3.3.2 Solution with YAWL

After a new review of the possible software solutions that could be used, YAWL was the most promising. YAWL includes a graphical editor and a server for executing the workflows. YAWL is also a single package where everything is made to work together. The server software is an Apache Tomcat application with a web interface. This server software and interface, which worked in our tests, was however not suited for the intention of this master thesis. The workflow editor was however very well suited for our needs, all we needed was a software solution to run the workflows created in the editor. What was already being developed fitted this task very well, it was therefor decided to change some of the requirements, but in essence continue developing the workflow application.

The refreshed requirements then became as follows:

- YAWL workflow document.
  - Replaces the work definition document.
- Event Log XML document.
  - Replaces the work data document.
  - Based on an event log structure created by us.
- Java SE based application (Unchanged)
- Output what tasks to preform next (Unchanged)

These requirements are the basis for the final workflow application.

In the implementation, we use the YAWL editor to create workflows, save them into XML documents, match the XML documents with the event logs and generate next tasks based on the matching done by the software created in this master thesis.

3.4 A quick presentation of the patterns.

A workflow pattern is a model. It models the workflow in a certain way to achieve a certain functionality or behavior, many people model it the same way to achieve the same thing means they are following a pattern.

There are 2 categories of well accepted patterns, “Basic control flow” patterns and “Advanced branching and Synchronization” patterns. Both are shown on the YAWL web site[30] and on a separate web site workflow patterns[31].

The patterns tested were the following basic patterns:

- Sequence
- Parallel Split
- Synchronization
- Exclusive Choice
- Simple Merge

and a custom pattern:

- Nested split

The last pattern is not part of the basic pattern set, but I added it to show that the application can handle a nested split (a split within another split), and that they can have different split and join code pairs (the start and stop nodes must have matching split and join codes).

[30] http://www.yawlfoundation.org/resources/patterns
[31] http://www.workflowpatterns.com/patterns/control/

3.4.1 Sequence

Figure 4: Sequence pattern

The sequence pattern (Russell, Hofstede et al. 2006) consists of a sequence of tasks. A task can be done when the previous task is completed.

3.4.2 Parallel Split

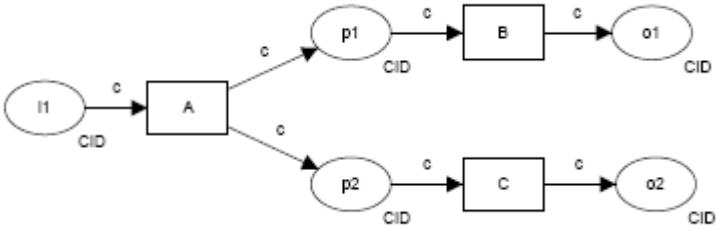

Figure 5: Parallel Split pattern

A parallel split (Russell, Hofstede et al. 2006) is a pattern that splits into two or more branches that are then executed concurrently. The task that splits, Task A in this illustration, will have AND as its split condition.

3.4.3 Synchronization

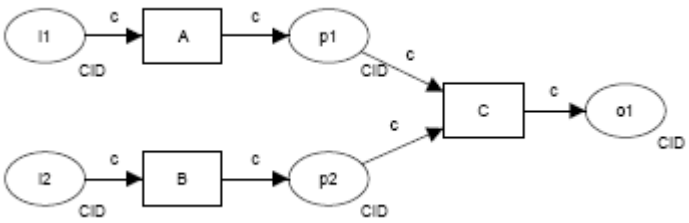

Figure 6: Synchronization pattern

The synchronization (Russell, Hofstede et al. 2006) pattern joins together two or more branches into a task so that subsequent tasks of the join task, Task C in this illustration,

can be preformed when the join condition has been met.

3.4.4 Exclusive Choice

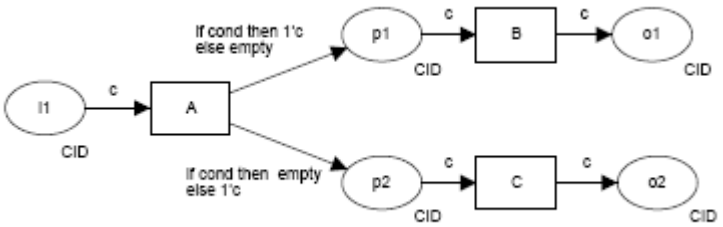

Figure 7: Exclusive Choice pattern

In the exclusive choice (Russell, Hofstede et al. 2006) pattern, a task splits into two or more branches where there is a mechanism in the split task so that only one of the branches are chosen. The different branches are considered different routes and separate paths in the workflow, and not concurrent tasks.

3.4.5 Simple Merge

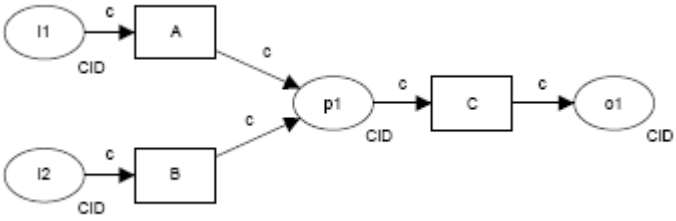

Figure 8: Simple Merge pattern

In the simple merge (Russell, Hofstede et al. 2006) pattern, two or more branches are converged into a single branch without considering synchronization. When either Task A or Task B is done, Task C will be enabled. It is allowed to complete all of the converging branches, but there is no control mechanism to check this, when one of the branch tasks are complete, the task after the convergence will be enabled, in this example, C.

3.4.6 Nested Split

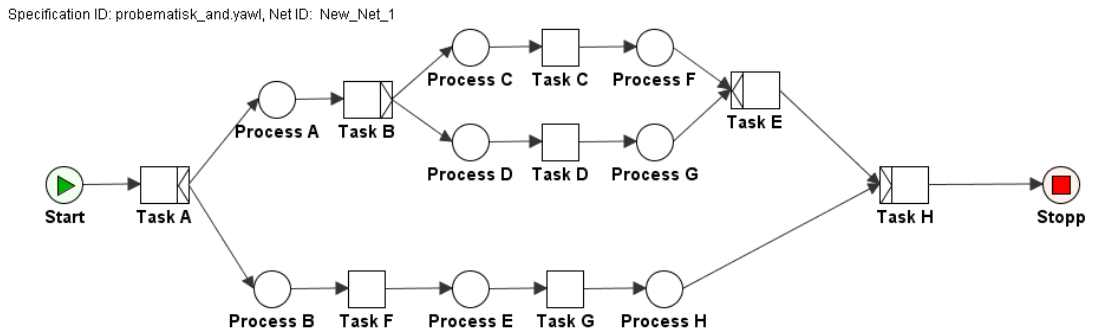

Figure 9: Nested Split pattern

The nested split pattern is created by me to show that the application handles a task that splits into two branches (or more, two is just an example), only to have one of the branches split again into two new branches within the first split.

The first split, Task A, has AND as its split condition, while the second split, Task B, has XOR as its split condition. This means that after the first split, not all tasks within it shall be preformed, either Task C or Task D is left out, even though the AND split condition initially means do all subsequent tasks, this however changes when the XOR condition is introduced within. This causes complexity when designing the application and its handling of split and join tasks, and the tasks within and evaluating what tasks are next and if tasks have been preformed according to the workflow design and rules.

3.5 System architecture

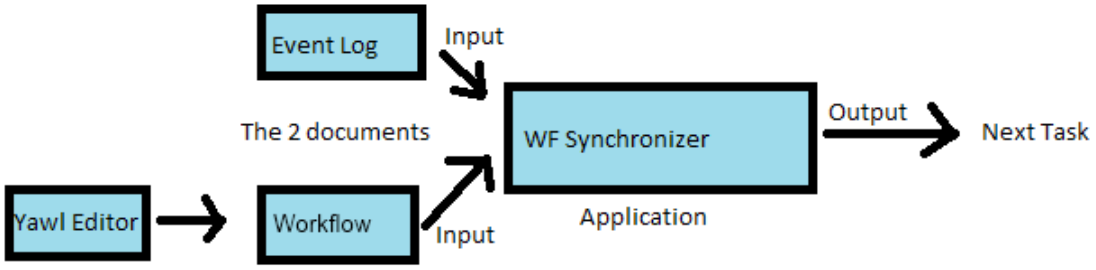

Figure 10: System Architecture

The Workflow Synchronizer is shown in Figure 10 as part of the system architecture and is labeled “WF Synchronizer”. The first part of the architecture is the “Yawl

Editor", it is a graphical editor for creating the workflow documents. The next part is the XML documents, the workflow document created in the YAWL Editor and an Event log document. The Event log document should in the future have a graphical interface for creating them, a text editor has however been used during this master thesis. These two documents are the input for the Workflow Synchronizer application created in this master thesis. This is the part of the system architecture that processes the documents, its output is the next task(s).

In the future, the "WF Synchronizer" should be included into the openXdata, this may change the system architecture.

3.6 Class diagram

In this class diagram (Figure 11), the yellow squares represent packages, and the white rectangles represent classes. The names within each of these figures corresponds with the actual package or class in the application code base.

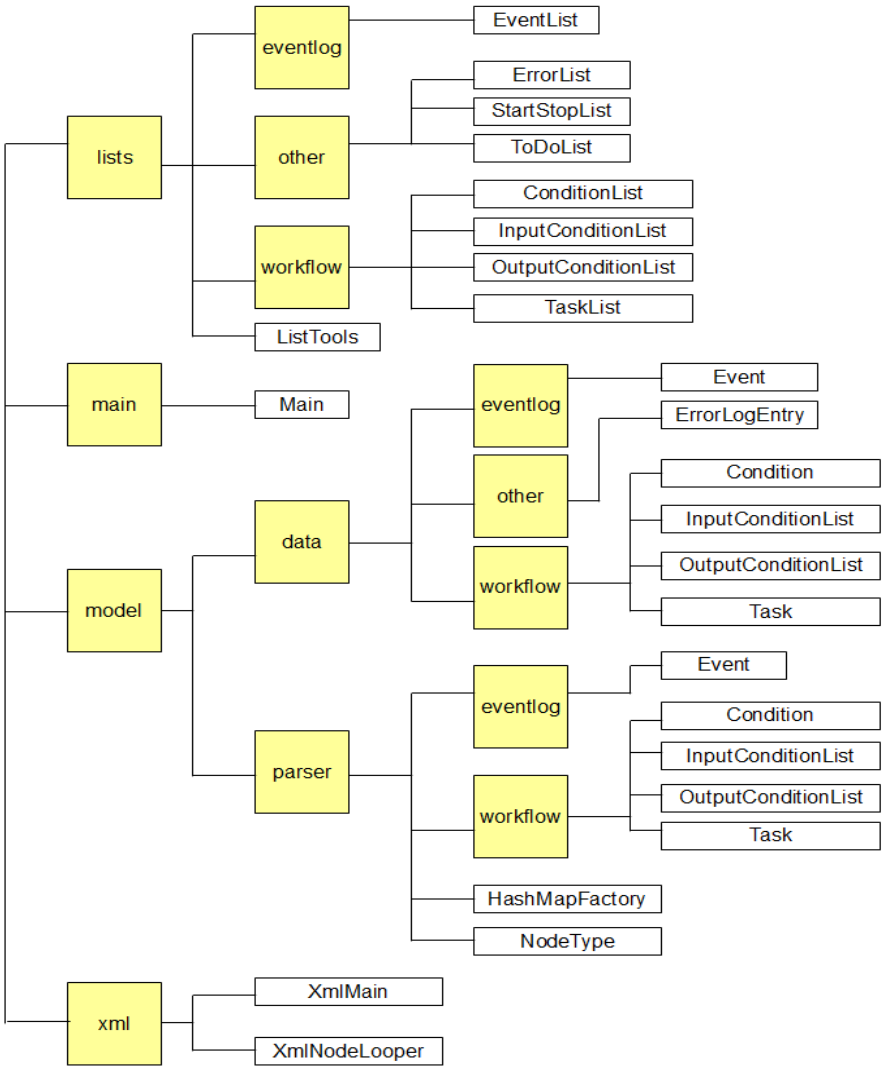

Figure 11: Application class diagram

When developing this application, "Eclipse IDE for Java Developers" was used as the coding tool. All the code for the workflow application is written in Java.

3.7 Creating test example

The workflow java application works with a YAWL workflow example modelling a

Clinical trial, and a corresponding event log file we designed. The Java application is able to parse both of these XML files, and detect some errors in the log files data. The next step is to create a new and entirely separate example, this means a YAWL workflow created in the YAWL Editior, and a corresponding manually written XML log file.

For test purpose, the cinema scenario was chosen. The YAWL cinema workflow and an event log were created.

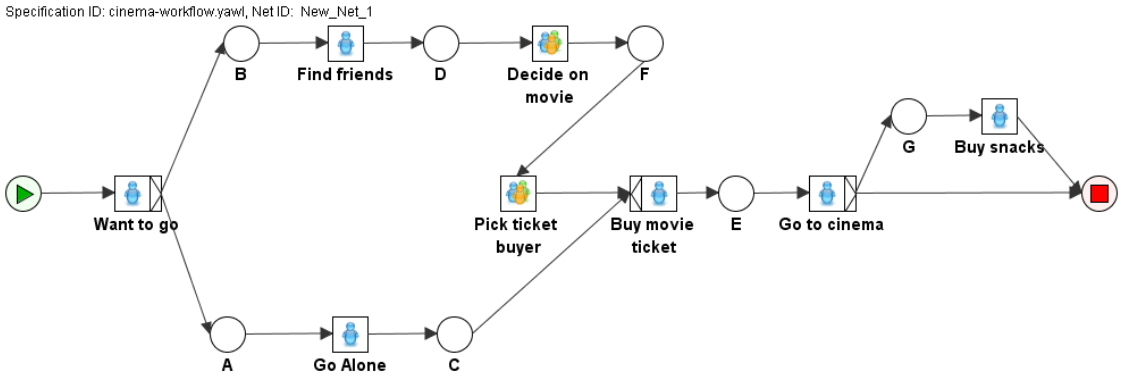

Figure 12: YAWL Cinema workflow model

Below is the corresponding XML log file for the workflow (Figure 12):

```
"<?xml version="1.0" encoding="UTF-8"?>
<eventlog>
        <event patientid='1'>
                <task>Want to go</task>
                <endtime>09:00</endtime>
        </event>

        <event patientid='1'>
                <task>Find friends</task>
                <endtime>09:01</endtime>
        </event>

        <event patientid='2'>
                <task>Want to go</task>
                <endtime>09:05</endtime>
        </event>

        <event patientid='2'>
                <task>Go Alone</task>
                <endtime>09:06</endtime>
```

```
</event>

<event patientid='2'>
        <task>Buy movie ticket</task>
        <endtime>09:06</endtime>
</event>

<event patientid='3'>
        <task>Want to go</task>
        <endtime>10:00</endtime>
</event>

<event patientid='4'>
        <task>Want to go</task>
        <endtime>10:05</endtime>
</event>

<event patientid='4'>
        <task>Find friends</task>
        <endtime>10:06</endtime>
</event>

<event patientid='4'>
        <task>Decide on movie</task>
        <endtime>11:00</endtime>
</event>

<event patientid='4'>
        <task>Pick ticket buyer</task>
        <endtime>11:03</endtime>
</event>

<event patientid='4'>
        <task>Buy movie ticket</task>
        <endtime>11:04</endtime>
</event>

</eventlog>"
```

The code where the files are referred to were changed to accommodate the test XML files:

```
"File workflow_file = new File("cinema-workflow.yawl");
//File workflow_file = new File("workflow.xml");
XmlMain workflow_parser = new XmlMain(workflow_file);
```

```
File eventlog_file = new File("cinema-eventlog.xml");
//File eventlog_file = new File("eventlog.xml");
XmlMain eventlog_parser = new XmlMain(eventlog_file);"
```

This allows the application to load the new XML files The application parsed the XML files successfully and gave the following output:

```
"Patients with invalid entries:
------------------------------
Patient id: 1
- Task name: Find friends
- End Time: 09:01
- Next tasks: Decide on movie,
------------------------------
Patient id: 2
- Task name: Buy movie ticket
- End Time: 09:06
- Next tasks: Go to cinema,
------------------------------
Patient id: 3
- Task name: Want to go
- End Time: 10:00
- Next tasks: Find friends, Go Alone,
------------------------------
Patient id: 4
- Task name: Buy movie ticket
- End Time: 11:04
- Next tasks: Go to cinema,
------------------------------"
```

During the test of this pattern, a flaw became clear in the software. It lacked a representation for stopping. If a log says that a person has come to the task of "`Go to cinema`", The next task will be "`Buy snacks`", but that person can also do nothing as in finished, but that does not show up in the output. It was therefor necessary to add the stop in the output, to represent the option of doing nothing, since one can be finished at that point. Stop is actually an output condition, while Start is an input condition. The workflow software was edited so that Start and Stop are both in some parts of the code handled as a Task, though an incomplete Task. Stop as a Task for instance does not have a list of tasks to flow into or a join condition, while Start does not have split condition.

4 Testing

4.1 Goal

The goal of testing is to show that the application code is capable of finding what tasks are to be preformed next.

4.1.1 Black-box vs White-box testing

Back-box testing (Bell 2005, p. 269-270) is testing without the knowledge of the internal structure of the test object. External descriptions of the test objects are used to create test cases. In this case, the relevant external description is of the software (specifications, requirements and design). The person who designs the test chooses what is valid and invalid input, and the tests can be functional or non-functional, the former is usually used.

White-box testing (Bell 2005, p. 272), also known as clear box testing among others, creates test cases where the internal structure is known. This kind of testing requires programming knowledge in order to identify paths in the software. Input test cases are chosen to test specific paths in the code, while the appropriate output is chosen to show the relevant results. If the code is rewritten, the tests may also have to change since they can be based on the actual implementation of the code. The tests can be implemented at different levels of detail, but creating tests that operate at a unit level could create an unmanageable amount of test cases. Unimplemented parts of the specification or requirements that are missing might not be uncovered with a detailed unit level test, but one will know that all the paths tested are executed.

When using black-box testing in software development, test cases are written towards high level classes and methods, and the test writer only knows the methods parameters, return type and intended use through external descriptions. This means that the tester does not know how the method or class works and what other methods and code is executed. An input is given to this “black-box”, and a result is returned. A white-box test however in software development intentionally writes test cases while having

access to the code. The test is based on the specific implementation of the code, not only on high level classes, but also low level classes. What this means is that the black-box testing could have a test that calls a “main” method, inputs the required data, and validates the output, while a white-box test could write a specific test for a number manipulation method, or a method that runs a database query.

The problem with the black-box testing is that coding errors can potentially stay hidden if the test cases are not diverse enough. Code with errors might work most of the time, and only “crash” in certain cases. White-box testing has the opposite problem, the tests can more or less make sure the code is without flaws, but the number of possible tests that can be run on a large code base can be overwhelming.

In this master thesis, black-box testing is used because it is not important how the internal structure is setup and works, what's important is if the application can take input and produce the correct output. When the application receives an input, the output from the execution is collected and evaluated. This adheres to the way black-box testing is done. There are several black-box techniques that exists, but they are not very applicable to this workflow application.

4.2 Testing plan

When testing the application code, it is important to know what to test for. In this case, testing that the application supports certain workflow patterns is important. Correctly parsing the documents is also crucial, but that is not the focus of this experiment, it is also indirectly demonstrated by the fact that the correct data is used and outputted.

4.3 Test setup

The test setup has two input file variables for each time the application executed, and the application is run once per workflow and event log document pair. The application then automatically parses the two documents and outputs what tasks to do next.

Coding and run time software: *Eclipse IDE for Java Developers*

Coding language: *Java version 6*

workflow file format: *YAWLEditor 2.0.1 Specification file (XML)*

Event Log file format: *A custom XML setup.*

All the workflow and event log file pairs were pre entered into the application code, but commented out, so the parser does not parse them when it executes the code.

The application was then executed from Eclipse once for each of the patterns to test, and for each execution, the pattern that was tested had its file references un commented before the execution. It was then commented out again so not to disturb the next execution. The first pattern tested was the Sequence pattern, and the next pattern to test was the Parallel Split, part of the code during the first execution then looked like this:

```
/**Sequence**/
File workflow_file = new File("DEMO_Patterns/sequence_workflow.yawl");
File eventlog_file = new File("DEMO_Patterns/sequence_eventlog.xml");

/**Parallel Split**/
//File workflow_file = new
File("DEMO_Patterns/paralell_split_workflow.yawl");
//File eventlog_file = new
File("DEMO_Patterns/paralell_split_eventlog.xml");
```

These variables are used by the application as its input, there is not implemented any system for designating input files at this point. The XML documents are then parsed by the XmlMain class, it is called once for each file.

```
new XmlMain(workflow_file);
new XmlMain(eventlog_file);
```

The XML data is now available in a few static Java classes, it is then validated before being used to determine what tasks should be completed next for each patient. The results are outputted as text in the Eclipse console. Here is a partial example from the sequence pattern:

```
--------------------------------
-         To Do Lists          -
--------------------------------
```

```
--- Patient 1's next task(s) ---------------
Mandatory tasks: Stop
Optional tasks:
One mandatory per [...]:
----------------------------------------

--- Patient 2's next task(s) ---------------
Mandatory tasks: Task B
Optional tasks:
One mandatory per [...]:
----------------------------------------
```

These results and the information from the event log were then entered manually into the tables in the next chapter.

4.4 Results

The results are presented with a heading, followed by the workflow diagram and a table with the results. The diagram has 3 column:

- The first column is Patient ID which has the patient identifier.
- The second column shows the event log, it has a table of what tasks have been completed.
- The third column shows the results from the application, "what tasks should be preformed next?".

Each row in the table belongs to a patient, represented by the patient id in the beginning.

4.4.1 Testing the patterns

4.4.1.1 Sequence

Figure 13: YAWL workflow - Sequence pattern

Table 1: Sequence event log and application results

<table>
<tr><th>Patient ID</th><th>Event Log / Tasks done</th><th>Application result / Tasks to do</th></tr>
<tr><td>1</td><td><table><tr><th>Task Name</th><th>End Time</th></tr><tr><td>Task A</td><td>10:00</td></tr><tr><td>Task B</td><td>10:01</td></tr></table></td><td><table><tr><th>Mandatory</th><th>Optional</th><th>One mandatory per bracket [ ]</th></tr><tr><td>Stop</td><td>…</td><td>...</td></tr></table></td></tr>
<tr><td>2</td><td><table><tr><th>Task Name</th><th>End Time</th></tr><tr><td>Task A</td><td>10:05</td></tr></table></td><td><table><tr><th>Mandatory</th><th>Optional</th><th>One mandatory per bracket [ ]</th></tr><tr><td>Task B</td><td>…</td><td>...</td></tr></table></td></tr>
<tr><td>3</td><td><table><tr><th>Task Name</th><th>End Time</th></tr><tr><td>Task B</td><td>11:05</td></tr></table></td><td>Invalid Event Log

Task B is preformed before Task A, which is not allowed.</td></tr>
</table>

In the sequence pattern, the tasks are preformed one after the other as a sequence in the order they are setup. In the pattern used in this test, it means that Task A should be preformed first, then Task B when Task A is completed. The default split condition is XOR, in YAWL, and each task only flows into one other task, so there are only one way of completing this pattern.

Patient 1 and 2 have been completed according to the workflow definition so far, and is therefor given the next tasks to do respectively. Patient 1 can do the task (or output condition) Stop, and patient 2 can do Task B. This is correct.

Patient 3 however is logged as having completed Task B, this is not allowed since the first task to do when one start is Task A, and one have to complete it before moving to the next task, Task B. Patient 3's event log is therefor invalid, and does not receive a

new task to preform.

4.4.1.2 Parallel Split

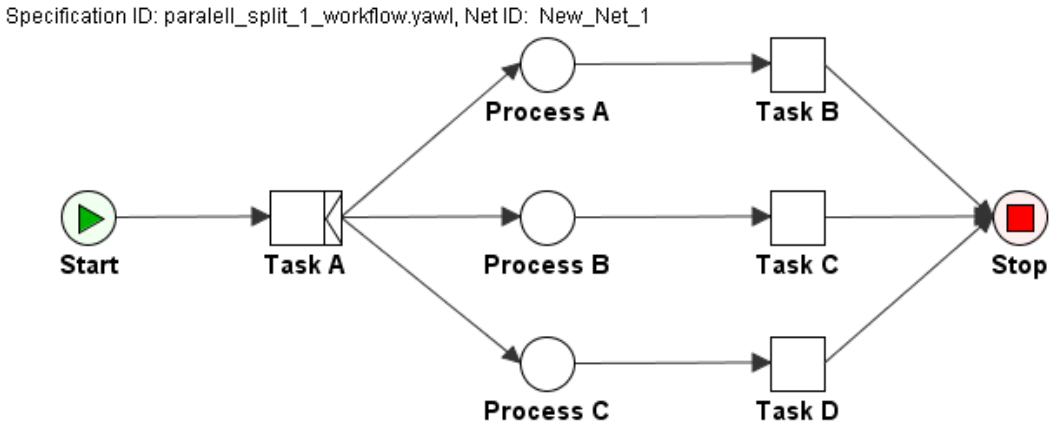

Figure 14: YAWL workflow - Parallel Split pattern

Table 2: Parallel Split event log and application results

<table>
<tr><th>Patient ID</th><th>Event Log / Tasks done</th><th>Application result / Tasks to do</th></tr>
<tr><td>1</td><td>
<table>
<tr><th>Task Name</th><th>End Time</th></tr>
<tr><td>Task A</td><td>10:00</td></tr>
<tr><td>Task B</td><td>10:01</td></tr>
<tr><td>Task C</td><td>10:02</td></tr>
</table>
</td><td>
<table>
<tr><th>Mandatory</th><th>Optional</th><th>One mandatory per bracket [ ]</th></tr>
<tr><td>Task D</td><td>…</td><td>...</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>
<table>
<tr><th>Task Name</th><th>End Time</th></tr>
<tr><td>Task A</td><td>10:05</td></tr>
</table>
</td><td>
<table>
<tr><th>Mandatory</th><th>Optional</th><th>One mandatory per bracket [ ]</th></tr>
<tr><td>Task B, Task C, Task D</td><td>…</td><td>...</td></tr>
</table>
</td></tr>
<tr><td>3</td><td>
<table>
<tr><th>Task</th><th>End Time</th></tr>
</table>
</td><td>
<table>
<tr><th>Mandatory</th><th>Optional</th><th>One</th></tr>
</table>
</td></tr>
</table>

<table>
<tr>
<td></td>
<td>
<table>
<tr><td>Name</td><td></td></tr>
<tr><td>Task A</td><td>11:00</td></tr>
<tr><td>Task B</td><td>11:05</td></tr>
<tr><td>Task C</td><td>11:15</td></tr>
<tr><td>Task D</td><td>11:20</td></tr>
</table>
</td>
<td>
<table>
<tr><td></td><td></td><td>mandatory per bracket [ ]</td></tr>
<tr><td>Stop</td><td>...</td><td>…</td></tr>
</table>
</td>
</tr>
<tr>
<td>4</td>
<td>
<table>
<tr><td>Task Name</td><td>End Time</td></tr>
<tr><td>Task B</td><td>11:05</td></tr>
</table>
</td>
<td>Invalid Event Log

Task B is done before Task A, this is not allowed.</td>
</tr>
</table>

The parallel split pattern splits on Task A with the condition AND, which means all three branches have to be completed. In the test workflow example, the first task is the one that splits, and all tasks ends in the Stop output condition. The model could have been made differently, and that could mean that not all tasks had to be completed, in this model however, all tasks are to be completed for the workflow to be complete.

Patient 1,2 and 3 all have completed their tasks according to the workflow definition, and are receiving their next tasks respectively.

Patient 4 however has preformed Task B before Task A, this is not allowed, Task A is the first task to complete.

It should be noted that even though Task A is the first task to complete, it does not matter in what order its following tasks (Task B,C and D) are preformed. Each branch could also have has a sequence of tasks, and one could have completed a sequence before preforming the next. Both points are demonstrated in the last custom pattern that is not part of the basic patterns.

4.4.1.3 Synchronization

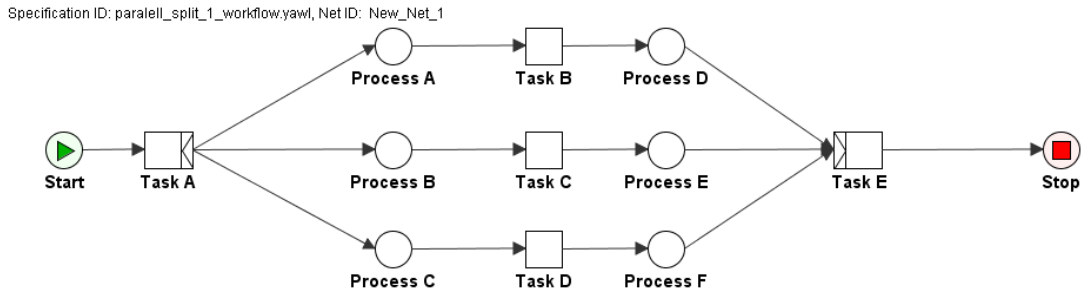

Figure 15: YAWL workflow - Synchronization pattern

Table 3: Synchronization event log and application results

<table>
<tr><th>Patient ID</th><th>Event Log / Tasks done</th><th>Application result / Tasks to do</th></tr>
<tr><td>1</td><td>
<table>
<tr><th>Task Name</th><th>End Time</th></tr>
<tr><td>Task A</td><td>10:00</td></tr>
<tr><td>Task B</td><td>10:01</td></tr>
</table>
</td><td>
<table>
<tr><th>Mandatory</th><th>Optional</th><th>One mandatory per bracket [ ]</th></tr>
<tr><td>Task C, Task D</td><td>…</td><td>...</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>
<table>
<tr><th>Task Name</th><th>End Time</th></tr>
<tr><td>Task A</td><td>10:05</td></tr>
<tr><td>Task B</td><td>10:06</td></tr>
<tr><td>Task D</td><td>10:07</td></tr>
</table>
</td><td>
<table>
<tr><th>Mandatory</th><th>Optional</th><th>One mandatory per bracket [ ]</th></tr>
<tr><td>Task C</td><td>…</td><td>...</td></tr>
</table>
</td></tr>
<tr><td>3</td><td>
<table>
<tr><th>Task Name</th><th>End Time</th></tr>
<tr><td>Task A</td><td>11:00</td></tr>
<tr><td>Task B</td><td>11:01</td></tr>
<tr><td>Task C</td><td>11:02</td></tr>
<tr><td>Task D</td><td>11:03</td></tr>
</table>
</td><td>
<table>
<tr><th>Mandatory</th><th>Optional</th><th>One mandatory per bracket [ ]</th></tr>
<tr><td>Stop</td><td>...</td><td>…</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td><table><tr><td>Task E</td><td>11:05</td></tr></table></td><td></td></tr>
<tr><td>4</td><td><table><tr><th>Task Name</th><th>End Time</th></tr><tr><td>Task A</td><td>12:00</td></tr><tr><td>Task B</td><td>12:05</td></tr><tr><td>Task E</td><td>12:08</td></tr></table></td><td>Invalid Event Log

Task E is preformed without all the previous tasks Task E joins together being done.</td></tr>
</table>

The synchronization pattern joins on Task E with the condition AND, that is the synchronization. All 3 branches meet at Task E, and because of its join condition, all have to be completed in order for Task E to be complete and move to the next task. The next task in this case is the output condition Stop. This pattern however also has a parallel split, this is because the 3 branches can not all start from the Start input condition. It is therefor necessary with a split in order for a join to be needed. The split condition in Task A is AND to match the join condition in Task E.

Patient 1,2 and 3 all completes the tasks in an order that adheres to the workflow specification. Notice that patient 2 completes Task D before preforming Task C, there is no order between the branches.

Patient 4 however does not preform the tasks in a legal manner. The synchronization patterns purpose is to make sure all the different branches completes before the moving on past the joining task, in this case Task E. Patient 4 preforms Task E while only having completed Task A and B, this leaves Task C and D uncompleted, both who are mandatory tasks to preform for the join in Task E to be valid.

4.4.1.4 Exclusive Choice

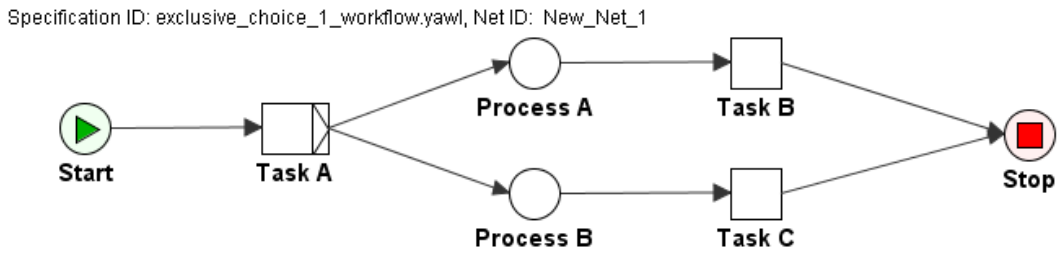

Figure 16: YAWL workflow - Exclusive Choice pattern

Table 4: Exclusive Choice event log and application results

<table>
<tr><th>Patient ID</th><th>Event Log / Tasks done</th><th>Application result / Tasks to do</th></tr>
<tr><td>1</td><td>
<table>
<tr><th>Task Name</th><th>End Time</th></tr>
<tr><td>Task A</td><td>10:00</td></tr>
<tr><td>Task B</td><td>10:01</td></tr>
</table>
</td><td>
<table>
<tr><th>Mandatory</th><th>Optional</th><th>One mandatory per bracket [ ]</th></tr>
<tr><td>Stop</td><td>…</td><td>...</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>
<table>
<tr><th>Task Name</th><th>End Time</th></tr>
<tr><td>Task A</td><td>10:05</td></tr>
<tr><td>Task B</td><td>10:06</td></tr>
<tr><td>Task C</td><td>10:07</td></tr>
</table>
</td><td>Invalid event log

Task A has an exclusive OR split condition, so only one of the Tasks it splits into can be preformed. However, both tasks are done in the event log, Task B and Task C.</td></tr>
<tr><td>3</td><td>
<table>
<tr><th>Task Name</th><th>End Time</th></tr>
<tr><td>Task A</td><td>11:00</td></tr>
</table>
</td><td>
<table>
<tr><th>Mandatory</th><th>Optional</th><th>One mandatory per bracket [ ]</th></tr>
<tr><td>...</td><td>...</td><td>[Task B ^ Task C]</td></tr>
</table>
</td></tr>
</table>

The exclusive choice pattern splits on Task A with the condition XOR. This condition means that the user have to choose one, and only one of the branches, no more, no less. A task that comes after it is in other words mandatory, but the user gets to choose which one. In this example, Task B and C are the two optional branches to choose between.

Patient 1 and 3 have completed their tasks correctly, following the workflow design. Patient 1 have completed Task A and B, and is therefor finished as the Stop output condition implies. Patient 3 have to choose between Task B or C, and must preform only one of them.

Patient 2 has an invalid event log. Both Task B and C are completed after completing Task A, this is as stated not allowed since the split condition is XOR. It would also be illegal to skip them both.

4.4.1.5 Simple merge

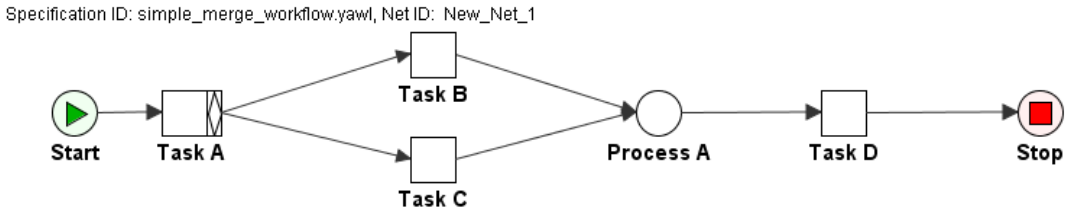

Figure 17: YAWL workflow - Simple merge pattern

Table 5: Simple Merge event log and application results

Patient ID	**Event Log / Tasks done**		**Application result / Tasks to do**		
1	**Task Name**	**End Time**	**Mandatory**	**Optional**	**One mandatory per bracket [ ]**
	Task A	10:00			
	Task B	10:01			
			Task D	Task C	...
2	**Task**	**End Time**	**Mandatory**	**Optional**	**One**

<table>
<tr><td></td><td><table><tr><td>Name</td><td></td></tr><tr><td>Task A</td><td>10:05</td></tr></table></td><td><table><tr><td></td><td></td><td>mandatory per bracket [ ]</td></tr><tr><td>…</td><td>Task B, Task C</td><td>...</td></tr></table></td></tr>
<tr><td>3</td><td><table><tr><td>Task Name</td><td>End Time</td></tr><tr><td>Task A</td><td>11:00</td></tr><tr><td>Task B</td><td>11:05</td></tr><tr><td>Task C</td><td>11:10</td></tr><tr><td>Task D</td><td>11:15</td></tr></table></td><td><table><tr><td>Mandatory</td><td>Optional</td><td>One mandatory per bracket [ ]</td></tr><tr><td>Stop</td><td>...</td><td>…</td></tr></table></td></tr>
<tr><td>4</td><td><table><tr><td>Task Name</td><td>End Time</td></tr><tr><td>Task A</td><td>12:00</td></tr><tr><td>Task D</td><td>12:05</td></tr></table></td><td>Invalid Event Log

Task D is done without Task B or Task C being done. Task A uses an OR split, but one can not skip Task B and Task C and do Task D even if the OR condition strictly speaking allows nothing to be done.</td></tr>
</table>

The simple merge pattern has the tasks converge at a single process. This means that there are no join or split conditions involved. When Task B or C are completed, Task D is enabled. There is no synchronization. One can however not skip Task B or C and go directly from Task A to Task D.

Patient 1,2 and 3 have valid event logs. Patient 1 have completed only Task A and B, and can then start on Task D or Task C which is optional, patient 3 however has completed all the tasks, and is finished. Both are allowed in the simple merge.

Patient 4 however completes Task D without having completed either Task B or C, this is not allowed. This diagram is in many ways the same as a sequence, but one gets a choice with two of the tasks whether one wants to complete both or only one of them,

but they still have to be completed in the order they are setup.

4.4.1.6 Nested Split

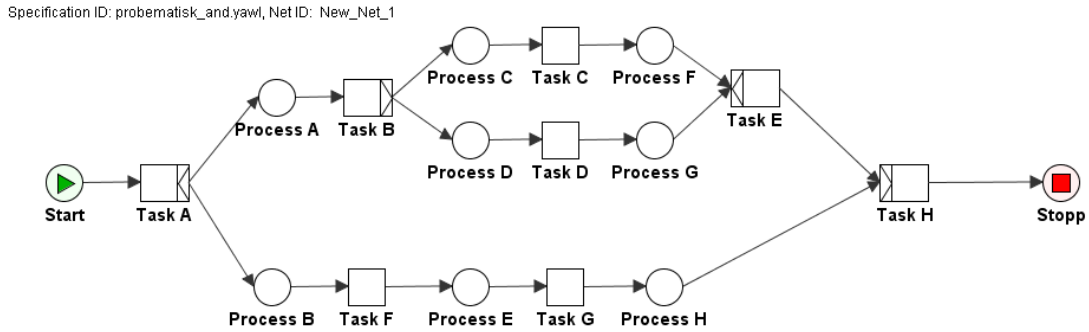

Figure 18: YAWL workflow - Nested Split pattern (Custom made)

Table 6: Nested Split event log and application results

<table>
<tr><th>Patient ID</th><th>Event Log / Tasks done</th><th>Application result / Tasks to do</th></tr>
<tr><td>1</td><td>
<table>
<tr><th>Task Name</th><th>End Time</th></tr>
<tr><td>Task A</td><td>10:00</td></tr>
</table>
</td><td>
<table>
<tr><th>Mandatory</th><th>Optional</th><th>One mandatory per bracket [ ]</th></tr>
<tr><td>Task B, Task F</td><td>...</td><td>...</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>
<table>
<tr><th>Task Name</th><th>End Time</th></tr>
<tr><td>Task A</td><td>11:00</td></tr>
<tr><td>Task B</td><td>11:05</td></tr>
</table>
</td><td>
<table>
<tr><th>Mandatory</th><th>Optional</th><th>One mandatory per bracket [ ]</th></tr>
<tr><td>Task F</td><td>…</td><td>[Task C ^ Task D]</td></tr>
</table>
</td></tr>
<tr><td>3</td><td>
<table>
<tr><th>Task Name</th><th>End Time</th></tr>
<tr><td>Task A</td><td>12:00</td></tr>
<tr><td>Task F</td><td>12:01</td></tr>
</table>
</td><td>
<table>
<tr><th>Mandatory</th><th>Optional</th><th>One mandatory per bracket [ ]</th></tr>
<tr><td>Task B, Task</td><td>...</td><td>…</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td></td><td><table><tr><td>G</td><td></td><td></td></tr></table></td></tr>
<tr><td>4</td><td><table><tr><th>Task Name</th><th>End Time</th></tr><tr><td>Task A</td><td>13:00</td></tr><tr><td>Task B</td><td>13:01</td></tr><tr><td>Task C</td><td>13:02</td></tr></table></td><td><table><tr><th>Mandatory</th><th>Optional</th><th>One mandatory per bracket [ ]</th></tr><tr><td>Task F, Task E</td><td>...</td><td>...</td></tr></table></td></tr>
<tr><td>5</td><td><table><tr><th>Task Name</th><th>End Time</th></tr><tr><td>Task A</td><td>14:00</td></tr><tr><td>Task B</td><td>14:05</td></tr><tr><td>Task C</td><td>14:10</td></tr><tr><td>Task D</td><td>14:15</td></tr></table></td><td>Invalid Event Log

Both Task C and Task D are done, but Task B splits with exclusive OR split, so only one of them should be preformed.</td></tr>
</table>

The nested split is included to demonstrate that the application is able to handle a split and join within another split and join. The first split, Task A, has the condition AND, while the next split, Task B, has the split condition XOR. What this means is that all tasks in the workflow except for Task C and D have to completed, and that only one of the exception tasks, Task C and D, can be completed.

Patient 1, 2, 3 and 4 completes their tasks according to the workflow definition. Notice that patient 4 completes Tasks A, B and C without having completed Task F yet, even though it is a mandatory task. The patient can also complete Task E before having to do Task F. Each branch can be completed before having to start preforming tasks on a new branch.

Patient 5 however completes both Task C and D. Task A has an AND condition that means complete all the following tasks that branch from me until a join or convergence, but the XOR split on Task B changes this for the tasks between Task B and E. For those tasks, the user only gets to choose one of them. This means that the event log and the way patient 5 has completed its tasks is invalid.

4.5 Summary and discussion

In the next sections, discoveries during development and testing of the workflow application are discussed. Some of them are problems that were solved in order to achieve the basic functionality we wanted, and other problems are declared and discussed, but not solved in the version of the program delivered within this master thesis.

4.5.1 How to handle split and join conditions

When I was working on the code for the application, I realized a problem, how is the or split code suppose to be handled in certain situations. “or” in programming can often mean that one have to choose one or more of the choices the or offers. “xor” means that one have to choose only one of the possibilities.

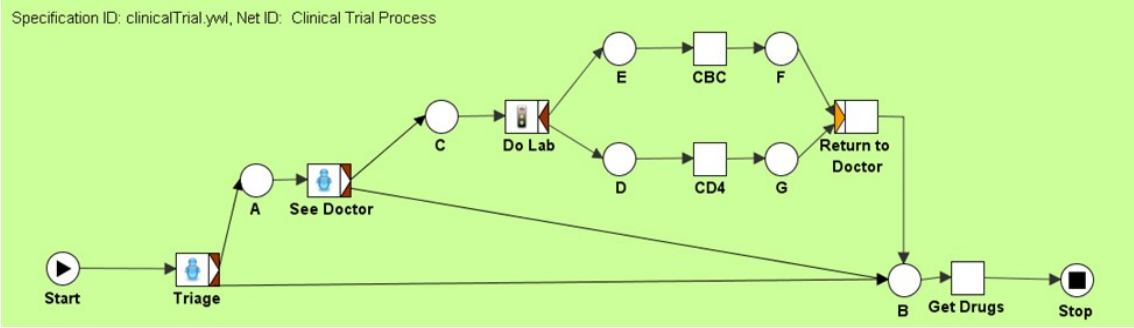

Figure 19: Clinical Trial workflow example

The problem discovered was how one handle “or” conditions in tasks that are further down the hierarchy then where one is currently. Say the user are preforming the tasks in the workflow illustration above, and the user have reached the task “Do Lab” (a square figure). The next two tasks are then “CBC” and “CB4”, and both since the “Do Lab” has “and” as its split condition. However. if “Triage” had “or” as its split condition, then strictly speaking, “Get Drugs” would also be a possible next task, since the “or” split condition allows (traditionally) one to also do the tasks one did not choose. This means that the next tasks are really “CBC, CD4 and Get Drugs”, and one can choose to do “Get Drugs” before, after or in between the “CBC” and “CD4” tasks.

This has a huge impact on not just how to code the application, but also how one should design the workflow. It is an important issue to have a clear policy on how to handle. One can simply say that the only tasks one can choose from are the ones on the same level, “CBC” and “CD4” are on the same level, “See Doctor” and “Get Drugs” are also on the same level when the “Triage” task is the root. Another possibility is to treat “or” as “xor”, this removes the problem all together. This last solution is what we initially decided to do during development.

4.5.2 Checking previously undone tasks

While I was rewriting the code that checks for undone tasks, I realized a problem. How far back in the node graph should I check for undone tasks? I could just check for undone tasks on the last split node behind the last node in the event log. This is however not without issues since a split could contain another split within it, and if both splits contain AND split code, it would mean that all tasks within the first, and therefor second, split would have to be done. The safe thing will therefor be to check all the tasks in the event log for splits, check their split codes and based on this list what tasks need to be or can be done. This is the way the final version of the Workflow Synchronizer handles this problem.

4.5.3 How many tasks can “Start” flow into

When testing a pattern, the simple merge pattern, I became aware of an unresolved modeling situation. I had tried to model the patterns as correctly as possible according to the models on the website[32], and with the simple merge pattern, I therefor ended up modeling it so that the “Start” item flowed into Task A and Task B as modeled below in Figure 20. The problem is that the Start item does not have a split (and join) code, and therefor does not describe how its split should be handled. The YAWL modeling tool does allow it however. What I've decided with the knowledge I have about how the workflows work is that this is not allowed to do. The way it is modeled below, or if Task A and Task B would both flow into the “Stop” item, leaves us with no information about

[32] http://www.workflowpatterns.com/patterns/control/

what tasks should be done or not. Can one do both, or only one of them? These questions have no apparent answer in these two scenarios.

The solution is therefor to limit the allowed tasks Start can flow into to 1, both as a maximum and minimum. Any other scenario is either problematic as described above, or useless since it would not have a starting point. Also it can not flow directly into the "Stop" item since this also would serve no purpose as none of these two items contains any other information than where the workflow model starts and stops.

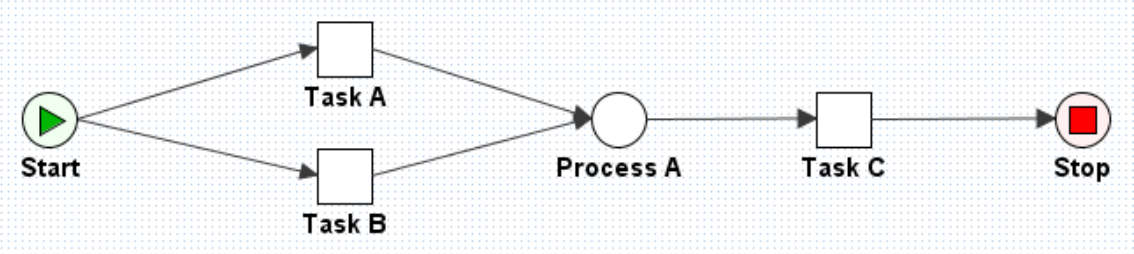

Figure 20: A "Start" with two tasks it flows into.

4.5.4 Composite tasks & duplicate names

When I modeled the sketch in the YAWL editor for the "Follow up study" scenario, I encountered a problem and realized flaw in my implementation.

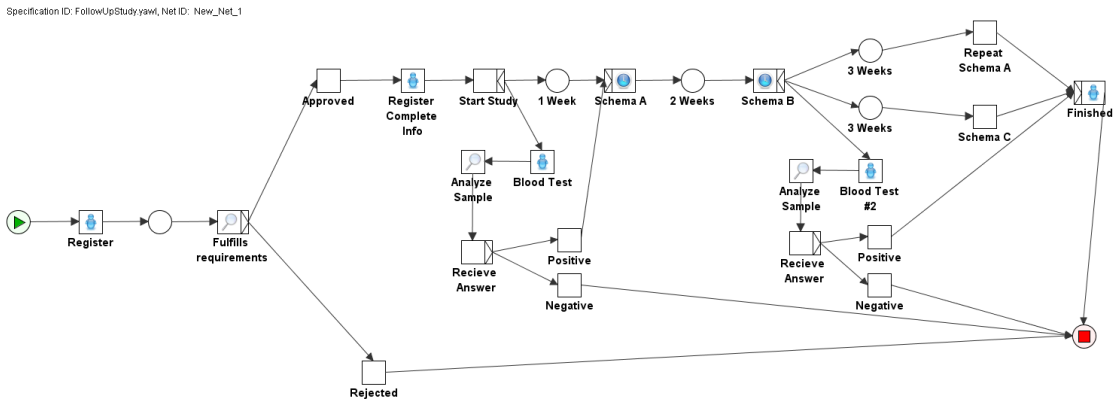

Figure 21: Followup Study workflow example

In Figure 21, one can see that the workflow has two blood tests, both of these are the

same blood test, but I found it difficult to reuse the workflow, so I made a duplicate, which starts with the “Blood Test #2” task. When reviewing an example pattern for YAWL, it became clear to me that a composite task could be a solution, I was not sure, but it was in any case a part of the modeling language that I had not implemented. To simplify things, I implemented the Followup study by remaking that part of the workflow as stated earlier. But this also put some light on another issue. The event log only contains the names of the tasks, not the id which is unique, and when duplicating the blood test pattern, I ended up with tasks that had the same names as other tasks.

The software can be altered to support composite tasks, which should not be to difficult. Also going from using the names in the event log, to using the id's is possible with some altering of the code. But even if these changes are not implemented, it should still support the basic workflow patterns.

5 Conclusions and Future Work

The goal of this thesis is to find a solution that could fill a functional need in openXdata. openXdata software is used to create and conduct surveys and studies, but it lacks a feature that would ensure the data from surveys and studies to be correct and consistent with the predefined workflow.

BPEL and its software was found to be difficult use, develop and did not satisfy the requirements. YAWL however did satisfy the requirements and was easy to use, but the execution engine and its web interface was found to be unsuited for use with openXdata. We therefor created our own software that would bridge the gap between what YAWL could offer and what openXdata needed. This resulted in the workflow application called Workflow Synchronizer. The name stems from process of synchronizing the event log file with the workflow file.

The Workflow Synchronizer was tested against five basic patterns and a custom-designed pattern.

The workflow application was able to successfully run the patterns created in the YAWL Editor. All outputs from the tests completed are as expected. There are parts of the workflow language that has not been implemented, but these are known and were left out on purpose.

The workflow patterns tested and their corresponding event log files were processed as intended. The event log documents had mistakes or errors in them that were put there intentionally, and the workflow application was able to detect them and handle them. That meant excluding the affected patients from the results when one or more of their log entries had errors in them. The tests show that the workflow application is able to return what tasks to do next and what tasks have already been completed.

There was much testing done during development, and those tests uncovered errors and flaws continually as the development progressed. These test were done, as the entire development, in an agile fashion, with no more documentation than needed, in iterations as the code matured and with new test patterns and scenarios as they became necessary.

5.1 Future work

Although the application passes the tests we performed, implementation of it into the openXdata is likely to uncover missing features. Actual use of the application will also show whether the advanced patterns are necessary or not. But as a standalone generic application, the Workflow Synchronizer should eventually support all the patterns and support the YAWL workflow files completely.

Support for composite tasks is one feature that is missing. Composite tasks are tasks that represent themselves a workflow. In YAWL, when adding a composite task to a workflow, one would also create a new workflow net and link the composite task to that net. Performing the composite task would then involve completing the workflow net, which has its own set of tasks and could also have a composite task as one of them. Future work would also involve creating some sort of framework around the Workflow Synchronizer. It needs an interface to become user friendly, this could be achieved through integration with openXdata and extending the existing interface to support workflows. This would also be the natural place to create and edit event log.

6 References

Aalst, W. M. P. v. d. (2004). "Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management."

Aalst, W. M. P. v. d. (2005). "Business Alignment: Using Process Mining as a Tool for Delta Analysis and Conformance Testing."

Alonso, G., R. Günthör, et al. (1996). "Exotica/FMDC: A Workflow Management System for mobile and disconnected clients." **4**(3).

Bahrami, A., C. Wang, et al. (2006). "The Workflow Based Architecture for Mobile Information Access in Occasionally Connected Computing."

Bell, D. (2005). Software Engineering for Students, Addison-Wesley.

Dongen, B. F. v., A. K. A. d. Medeiros, et al. (2005). "The ProM Framework: A New Era in Process Mining Tool Support." **3536/2005**.

Hackmann, G., M. Haitjema, et al. (2006). Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. Service-Oriented Computing – ICSOC 2006, Springer Berlin / Heidelberg. **4294/2006:** 503-508.

Hollingsworth, D. (1995). "Workflow Management Coalition - The Workflow Reference Model." (1.1).

Jing, J., K. Huff, et al. (1999). "Workflow and Application Adaptations in Mobile Environments."

Leymann, F., D. Roller, et al. (2003). "Goals of the BPEL4WS Specification."

Russell, N., A. H. M. t. Hofstede, et al. (2006). "Workflow Control-Flow Patterns - A Revised View."

Thatte, S., T. Andrews, et al. (2003). "Business Process Execution Language for Web Services."

Weske, M. (2007). Business Process Management - Concepts, Languages, Architectures, Springer.