ulm university universität
uulm

Faculty of
Engineering
and Computer Science
Institute of Databases and
Information Systems

Process Automation and Optimization for the Audit Committee

Bachelor Thesis at Ulm University

Submitted by:
Kevin Andrews
kevin.andrews@uni-ulm.de

Reviewer:
Prof. Dr. Manfred Reichert

Supervisor:
Jens Kolb

2012

Version May 7, 2012

Compiled with: PDF-LaTeX 2ε

Contents

1 Introduction **1**

1.1 The Audit Committee and its Tasks 1

1.2 Process Optimization Potential with BPMS 2

2 Current Processes of the Audit Committee **5**

2.1 Documentation of the Different Processes 5

2.1.1 BAföG Funding Approval 6

2.1.2 ECTS Credit Acknowledgment 7

2.1.3 Changing of Major 10

2.1.4 Attribution of Course Credits 11

2.1.5 Work Experience Acknowledgment 14

2.1.6 Thesis Deadline Extension 15

3 Optimization of the Thesis Deadline Extension Process **19**

3.1 Optimization Changes to the TDE Process 19

3.2 Creating the Process Model in the AristaFlow BPM Suite 22

3.3 Planning the Implementation 24

4 Implementation **25**

4.1 Implementing an AristaFlow Component 25

4.1.1 Process Parameters 26

4.1.2 Activity Configurations 27

4.1.3 Multiple Optional Parameters 29

4.1.4 Debugging a Component 30

4.1.5 Exception Handling 30

4.2 PDF Printer Component 31

4.3 PDF Filler Component 32

4.4 Encrypted Mailer Component 33

4.5 PDF Signer Component 34

4.6 The Finished Process 35

5 Conclusion **39**

A Paper-Based Forms **41**

B Component Source Codes **45**

1 Introduction

This thesis gives the reader extensive information on the processes of the audit committee of Ulm University. Furthermore it closely examines the thesis deadline extension process and gives the reader an introduction on how to go about optimizing an existing process and implementing it as an automated process in a business process management suite (BPMS).

Business process optimization (BPO) is an important topic for any company because "the companies with the most efficient processes survive" [1]. Ulm University is not the typical company referred to in the quote, but definitely has a lot in common with one, foremost the need to save money. And saving money is usually done by maximizing the amount of work one paid worker can complete. The optimization this thesis examines is automation through software, moving away from paper-based forms and copying machines to purely digital processes, in hopes of saving time for secretaries, students and members of audit the committee of Ulm University.

The task of optimizing a business process has four basic steps, as described by J.S. Arlbjorn [1]:

1. Analysis
2. Design
3. Implementation
4. Evaluation

These steps also provide the basic outline for this thesis. Sections 1 and 2 analyze the audit committee and its processes. Section 3 examines options on how to design an optimized process for automation. Section 4 goes into detail on implementing an automated process, and finally Section 5 gives an evaluation of the optimized process and process automation of this sort in general.

1.1 The Audit Committee and its Tasks

The audit committee of Ulm University has numerous tasks. This thesis examines a few examples that involve multiple people and organizational units within the university. The examined processes are for instance the acknowledgment of credits from other universities or other majors within Ulm University, creating certificates for students making them eligible for BAföG[1], and naturally extension of thesis deadlines for bachelor master and diploma students. These tasks

[1] Bundesausbildungsförderungsgesetz

also offer the most optimization and automation potential which as the main topic of this thesis makes them relevant to this paper.

1.2 Process Optimization Potential with BPMS

Process optimization to the extent examined in this thesis is all about reducing media breaks and reducing the amount of time needed to process individual steps.

An introductory example for a non-automated process is given in Figure 1.1.

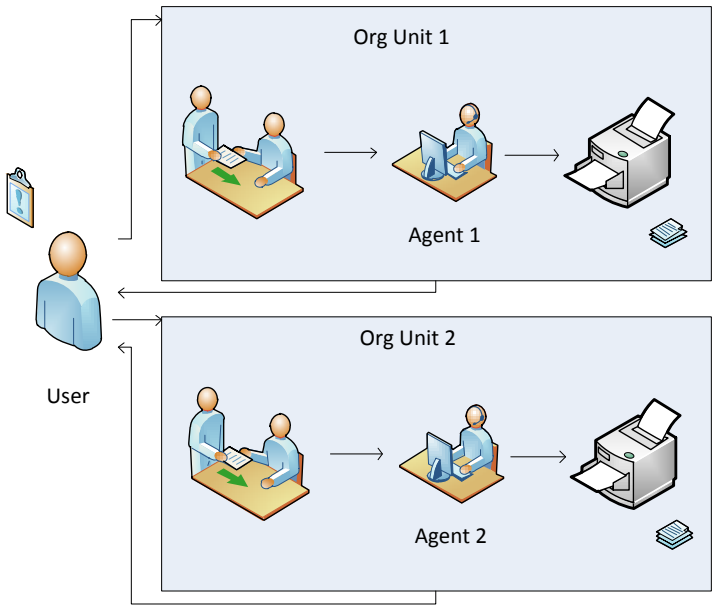


Figure 1.1: Non-automated Process

Assume a user needs a request serviced that requires information or input from two organizational units (OrgUnit) in his enterprise. He writes down the request, and takes it to the agent in OrgUnit 1. The agent then types the request into the computer, prints out the partial result, and returns it to the user. The user then takes the request to OrgUnit 2 and the agent there for completion. The user's request could not be fully serviced in OrgUnit1 because the agent there does not have access to the same information systems as the agent in OrgUnit 2 and vice versa. In OrgUnit 2 the agent must again input the request into the computer and print out the results. The request is now serviced and the user has his results.

The speed at which this process can be executed is mostly limited by the media breaks. And for correct execution one is assuming that the user and both agents know where exactly to forward the request to. Optimization of this process can be achieved by automation using business process management software, like the AristaFlow BPM Suite[2]. Ideally the whole process would

[2] http://www.aristaflow.com/

take place digitally, involving only forms and connections to the involved information systems, thereby eliminating all media breaks. This way the participants of the process do not have to physically change their location and need not have knowledge about process details. The process would then be, from the user's point of view, the content of Figure 1.2.

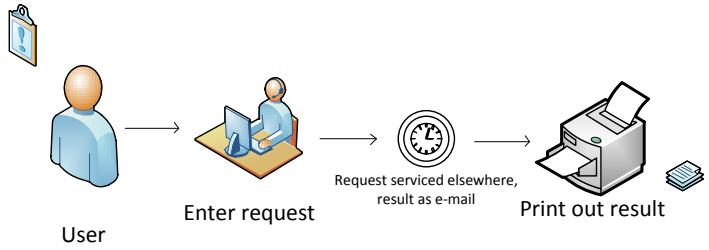


Figure 1.2: User's View of Automated Process

The actual servicing of the request would still be done by the agents 1 and 2 in the different organizational units, so the complete process would be as seen in Figure 1.3.

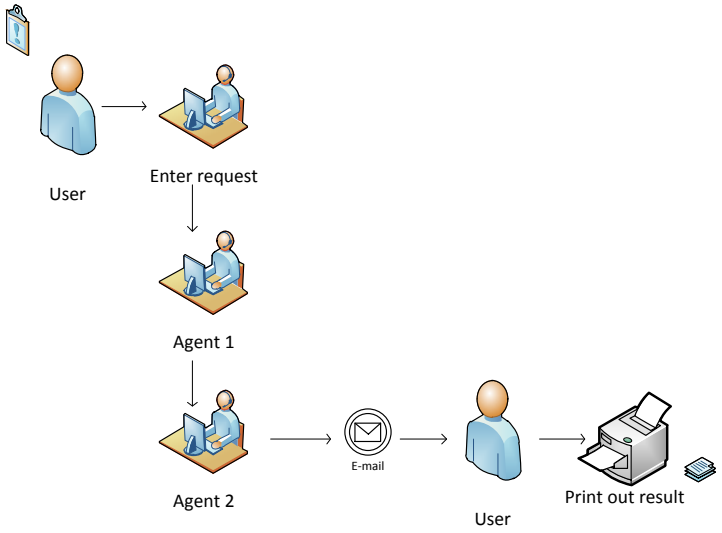


Figure 1.3: Complete View of Automated Process

The flow of information in the process would be handled entirely by a BPMS, as a BPMS has a model of the process. Using this model it can determine where the information entered by the student has to go next. The BPMS would also inform the next agent in the process flow that it is his turn to complete a step of the process.

A BPMS could also notify the student of acceptance or rejection of his request by e-mail. Without going further into the details of the specific processes it is clear how this reduction of media breaks can significantly reduce the time spent on a process both for students and the audit committee.

Naturally there are some real world limitations on how purely digital a process can be, especially in public administration. These stem from for instance legal issues or the need to archive important applications or other documents physically.

2 Current Processes of the Audit Committee

As mentioned in Section 1 the audit committee currently has numerous tasks, some of which will be described in the following. This thesis assumes that the reader has basic knowledge of process management and BPMN[1], which will be used to document the underlying processes.The following as-is processes are examined:

- BAföG funding approval (2.1.1)
- ECTS credit acknowledgment (2.1.2)
- Changing of major (2.1.3)
- Attribution of course credits (2.1.4)
- Work experience acknowledgment (2.1.5)
- Thesis deadline extension (2.1.6)

2.1 Documentation of the Different Processes

The following processes all have four main participants: the *student* requiring a service from the audit committee, the *chairman* of the audit committee, his *secretary* and a representative of the *student administration*. The student is always the initiator of the process. The secretary acts as a contact person between the student and the chairman, informs the student administration about the result of the process and is also responsible for the archiving of any legally binding documents created during the process. The chairman of the audit committee is the sole decision-making entity in the processes and therefore needs a broad knowledge about the different majors. The student administration registers the outcome of the process in the university's IT system.

In the following the processes are described in textual form as well as BPMN models. The BPMN notation can be looked up in [8].

[1] Business Process Management Notation

2.1.1 BAföG Funding Approval

This process is necessary when a student receiving *BAföG funding* is asked by the state to provide confirmation from the university that he or she is studying there and achieving the necessary grades and credits enabling the student to receive the BAföG advancements. The BPMN model for this process can be found in Figure 2.1.

The processes is started by the student visiting the secretary of the audit committee, bringing a current course assessment and the form which he has to fill out for the BAföG approval. The student gives the form and assessment to the secretary, who checks both for completeness and hands them on to the chairman if they are. The student may now leave. The chairman then checks the assessment and makes sure the student has achieved the minimum required credits necessary for BAföG funding approval. If that is the case he signs the form, stamps it and informs his secretary. The secretary notifies the student by e-mail that he can get back his filled out form. The student then returns to the secretary and picks up his form. This concludes the process.

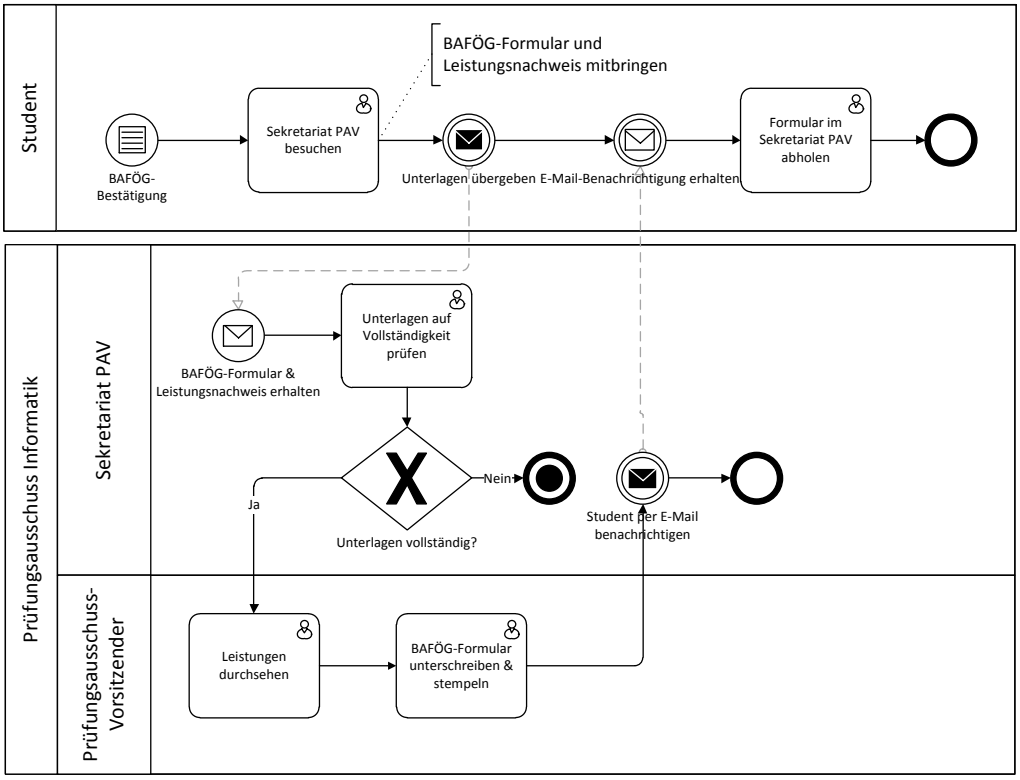


Figure 2.1: BAföG Funding Approval

2.1.2 ECTS Credit Acknowledgment

In the event that a student received ECTS[2] points from a different university, be it after spending a year in a foreign country or changing universities permanently to Ulm University, the student may have these acknowledged by the audit committee and assigned to his current course credits. The underlying process is explained here in textual form and as a BPMN model spread over Figures 2.2 and 2.3.

The process is initiated by the student filling out the paper-based form ”Antrag auf Anerkennung von Prüfungsleistungen” (Appendix A.2). Afterwards the student takes the filled out form containing a list of courses passed at a different university to a professor responsible for the student's field of study.

The professor compares the contents of the other university's courses to those of Ulm University, and signs the form if satisfied. The student then fills out the form ”Antrag Prüfungsausschuss Informatik” (Appendix A.1), in which he can state which services he needs from the audit committee, in this case *ECTS credit acknowledgment*.

The student may now visit the chairman's consultation hour, to which in addition to both forms he has to bring his current course assessment and certificates for the externally received credits. The student gives the forms, assessment and certificates to the secretary.

The secretary checks the forms and certificates for completeness and hands them on to the chairman if they are complete. The chairman then signs the student's application form and proceeds to calculating the equivalent German grades (if necessary). The paperwork is handed back to the secretary who creates the *notification letter* from a standard template.

The letters are printed out twice and signed by the chairman. A copy of the first printout is physically archived along with the forms and the student's course assessment. The original of the first printout is then handed to the student as proof of acceptance. The second printout is sent by in-house post to the student administration, who enters the grades and credits into the university information system LSF[3]. This concludes the process.

[2] European Credit Transfer and accumulation System
[3] Lehre-Studium-Forschung

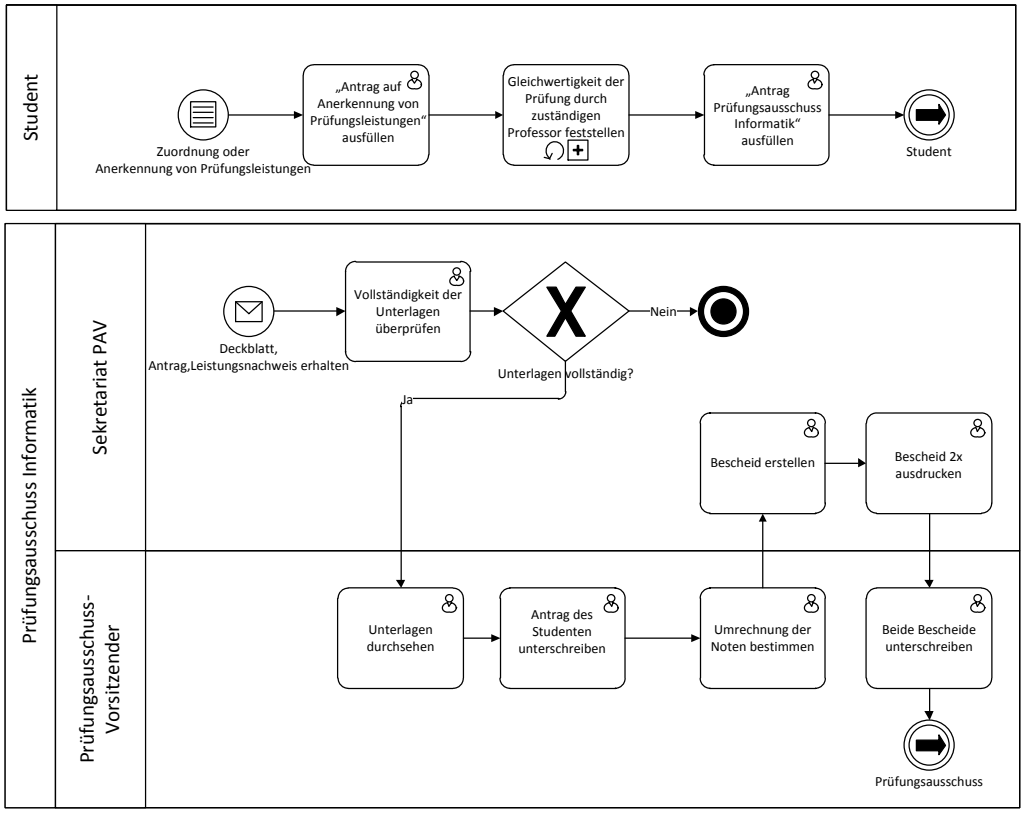


Figure 2.2: ECTS Credit Acknowlegment A

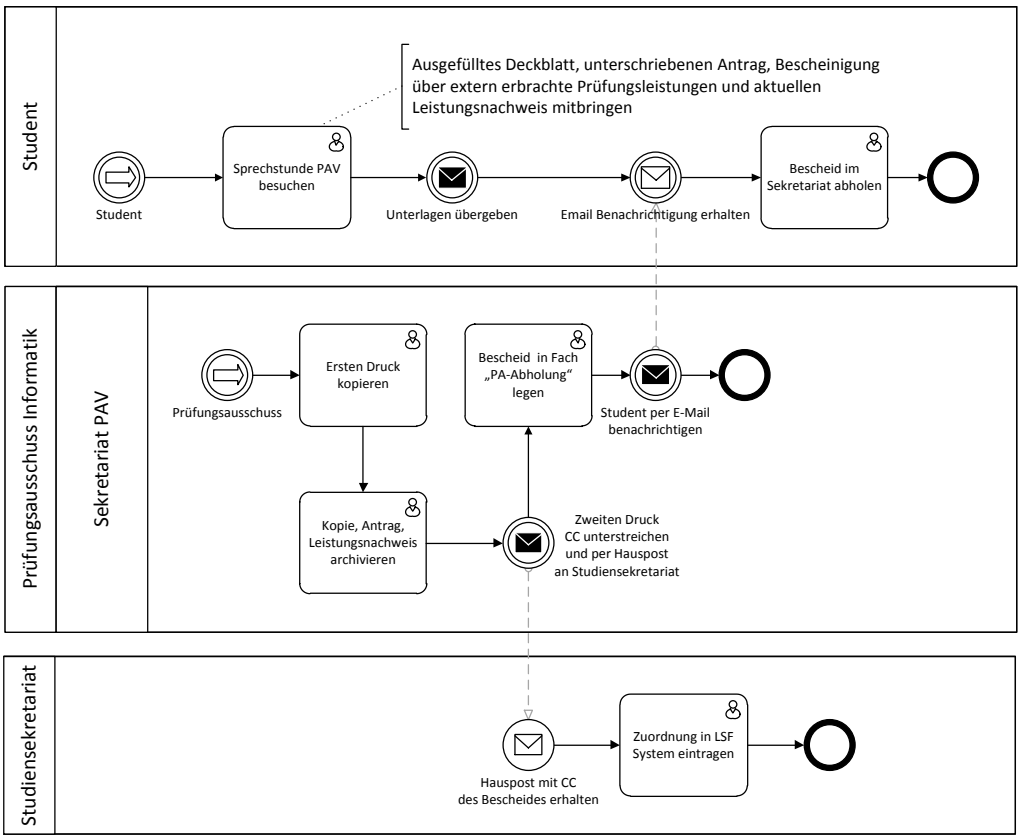


Figure 2.3: ECTS Credit Acknowlegment B

2.1.3 Changing of Major

This process is required when a student seeks to change his major field of study. It most often encapsulates the process *attribution of course credits* (cf. Section 2.1.4). The process is documented in the BPMN model in Figure 2.4.

The process is initiated by the student filling out the paper-based forms ”Antrag Prüfungsausschuss Informatik” and ”Antrag Studiengangwechsel” (Appendix A.3). Afterwards the student visits the chairman’s consultation hour, to which in addition to both forms he has to bring his current course assessment. The student gives the forms and assessment to the secretary. The secretary checks the forms and the assessment for completeness and hands them on to the chairman if they are complete. The chairman then decides into which semester the student will be setteled in in his new major. After having appended the information to the ”Antrag Studiengangwechsel” the process *attribution of course credits* (cf. Section 2.1.4) is started as a sub-process to ensure correct attribution of credits the student received in his previous major to his new major. Once the sub-process terminates the student receives the completed and signed forms and presents these to the student administration for approval. This concludes the process.

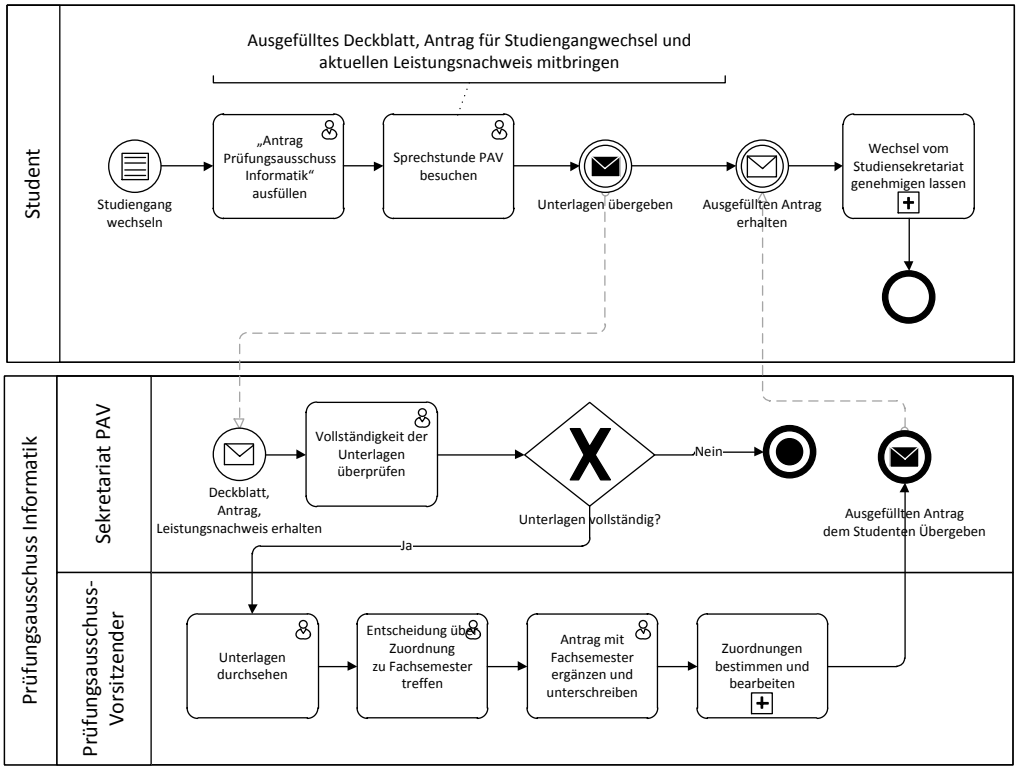


Figure 2.4: Changing of Major

2.1.4 Attribution of Course Credits

This process is similar to the process described in Section 2.1.2, *ECTS credit acknowledgment*, although in this case the credits that the student has achieved stem from different fields of study at Ulm University. The BPMN model of this process is spread over Figures 2.5 and 2.6 This process is also, most of the time, part of the process *changing of major*, detailed in Section 2.1.3.

The process is initiated by the student filling out the paper-based form ”Antrag auf Anerkennung von Prüfungsleistungen”. The student then fills out the paper-based form ”Antrag Prüfungsausschuss Informatik”, in which he can state which services he needs from the audit committee, in this case credit acknowledgment.

He or she may now visit the chairman’s consultation hour, to which he has to bring his current course assessment in addition to both forms. The student gives the forms and assessment to the secretary.

Afterwards the secretary checks the forms and the assessment for completeness and hands them on to the chairman if they are complete. The chairman checks the list of courses the student passed in his previous major and creates a recommendation for the attribution of the achieved course credits to the new major. If the student agrees to the recommendation the chairman signs the student’s application form. The paperwork is handed back to the secretary who creates the notification letter from a standard template.

The letters are printed twice and signed by the chairman. A copy of the first printout is physically archived along with the forms and the student’s course assessment. The first printout is then handed to the student. Then the second printout is sent by in-house post to the student administration who enter the grades and credits counting towards the student’s new major into the LSF system. This concludes the process.

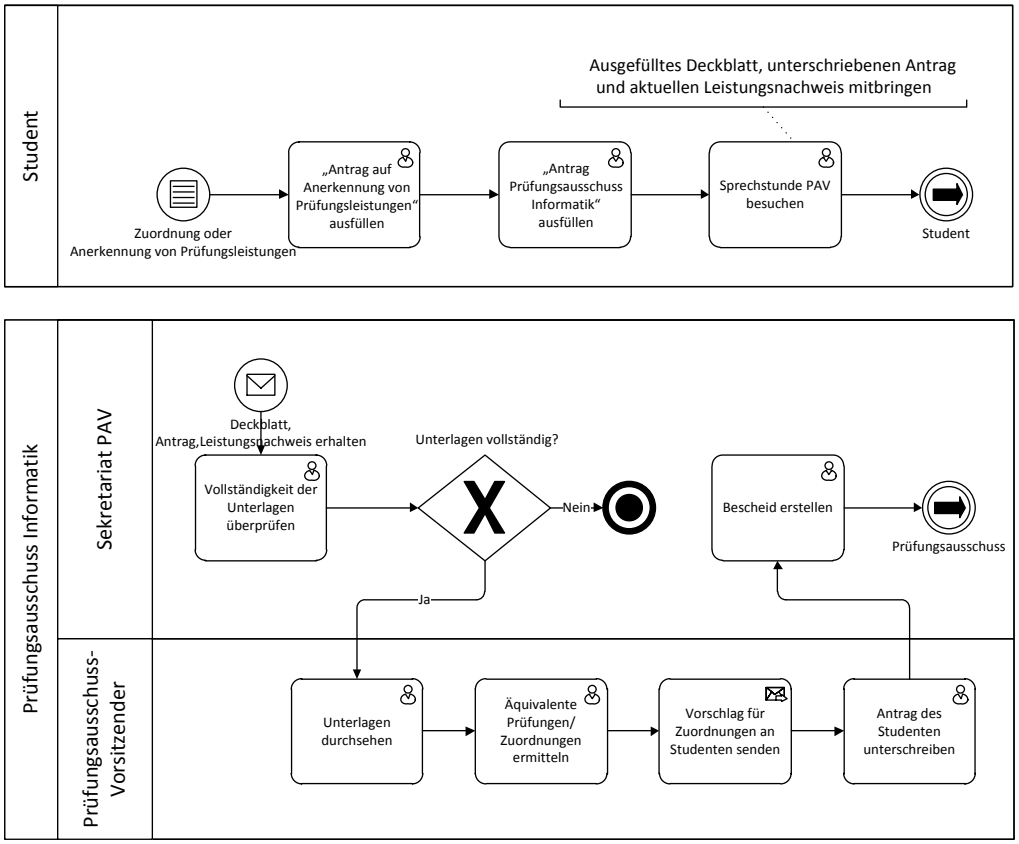


Figure 2.5: Attribution of Course Credits A

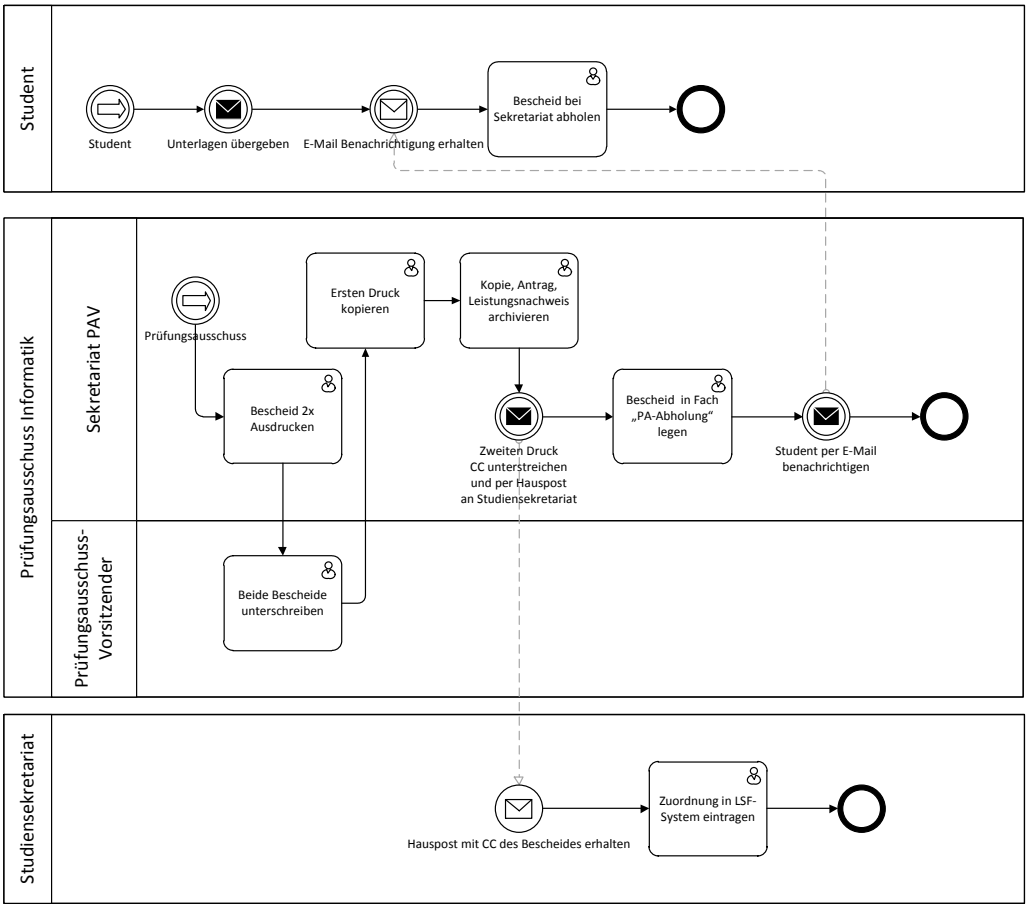


Figure 2.6: Attribution of Course Credits B

2.1.5 Work Experience Acknowledgment

This process is necessary when the student wishes to have work experience accumulated before or during his studies acknowledged in form of an internship or course at the university.

This is only possible for students currently studying for a diploma major and therefore the process will most likely be phased out in a few years. The BPMN model documenting the process can be found in Figure 2.7.

The process is initiated by the student filling out the paper-based form ”Antrag Prüfungsausschuss Informatik”. The student can now visit the chairman’s consultation hour, to which in addition to the form he must bring a certification from the former workplace. The student gives the form and certification to the secretary, who checks both for completeness and hands them on to the chairman if they are.

The chairman now decides whether to acknowledge the student’s practical work experience as a university course or practicum. Once he has made his decision he informs his secretary who then either enters the acknowledgment into the LSF system or informs the student of the rejection by e-mail. This concludes the process.

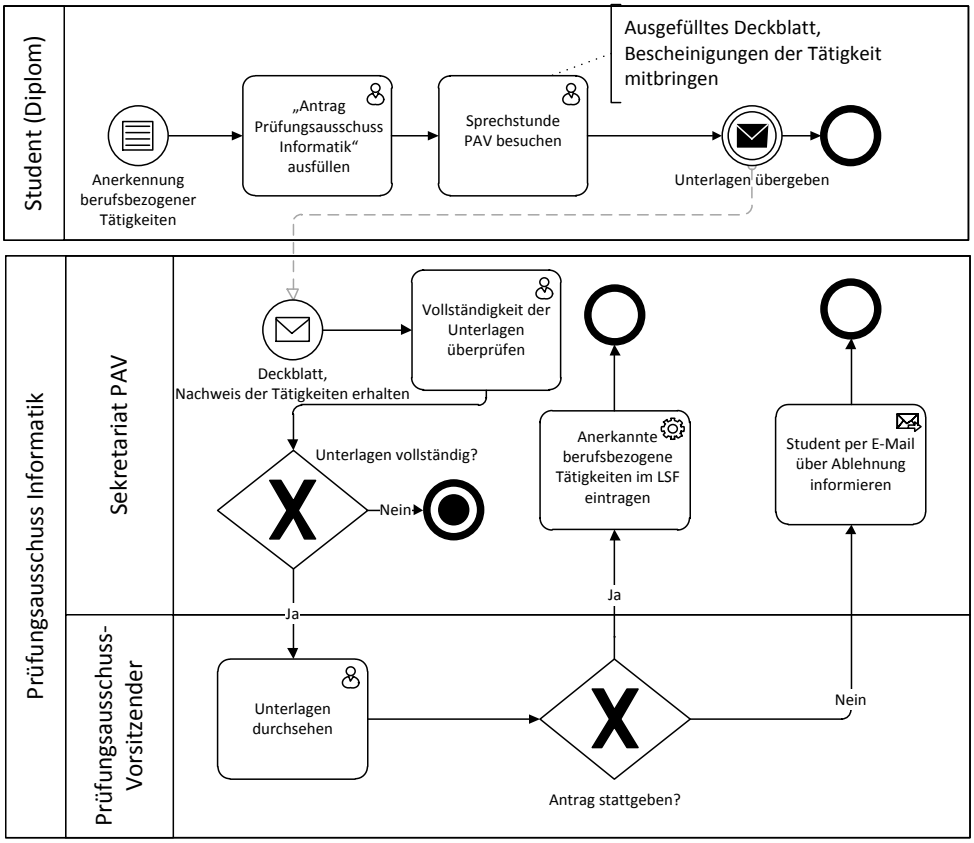


Figure 2.7: Work Experience Achnowlegment

2.1.6 Thesis Deadline Extension

This process is necessary when a student currently writing a bachelor's, master's or diploma thesis has ample reasons for requiring a deadline extension. The deadline for a bachelor's thesis may be extended up to two weeks, the deadline for a master's or diploma thesis up to four weeks. The request for a thesis deadline extension must be made not later than one month before the original deadline. This is the most relevant process for this thesis as it is the process that is optimized and automated using a BPMS in the course of this thesis. Automation could naturally have been applied to any of the processes but the thesis deadline extension process being one of the more complex was chosen as an example. The as-is model of the process is spread over Figures 2.8 and 2.9.

The process is, akin to the other discussed processes, initiated by the student. He has to write a letter of reasoning for the thesis deadline extension. This letter is given to the supervisor of the thesis and has to be signed by him. Afterwards the student fills out the form "Antrag Prüfungsausschuss Informatik". He may now visit the chairman of the audit committee's consultation hour, to which he has to bring the filled out form, the signed letter of reasoning and a current course assessment.

He hands this paperwork to the secretary of the audit committee who checks it for completeness and hands it on to the chairman if it is complete. The chairman must now reach one of three decisions: he may either grant the extension, deny it or grant only a partial extension. In case the chairman denies the extension he informs the student thereof by e-mail. If he grants it partially he must still decide on a new deadline date.

After having fixed a new deadline, the process continues in the same way as it would if the chairman were to grant the full extension the student asked for. In both cases the secretary types the letter of notification which gets printed twice. Both printouts are then signed by the chairman, and the first signed printout is copied and physically archived along with the letter of reasoning and the course assessment. The second printout is sent by in-house post to the student administration, who in turn enters the new deadline into the LSF system. The secretary of the audit committee also informs the student that his notification letter is ready to be picked up. This concludes the process.

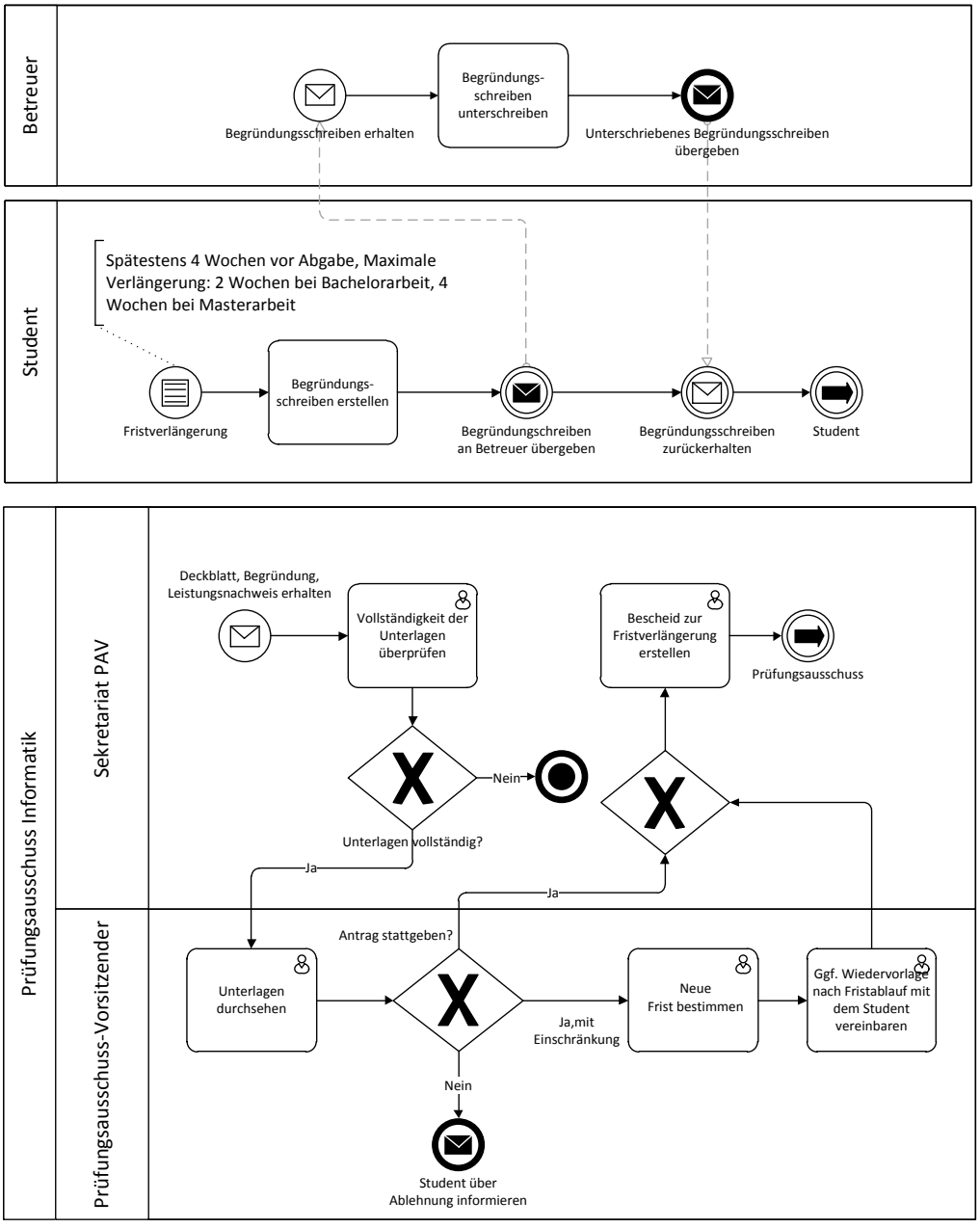


Figure 2.8: Thesis Deadline Extension A

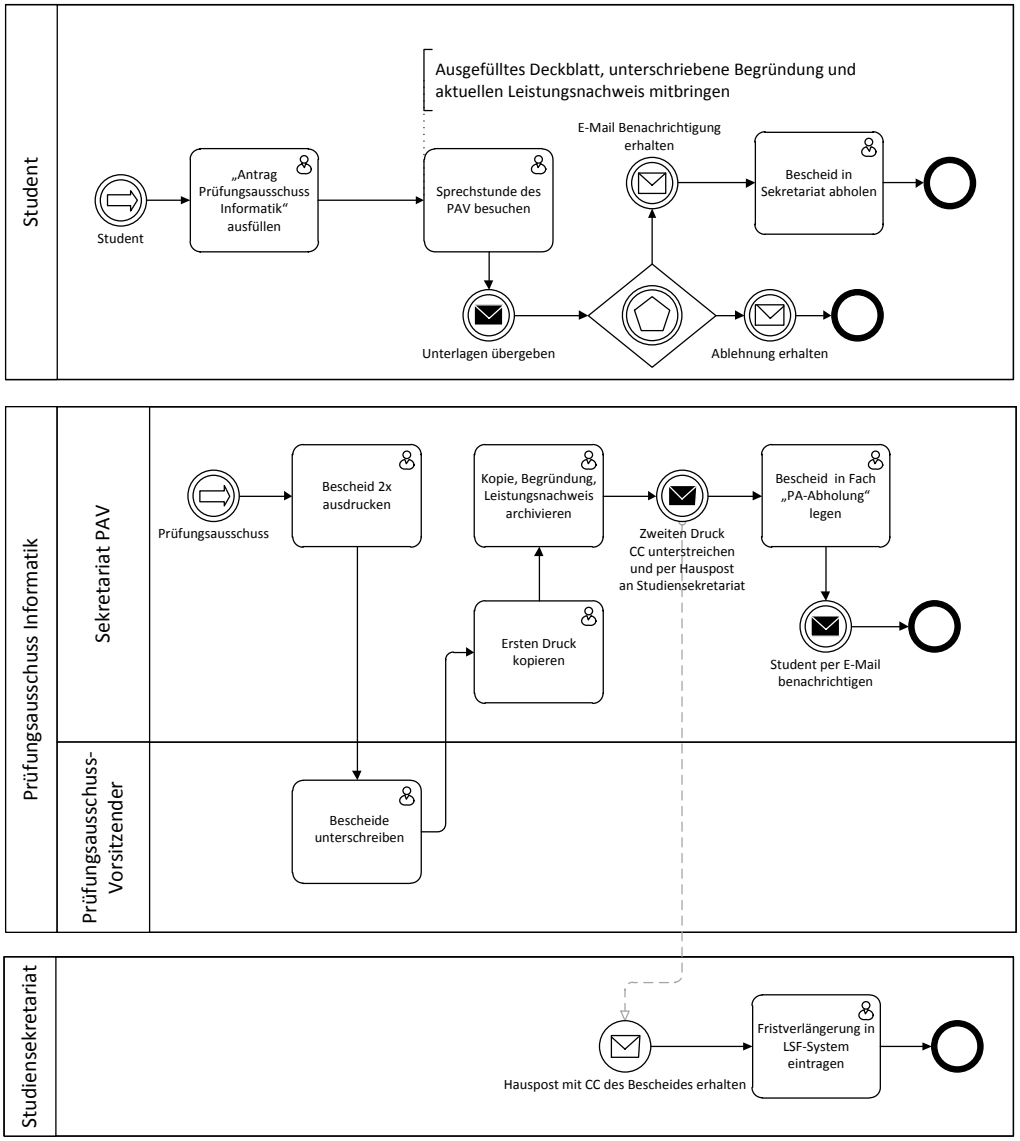


Figure 2.9: Thesis Deadline Extension B

The documentation of the processes makes it clear that there is a lot of room for optimization through automation as the current processes involve many paper-based forms, physical relocation of the student and usage of the, in contrast to electronic mail, slow in-house post of Ulm University.

3 Optimization of the Thesis Deadline Extension Process

This section discusses the necessary steps in optimizing the thesis deadline extension process. The process, as it is being currently executed at the university, is described in Section 2.1.6 and the corresponding BPMN model is spread over Figures 2.8 and 2.9.

The first step in optimizing the thesis deadline extension (TDE) process was to contemplate possible automation entry points in the process. The aim is to make the process as completely digital as possible, except for physical archiving. The changes made are detailed in Section 3.1. Sections 3.2 and 3.3 describe the first steps leading up to a process implementation, specifically modelling and planning the implementation of the optimized process in a specific BPMS.

3.1 Optimization Changes to the TDE Process

The need for optimizing the TDE process stems from the numerous media breaks and excessive use of paper-based forms and print-outs. The following changes aim at speeding up the process as a whole and reducing the workload for the participants.

The first major change to the TDE process is to eliminate the need for the student in question to fill out a paper-based form and visit the consultation hour offered by the chairman of the audit committee (cf. Figure 3.1) by using a web form. The information gathered in this form is the basis for the process instance.

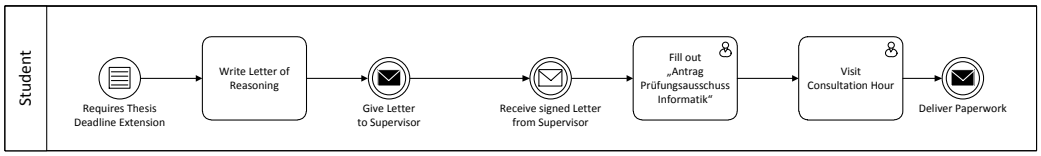


Figure 3.1: Student's As-Is Process

The letter of reasoning, signed by the supervisor, is no longer physically handed to the secretary of the audit committee, but instead scanned and uploaded as a PDF file to the web form. This change is reflected in Figure 3.2.

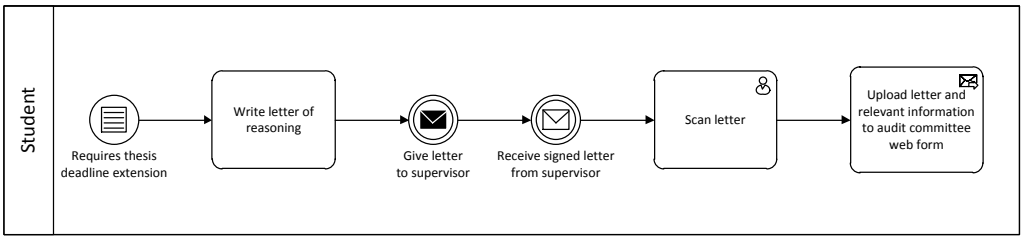


Figure 3.2: Student's Optimized Process

The advantages of this change are obvious: the student's information is, without the secretary having to type it into her computer, available digitally in the process. Also the student does not have to show up in person in order to make his request. These two facts save time for all participants.

The secretary is now informed by the process that a new instance has been started. She may now check the submitted web form and PDF files for completeness. If she deems the forms to be complete she may signal the process to send them on to the chairman of the audit committee for further review. In case the forms are not complete she has the option to send the student an e-mail detailing the missing information. The template for this e-mail and the student's e-mail address are provided to her by the business process management suite.

As soon as the secretary marks the form data as complete the BPM suite signals the chairman by adding an item to his work list. The chairman may now review the data entered by the student and examine the uploaded PDF files. He is also shown a form (cf. Figure 3.3) with options, these include rejecting the student's request for an extension of his thesis deadline, accepting it or sending an e-mail to the student requesting him to come in person to the next consultation hour. In case the student is summoned the form will be re-added to the chairman's work list, so he can make an informed decision after the consultation hour.

After the chairman has accepted the deadline extension the BPMS automatically creates a letter of notification as a PDF using the student's data. This letter is attached to a short automated e-mail and is sent to the student. The letter is now digitally signed in preparation for being sent to the student administration. Once the PDF has been digitally signed the BPMS attaches it to an e-mail which is then encrypted and sent to the student administration.

The student administration now enters the information from the PDF letter into the LSF system.

A very simple view on the typical execution of the process is given in Figure 3.4.

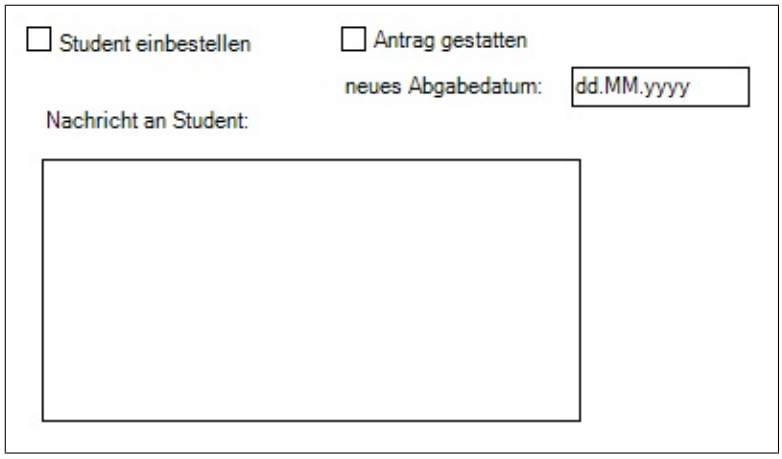


Figure 3.3: Chairman's Web Form Options

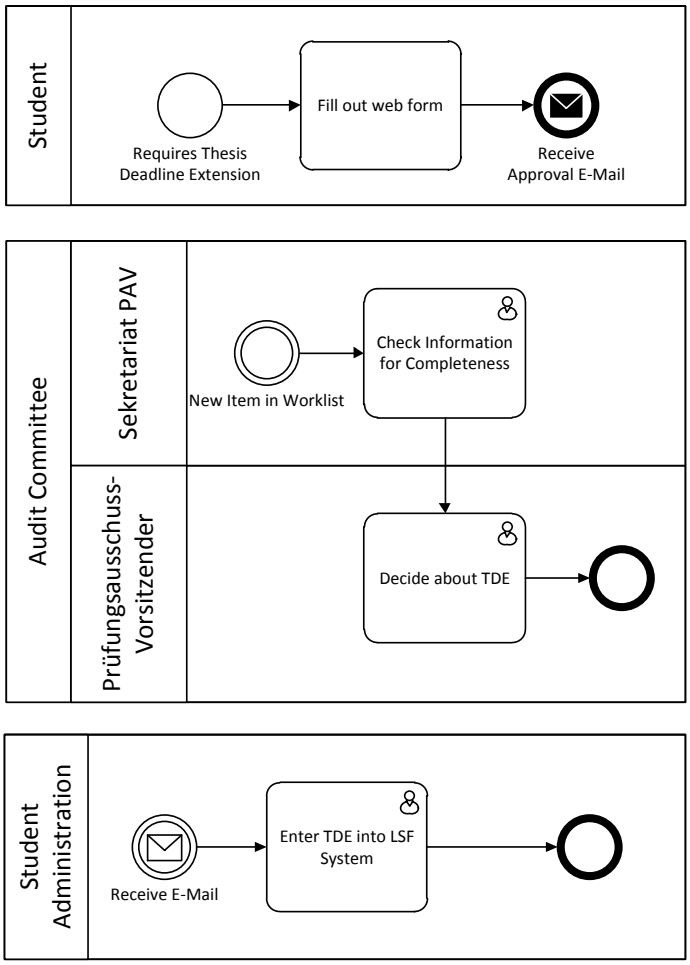


Figure 3.4: Oversimplified View on Optimized Process

3.2 Creating the Process Model in the AristaFlow BPM Suite

After having decided which parts of the process to automate and having a clear idea how the finished process should ideally work, the next step is to model the process in a BPMS. For this purpose the AristaFlow BPM Suite[1] is used [2]. The AristaFlow BPM Suite helps modelers by enforcing guidelines, part of the *correctness by construction* principle described in [2]. Some of these conform to the 7PMG[2], specifically G3 ”Use one start and one end event”, G4 ”Model as structured as possible” and G5 ”Avoid OR routing elements” [6].

The implementation-ready model should only reflect aspects of the later process that are optimized. For instance a BPMN model of a simple process, in which a secretary has to print out a form and then sign it by hand, would only contain the printing out of the form, as the signing of the document can not be automated. It is possible though to have an activity in the AristaFlow model reminding the secretary of signing the document.

Figure 3.6 is an example of how a part of the BPMN model of the TDE process (cf. Figure 3.5) was translated into an implementation-ready AristaFlow process model.

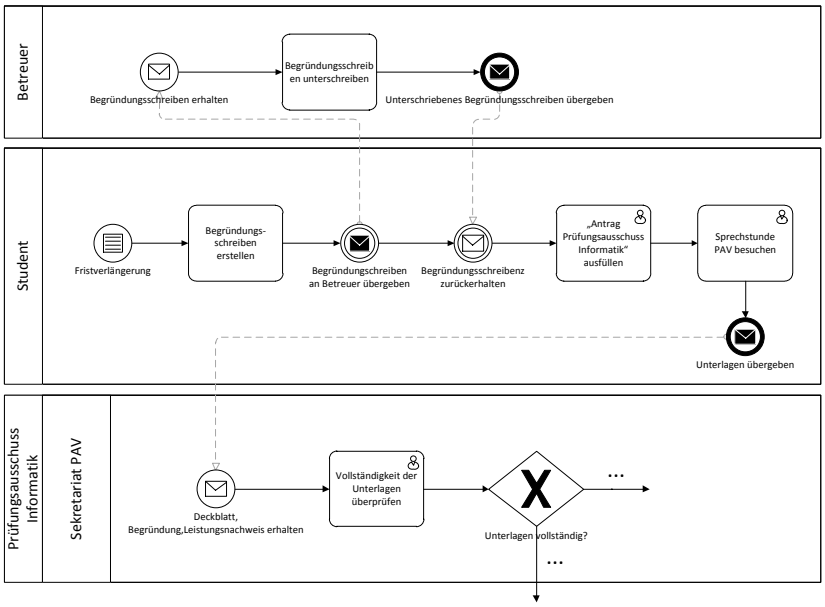


Figure 3.5: Partial Process (BPMN)

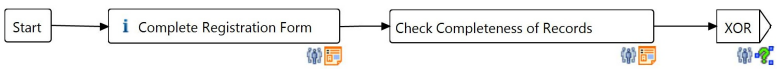


Figure 3.6: Partial Process (AristaFlow)

[1] http://www.aristaflow.com/
[2] Seven Process Modeling Guidelines

As can be seen in this short example, the AristaFlow model is a lot less detailed than the BPMN model. This is of course, due to the non-automatable parts missing and the optimization. In AristaFlow process logic is contained in the activities themselves. An example of this is the message flow in the BPMN model between the student and secretary, the handing over of the paperwork. In the AristaFlow model the activity the secretary completes contains information about where to get the student's information entered into the web form and the signed reasoning letter PDF from.

Figure 3.7 showcases the final part of the automated process model in AristaFlow. Note the parallelization achieved through automation of printing, automated e-mails to the student and student administration and digital archiving in contrast to the original process, depicted in Figure 3.8.

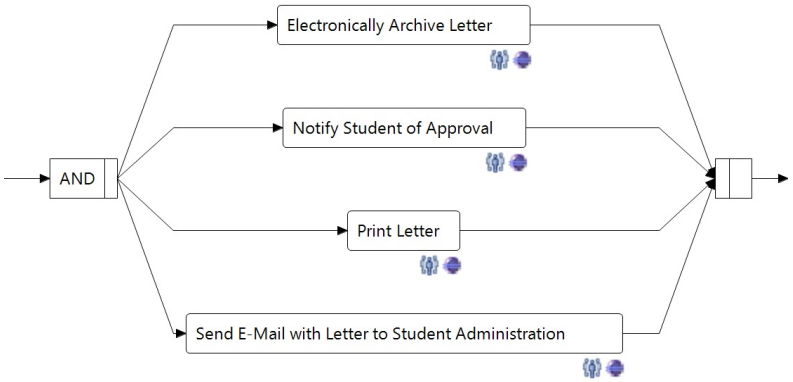


Figure 3.7: Activity Parallelization in AristaFlow

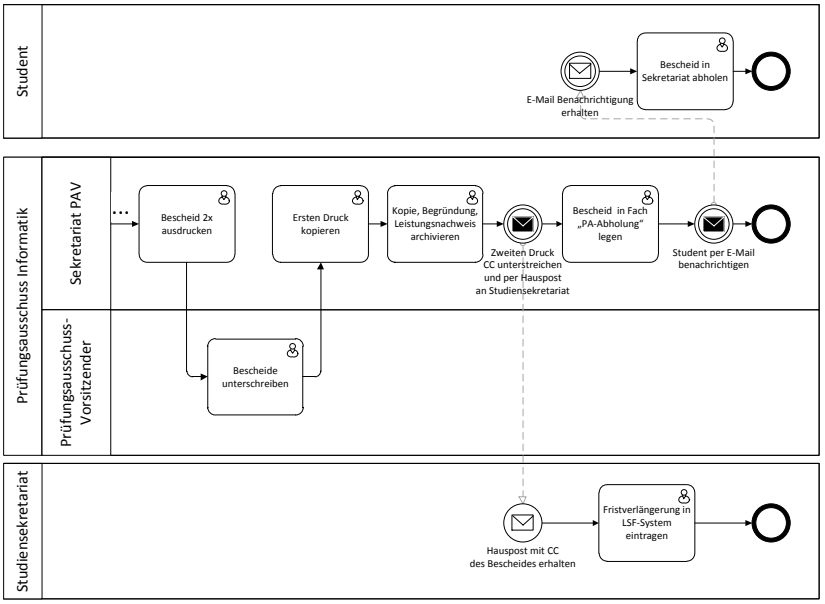


Figure 3.8: Sequential Execution of Previous Process

3.3 Planning the Implementation

After having created the basic process model without any assigned activity functions or data elements, the next step is to plan the implementation of the activities in the model. This was done by creating a list of activity templates needed for the TDE process in AristaFlow. The basic requirements were:

1. Displaying web forms for gathering of student information
2. Uploading PDF files into the process
3. Displaying interfaces for the secretary and the chairman to review information and make decisions
4. Sending automated and template-based e-mails
5. Automatically filling PDF forms
6. Digitally signing PDF forms with certificates
7. Sending of e-mails with encrypted PDF attachments
8. Displaying and printing PDF files

Some of these requirements can be fulfilled by default AristaFlow activities, for instance the ”user form” activity for the web forms and interfaces, which can also handle file uploads by users and display these files to other users, using local software such as Adobe reader. Other requirements however, such as encryption and digital signing, require implementations as components for AristaFlow, effectively creating new activities for use with the process.

To be more precise, requirements 1 through 4 can be fulfilled by AristaFlow without requiring an activity implementation, requirements 5 through 8 on the other hand require a specific implementation as they are not included in the standard AristaFlow tool set. The implementation of these components is detailed in Section 4.

4 Implementation

4.1 Implementing an AristaFlow Component

In order to be able to program an AristaFlow component one needs to understand what a component is. Simply put it is a small Java application that AristaFlow can run as an activity in the process model. The basic idea is to hide the complexity of the component's code from the process modeler, as "a process implementer should not need to know the details about the implementation of these application functions" [3].

This section only describes the basics of programming such a component. After the implementation is complete the component has to be registered in the *AristaFlow Activity Repository*. This is where the necessary information for component use by the *Process Template Editor* is saved. This is, in essence, a list of the parameters the component expects from the process and of what data types these must be.

AristaFlow has an open API that has a "modular and service-oriented design" [4], meaning that new components can be easily developed, extending AristaFlow by any number of capabilities. One of the more prominent examples of this is the WJ&P WebFormDesigner[1], described in detail in [7]. The form designer allows the creation of web forms like the ones used in the TDE process in a WYSIWYG style.

In the following the creation of more basic components is discussed, i.e. without a GUI; basically Java console applications. Such a component must have one main class that extends *ExecutionEnvironment*[2], one constructor with a specific signature (cf. Listing 4.1), and a *run()* method that has to end with a specific ending statement to inform the execution engine that the activity has finished. The *run()* method may contain any Java code, using different libraries, calling methods from different classes, or almost anything else you could do in standard Java. Also the component must reference the *aristaflow-integration-libraries.jar*, containing the classes and methods needed for component creation. In conclusion, the necessary lines of code for an AristaFlow component are given in Listing 4.1.

[1] www.wjp.de
[2] de.aristaflow.adept2.core.runtimemanager.executionenvironments.ExecutionEnvironment

```
1  public class Somecomponent extends ExecutionEnvironment
2  {
3          public Somecomponent(ActivityInstance activityInstance)
4          {
5                  //create ExecutionEnvironment
6                  super(activityInstance);
7          }
8          public void run()
9          {
10                 //*
11                 ANY CODE HERE
12                 *//
13
14                 //shutdown component
15                 sessionContext.getRuntimeEnvironment().applicationClosed();
16         }
17 }
```

Listing 4.1: Empty AristaFlow Component

The code in Listing 4.1, however, has no functionality at all. In order for a component to have any functionality it needs to be able to read data from and write data to the process. There are two types of data an AristaFlow component can read from the process as it is being executed:

- Process parameters
- Activity configurations

In the following Sections 4.1.1 and 4.1.2 these two types of data are described in detail.

4.1.1 Process Parameters

The *process parameters* are what the AristaFlow Process Template Editor (PTE) refers to as data elements. They encapsulate data that is reused and exchanged in the process. In the TDE process examples of data elements are the student's e-mail address, that was gathered from a web form or the PDF file the student uploaded. As per convention, data elements in the process model should only contain dynamic data specific to one instance of a process.

One line of code (cf. Listing 4.2) is necessary to prepare a new component for reading and writing of process parameters, it is inserted into the *run()*method:

```
1  DataContext dataContext = sessionContext.getDataContext();
```

Listing 4.2: Getting the Process' DataContext, enabling Access to all Data Elements

The variable *dataContext* can now be used with the line of code listed in Listing 4.3 to access the data element named *someparameter*.

```
1  String someString = dataContext.retrieveStringParameterValue("someParameter");
```

Listing 4.3: Retrieving a String Parameter

Note that *someParameter* has to be the exact name of the data element in the process model and that the correct method corresponding to the type of data element has to be used, for example *retrieveDateParameterValue()* for a *Date* object.

The variable *someString* is now filled with the content of the data element from the process model at run-time and can be freely manipulated like any other Java object. The *DataContext* class, and its instance *dataContext* also have further methods for other standard Java data types, such as *Boolean* or *Integer*. AristaFlow also supports the usage of *User-Defined Data Types (UDT)* in its process models, that can be read as *Byte[]* objects in the component.

Once manipulated, data can be reinserted into the process instance. This is done by again leveraging methods offered to us by the DataContext class, as shown in Listing 4.4:

```
1 dataContext.storeStringParameterValue("someParameter", someString);
```

Listing 4.4: Storing a String Parameter

This will save the value of *someString* to the data element *someParameter* in the process instance.

4.1.2 Activity Configurations

The second type of data a component may read from the process model are *activity configurations*. Activity configuration data is appended by the process modeler to an activity. Activity configurations are mostly strings telling a component to act in a certain way at run-time, they are static in the process model and as such are the same for all instances of a certain process model.

A short example to clarify the roles of activity configurations and process parameters is given in Figure 4.1. The process writes an integer each to the data elements *Number A* and *Number B*. The activity on the second node then completes a calculation using the integers.

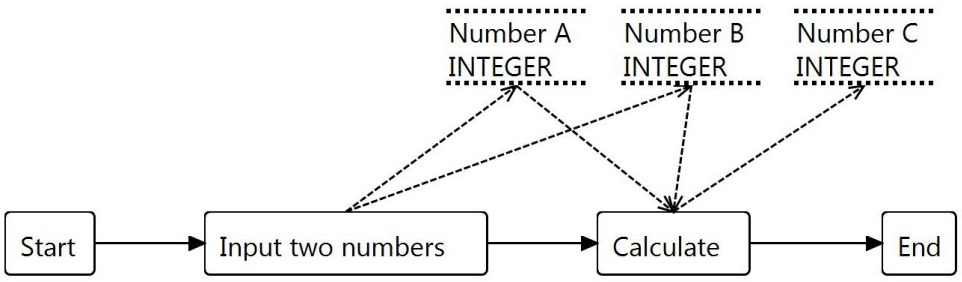


Figure 4.1: Simple Calculation Process

The type of calculation performed, i.e. adding or subtracting the two numbers, is defined by the second node's activity configuration, as shown in Figure 4.2. In the example the activity configuration in Figure 4.2 configures the activity to *add* the two numbers.

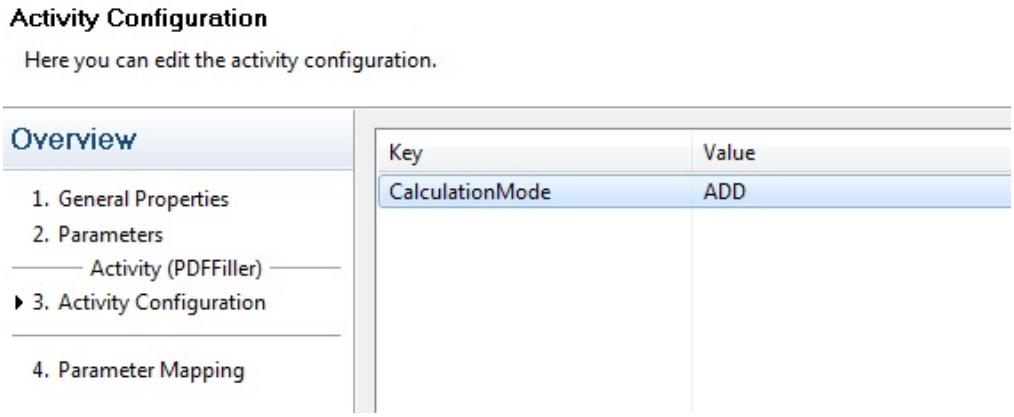


Figure 4.2: Setting an Activity Configuration

Obviously the component must support activity configurations and the *run()* method must therefore be modified (cf. Listing 4.5).

The *activityConfiguration* variable may now be used to gain access to the strings in the activity configuration (cf. Listing 4.6).

```
1 ActivityConfiguration activityConfiguration = activityInstance.getConfiguration();
```

Listing 4.5: Getting the Process' Activity Configuration

```
1 String someConfigurationString=activityConfiguration.getString("someConfiguration");
```

Listing 4.6: Retrieving a String from the Activity Configuration

Finally returning to the example from Figure 4.1 the complete code listing for a working calculation component would be the content of Listing 4.7.

```
1  public class Calculator extends ExecutionEnvironment
2  {
3          public Calculator(ActivityInstance activityInstance)
4          {
5                  super(activityInstance);
6          }
7
8           public void run()
9          {
10                 DataContext dataContext = sessionContext.getDataContext();
11                 ActivityConfiguration activityConfiguration = activityInstance.getConfiguration();
12
13                 //Integers returned by the process are in fact of type long
14                 long numberA = dataContext.retrieveIntegerParameterValue("Number A");
15                 long numberB = dataContext.retrieveIntegerParameterValue("Number B");
16                 String calculationMode=activityConfiguration.getString("CalculationMode");
17
18                 long result;
19                 if(calculationMode.equals("ADD"))result=numberA+numberB;
20                 if(calculationMode.equals("SUB"))result=numberA-numberB;
21
22                 dataContext.storeIntegerParameterValue("Number C", result);
23
24                 sessionContext.getRuntimeEnvironment().applicationClosed();
25         }
26 }
```

Listing 4.7: Calculation Component Example

Note that the return value of the *retrieveIntegerParameterValue()* method is actually a *long*. Apart from that Listing 4.7 is the composition of all previous listings in this section. Imports are omitted in the example as they are generally automatically generated by a suitable IDE such as Eclipse.

4.1.3 Multiple Optional Parameters

An advanced feature in component creation is to allow an undefined amount of optional parameters. This could be used to expand the Calculator example to allow the adding of an undefined amount of integers.

To do this, a *Set* of all attached process parameters is needed (cf. Listing 4.8:

```
1  Set<ProcessModelParameter> processParameters = activityInstance.getParameters(AccessType.READ);
```

Listing 4.8: Creating a Set of all Process Parameters

This *Set* does not contain the data of the process parameters but has information like the name and data type of all attached parameters.

To get an *ArrayList* of all attached data elements of the integer data type the following code in Listing 4.9 would suffice.

```
1 ArrayList<Integer> list = new ArrayList<Integer>();
2 for (ProcessModelParameter p : processParameters)
3 {
4         //the AdeptDataType class has fields for all supported data types
5         if(p.getDataType()==AdeptDataType.INTEGER)
6                 list.add(dataContext.retrieveIntegerParameterValue(p.getName()));
7 }
```

Listing 4.9: Getting all Attached Integer Parameters

4.1.4 Debugging a Component

As components will not run in Eclipse or any other Java IDE, debugging is done by logging errors to a *java.util.logging.Logger*. The logged errors can be viewed at run-time in the AristaFlow Test Client. A simple way of defining a *Logger* variable in the *run()* method is described in Listing 4.10.

```
1 Logger log = Logger.getAnonymousLogger();
```

Listing 4.10: Creating a Logger Object

An exemplary usage for the *Logger* is given in Listing 4.11.

```
1 Logger log = Logger.getAnonymousLogger();
2 if(calculationMode.equals("ADD"))result=numberA+numberB;
3 else if(calculationMode.equals("SUB"))result=numberA-numberB;
4 else log.severe("Incorrect or missing calculation mode!");
```

Listing 4.11: Usage of the Logger for Debugging

”Incorrect or missing calculation mode!” will now be displayed in the AristaFlow Test Client’s log on execution of the component if the modeler forgot to set the activity configuration properly.

4.1.5 Exception Handling

It is important to properly handle exceptions in an AristaFlow component as ”process-aware information systems will not be accepted by users if rigidity of idleness due to failures comes with them” [5]. To support the error handling techniques described in [5], the Java component has to tell AristaFlow what went wrong, when a run-time exception happens.

Exceptions in an AristaFlow component are split into two main types:

- ApplicationFailedException
- ApplicationEnvironmentException

If a ”standard” Java error is caught in a *try-catch()* block, for instance an *IOException*, the best practice is to throw a new *ApplicationFailedException* with an error message, state information (can be null), an error code, and the exception itself. This will hand the exception on to the Test Client and show it to the user (cf. Listing 4.12). The *ApplicationErrorCodes* class has error code fields for most standard Java exceptions.

```
1 catch (IOException e)
2 {
3         throw new ApplicationFailedException
4                 ("File could not be read", null, ApplicationErrorCodes.IOEXCEPTION, e);
5 }
```

Listing 4.12: Exemplary IOException Throwing

ApplicationEnvironmentExceptions are exceptions caused by AristaFlow specific influences to the component, for instance a read attempt on a missing non-optional process parameter would throw a *NoSuchParameterException*. These exceptions are handled slightly different than standard Java exceptions, as they are AristaFlow specific.

As can be seen in Listing 4.13, the thrown exception is of the *ApplicationEnvironmentException* type and does not contain the state information that the ApplicationFailedException does. Again, the error codes themselves are found in the *ApplicationErrorCodes* class.

```
1 catch (NoSuchParameterException e)
2 {
3         throw new ApplicationEnvironmentException
4                 ("Parameter missing", ApplicationErrorCodes.PARAMETER_NOT_EXISTING, e);
5 }
```

Listing 4.13: Exemplary NoSuchParameter Throwing

Having covered most of the necessary techniques for creating a component, the next sections describe the components implemented for use in the TDE process.

4.2 PDF Printer Component

The *PDF Printer* component was developed to cope with the physical archiving requirement of the TDE process (cf. Section 2.1.6). It uses the *com.sun.pdfview* package[3], specifically the *PDFRenderer* class, and the classes of *java.awt.print* to print a PDF file. The PDF file to be printed is provided by a process parameter in *FileUDT* format. *FileUDT* is an XML based AristaFlow specific format, that encapsulates data and meta data, such as MIME-type etc. into a Java object or process model data element.

The activity configuration for the component tells the component what printer name and what paper size to use. If none of these are specified, the component prints to the default printer using A4 paper.

[3] http://java.net/projects/pdf-renderer

The core concept of the component is to implement the *java.awt.print.Printable* interface with a new class, using the *PDFRenderer* and *PDFPage* classes as described in the online tutorial for the *com.sun.pdfview package*. Wrapping a *PDFFile* in this new class via the constructor makes it printable using *java.awt.print.*

ACTIVITY CONFIGURATION	DESCRIPTION
Printer Name	Must be the exact printer name. ex. "printserver/printername"
Paper Size	The paper size to print to: "A3","A4","A5","Letter" or "Legal".
PROCESS PARAMETERS	DESCRIPTION
PDF :UDT	This FileUDT must encapsulate a PDF file.

Table 4.1: Activity Configuration and Process Parameters for PDF Printer Component

4.3 PDF Filler Component

In order to allow AristaFlow to fill *AcroForms* and *static XFA* PDF forms, the *PDF Filler* component was developed. The application of the component in the TDE process is the filling of a PDF template for the notification letter which is sent to the student and the student administration. The PDF template is an AcroForms PDF form, with the Ulm University letterhead and form fields for the address, the title of the letter and most of the text. The template is created using Adobe Acrobat 8 or newer. The fields are filled with the information gathered from the student by the TDE process and its web forms. This data, which is managed as data elements in the process instance, is handed to the component in form of process parameters. Fields like the letter heading field, which consist of multiple pieces of data, can be concatenated using rules defined in the activity configuration (cf. Table 4.2).

KEY	VALUE
addressField	%s:sex%\n%s:fullName%\n%s:address%

Table 4.2: PDF Filler Activity Configuration Example

Note the concatenation format %s:*someParameter*%, this will later insert the content of *someParameter* into the activity configuration string addressField at exactly the position between the opening % and the closing %. The usage of *s:* denotes that the content of *someparameter* should be inserted as a String, which is always the case for the PDFFiller component, it does not have any meaning in regard to the data type *someparameter* has.

The activity configuration given in Table 4.2 would tell the component to concatenate the content of the string data element containing the student's salutation with a new line symbol to the student's full name and that to, again on a new line, the content of the data element containing his address. This concatenation is handled in the component similarly to how it is explained in Section 4.1.3, with the addition of the usage of the *SystemDataTools.formatter()* method, that fills the activity configuration string with the content carried by the appropriate attached process parameters.

The PDF Filler component uses, apart from the standard AristaFlow integration library, the *iTextPDF* library[4]. This library exposes the API necessary for parsing a PDF file in the file system as a Java object, and manipulating it.

The basic outline of the component, after having parsed the string from the activity configuration and filling its parameters, is to first get the location of the PDF template from the activity configuration, load the file into a *PDFReader* object and finally parse the names of the empty form fields in the document. The strings from the activity configuration are now written into the corresponding fields contained in the ”key” part of the activity configuration. The form is then flattened and handed over to the process instance. The component then unloads itself.

ACTIVITY CONFIGURATION	DESCRIPTION
PDF Template Location	Location of Acroforms PDF, passed as String, ex. ”D:\form.pdf”
any other	Will be interpreted as PDF field name and String to fill the field.
PROCESS PARAMETERS	DESCRIPTION
PDF :UDT	This is the output parameter for the filled PDF form.
any other	Accessible via %s:parametername% in Activity Configuration.

Table 4.3: Activity Configuration and Process Parameters for PDF Filler Component

4.4 Encrypted Mailer Component

The *Encrypted Mailer* component utilizes the *Java Mail API*, specifically the *javax.mail* package, in conjunction with the *bcmail* and *bcprov* libraries, maintained by the Legion of the Bouncy Castle[5], to provide encrypted e-mail capabilities to an AristaFlow process. The component is used in the TDE process to ensure the encryption of the e-mail containing the notification letter as it is sent to the student administration. As the component is also capable of sending non-encrypted MIME multipart messages it is also used on every other activity in the process requiring the sending of an e-mail.

The component gets SMTP server configuration data from the activity configuration, such as server address, server port, user-name/password and TLS/SSL/encryption flags. Additionally to this, when running in S/MIME encryption mode, the component gets the location of a X.509 certificate from the activity configuration. This certificate is loaded and wrapped into a *X509Certificate* object, which is used to create a *SMIMEEnvelopedGenerator,* which in turn is used to encrypt MIME multipart messages.

After having created the object that can encrypt the message before sending, the next step the component undertakes is to piece together the e-mail message from the process parameters provided to it by the process instance. The component receives the main recipient, the recipient of the carbon copy, the subject, the body text, and optionally the attachment as process parameters. Encrypted Mailer then takes the body text and attachment and wraps them into separate

[4] http://www.itextpdf.com/
[5] http://www.bouncycastle.org/

MimeBodyPart objects and then wraps these two body parts into a *MimeMultiPart* object. Once this is complete a *MimeMessage* object is created, the recipients and sender are set and the *MimeMultiPart* is attached.

The message is now ready to be sent, but is not encrypted yet, it is still of the *multipart/mixed* type. Now the *bcmail* package, specifically the previously created *SMIMEEnvelopedGenerator,* converts the multipart/mixed into a *multipart/encrypted* message, by encrypting it using the X.509 certificate provided by the process. Once encryption is complete the *MimeMessage* is sent via the SMTP server to the recipient, who can decrypt the e-mail by having the correct private key and password for the public key contained in the X.509 certificate.

This complete process ensures that only the intended recipient of the e-mail, in the case of the TDE process the student administration, can read the mail and the attached notification letter.

ACTIVITY CONFIGURATION	DESCRIPTION
SMTP-Server	The IP or DNS address of the SMTP mail-server.
SMTP-Port	The port that the SMTP server uses.
From	The sender e-mail address, ex. this.is@te.st.
User	SMTP user name.
Password	SMTP password.
Enable TLS	This is used to control SSL/TLS usage.
Encrypt Message	This is used to control MIME encryption.
Certificate Location	Location of the certificate, ex. "D:\certificate.cer".
PROCESS PARAMETERS	DESCRIPTION
to :String	The primary recipient of the e-mail.
body :String	This String contains the e-mail body text.
subject :String	The e-mail subject.
cc :String	E-mail address of the recipient of the carbon copy of this e-mail.
attachment :UDT	The attachment in FileUDT format.

Table 4.4: Activity Configuration and Process Parameters for Encrypted Mailer Component

4.5 PDF Signer Component

The PDF Signer was created to allow the digital signing of a PDF document by AristaFlow. It uses the iTextPDF library that is also used in Section 4.3 and the *bcprov* library that was also used in the Encrypted Mailer (cf. Section 4.4). As the Encrypted Mailer can ensure that a sent e-mail can only be read by someone with the correct private key, the PDF Signer can ensure, on the receiving side, that the sender of the e-mail and its attached PDF file, is in fact the person he states to be. This is done by signing the PDF file with a private key and password. As the component is written in Java, the private key is in a Java *keystore* (JKS). The recipient has the sender's public key and can use this to verify that the keystore or private key, and the password used, were correct.

In summary, the two components PDF Signer and Encrypted Mailer give the users of the TDE process full confidence, that the e-mails and PDF files are on the one hand only read by the intended people and on the other are not replaced by fake files during e-mail transportation.

The component itself is relatively straightforward, it receives the location of the Java keystore in the file system, the keystore alias and password and the type of keystore. Additionally a digital signature of this kind needs a signature location and reason, for instance for the TDE process:

Location: Ulm, Germany

Reason: Digital Signing of Notification Letter

These two properties are also gathered from the activity configuration. Apart from this the component gets a PDF file as a process parameter. The component first creates a *PrivateKey* object from the keystore and password using methods from the bcprov library. Then it uses the iTextPDF library to read the PDF file from the process parameter and creates a *PDFStamper* object to manipulate the PDF file. The component calls the *PDFStamper*'s *createSignature()* method to create a digital signature from the *PrivateKey* object and apply it to the PDF file. The signed document is then handed back to the process instance, ending the component.

ACTIVITY CONFIGURATION	DESCRIPTION
KeystoreLocation	Location Java keystore, passed as String, ex. ”D:\keystore.ks”.
KeystoreType	The type of keystore in use, valid input is ”jks” or ”pkcs12”.
KeystoreAlias	The alias in use in the keystore.
KeystorePassword	The password for the selected alias.
SignatureLocation	The physical location the signing is taking place at.
SignatureReason	The reason for digitally signing the document.
PROCESS PARAMETERS	DESCRIPTION
PDF :UDT	This is the input/output parameter for the signed PDF form.

Table 4.5: Activity Configuration and Process Parameters for PDF Signer Component

4.6 The Finished Process

Having modeled the entire Thesis Deadline Extension process, and having implemented all necessary components, it is time to take a look at the result. The final AristaFlow process model can be viewed spread over Figures 4.3 and 4.4. The figures are on a double page and rotated to allow for printing at a readable size. The completed TDE process is fully capable of replacing the as-is process and provides a good basis for further process implementations by effectively providing a framework for secure communication at Ulm University through the PDF Signer and Encrypted Mailer. The source code for the components is provided in Appendix B to allow for process specific customizations to the components if the need arises.

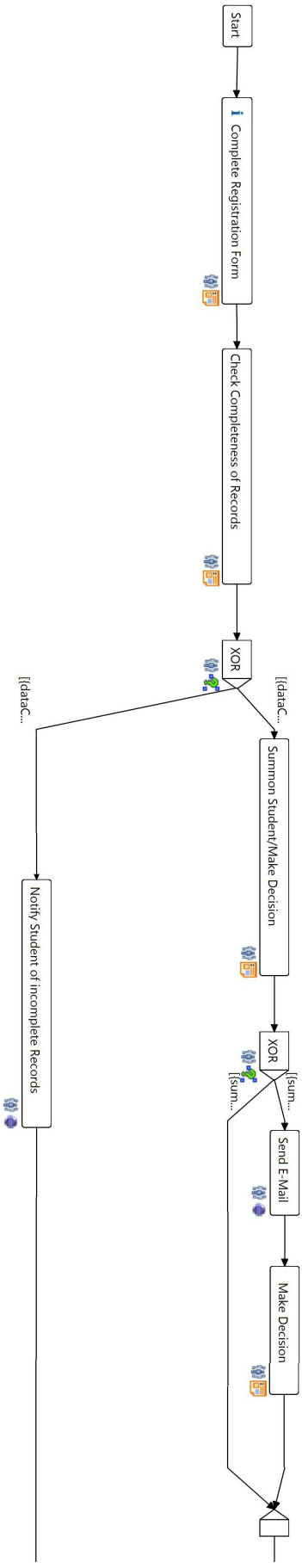


Figure 4.3: Complete TDE Process A

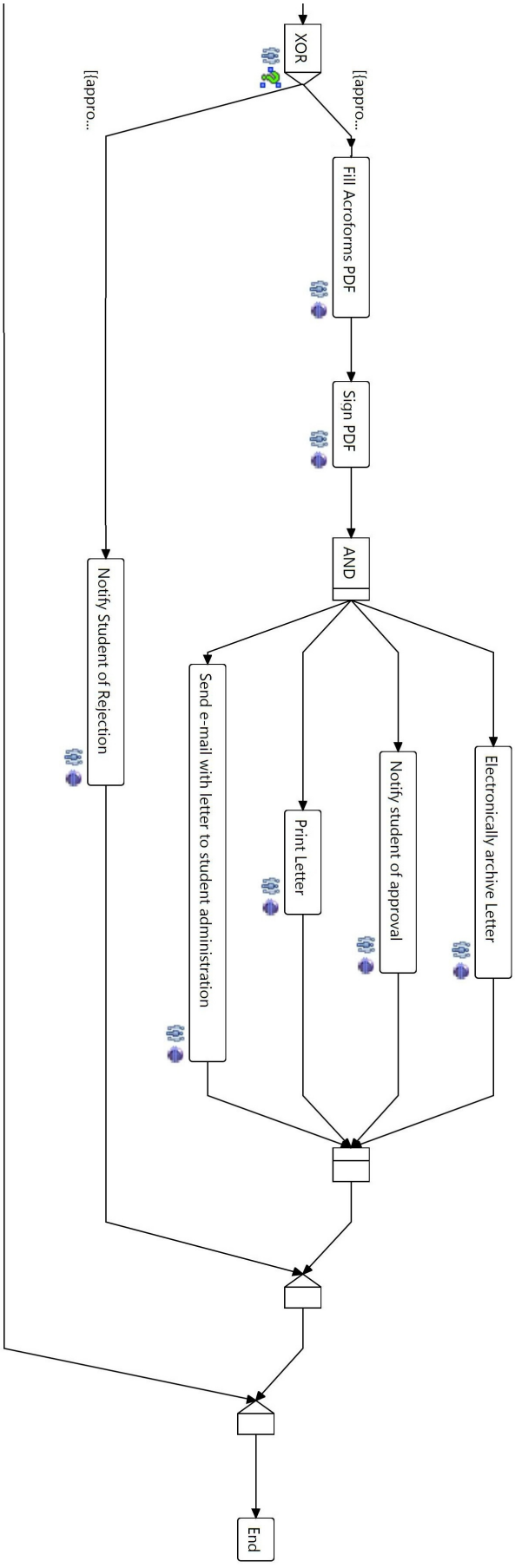


Figure 4.4: Complete TDE Process B

Table 4.6 gives a short overview over the nodes in the TDE process model and which components were used to realize their functions. Note that even though only one of the e-mails sent during the execution of the TDE process is actually encrypted, the Encrypted Mailer component was used for all e-mail sending nodes in the process model instead of using the *FormattedMailer* provided by AristaFlow. This is done to demonstrate the ability to send regular e-mails as well as encrypted ones with the Encrypted Mailer component and to keep maintenance of the process simpler due to the usage of only one e-mailing component.

COMPONENT	USAGE AT NODE
WJ&P Form Designer	Complete Registration Form
WJ&P Form Designer	Check Completeness of Records
WJ&P Form Designer	Summon Student/Make Decision
Encrypted Mailer	Notify Student of incomplete Records
Encrypted Mailer	Send E-Mail
WJ&P Form Designer	Make Decision
PDF Filler	Fill Acroforms PDF
PDF Signer	Sign PDF
Encrypted Mailer	Notify Student of Rejection
Encrypted Mailer	Notify Student of Approval
PDF Printer	Print Letter
Encrypted Mailer	Send E-Mail with Letter to Student Administration

Table 4.6: Component Usage in the TDE Process

5 Conclusion

Having analyzed, designed and implemented the optimized TDE process, according to [1] it is time for an evaluation. The evaluation of any project similar to this is to compare the costs and the benefits. The costs in this case are time spent on optimizing and automating the process, and the benefits are time saved by enacting the optimized process vs. the former process.

The major part of the time spent on the optimization of the TDE process was spent on extending the AristaFlow functionality to include the capabilities discussed in Section 3.3. The size of the implementation part of this thesis reflects this very well. If AristaFlow would ship with an activity for every possible task needed for optimizing a given process, the workload for the implementation would be reduced to:

- modeling the optimized activity with the correct activities
- creating or extending the org. model
- testing and deploying the finished process

As the AristaFlow BPM Suite supports the process implementer in many ways with these tasks [2], the margin of error and thus time needed in testing is a lot less than if the process is automated ”by hand”, i.e. without a PAIS[1] like AristaFlow. Also, again assuming that all needed activities are supplied to the implementer, the ability to model the process without having to touch code speeds up the implementation. This would be the case if for instance the other processes, detailed in Section 2.1 were implemented in AristaFlow.

Even though automation using AristaFlow is obviously preferable to automation without a PAIS it is still hard to judge the costs and benefits. The TDE process is simple enough to make a clear statement on the increased speed and reduced paperwork that result from the optimization. There are also few people involved in the process who need briefing on the new process, which reduces deployment complexity.

The single largest problem in judging the benefit of automation is the handling of exceptions in the process. Exceptions in this context do not necessarily mean run-time errors or computer failures that force process participants to redo steps or restart the process. Exceptions can also be unforeseen changes that need to be made to the process. Take for instance a student who enters his name incorrectly into the initial web form. The secretary checks the form but does not notice the error. The error is first noticed by the student when he receives the notification letter with his name on it. In the current, non automated, TDE process the compensation steps for this

[1] Process-Aware Information System

would be to inform the secretary, who would open up the last printed letter, change the name, and reprint it. In the automated process however, the process would have to be restarted, the chairman would have to approve the thesis deadline extension from his computer again and only then would the letter be printable with the corrected name.

One could just give the letter template to the secretary, but that would contradict the idea of the fully automated and therefore documented process execution. The solution here would be an ad-hoc change to the process instance, if it is still running [2]. But even this tiny change to the process instance would require that someone thought of the occurring exception beforehand, it would have to be modeled into the process itself as an ad-hoc change option for the user. The ad-hoc change model described in [2] supports this claim.

Altogether, the benefits in execution speed surely do outweigh the costs and time needed to plan and implement the process optimization as the implementation can be done relatively swiftly, with proper planning.

A Paper-Based Forms

An den Prüfungsausschuss Informatik
Prof. Dr. Manfred Reichert
Universität Ulm
Datenbanken und Informationssysteme
James-Franck-Ring
89069 Ulm

Name ..... Matr.Nr. .....
Anschrift .....
Tel.Nr ..... E-Mail .....
Studienrichtung ..... Anzahl Semester .....
Studiengang □ Diplom □ Bachelor □ Bachelor alt (PO 2001) □ Master

ANTRAG auf (bitte ankreuzen)

□ Fristverlängerung (Diplom) zur Ablegung von folgender Prüfung(en)..... bis zum.....
□ Fristverlängerung (Bachelor/Master) zum Erreichen der fehlenden .....Leistungspunkte bis zum .....
□ Fristverlängerung zur Abgabe der Diplomarbeit (mit Bestätigung des Betreuers) von ..... auf .....
□ Fristverlängerung zur Abgabe der Bachelorarbeit (mit Bestätigung des Betreuers) von .....auf .....
□ Fristverlängerung zur Abgabe der Masterarbeit (mit Bestätigung des Betreuers) von ..... auf .....
□ Zweite Wiederholung der Prüfung(en) (nur Diplom) .....
□ Anerkennung von externen Prüfungsleistungen
□ Änderung der Zurechnung von Prüfungsleistungen
□ Anerkennung von berufsbezogener Tätigkeit (Nachweis über 320 Stunden)

Anlagen: (bitte ankreuzen)

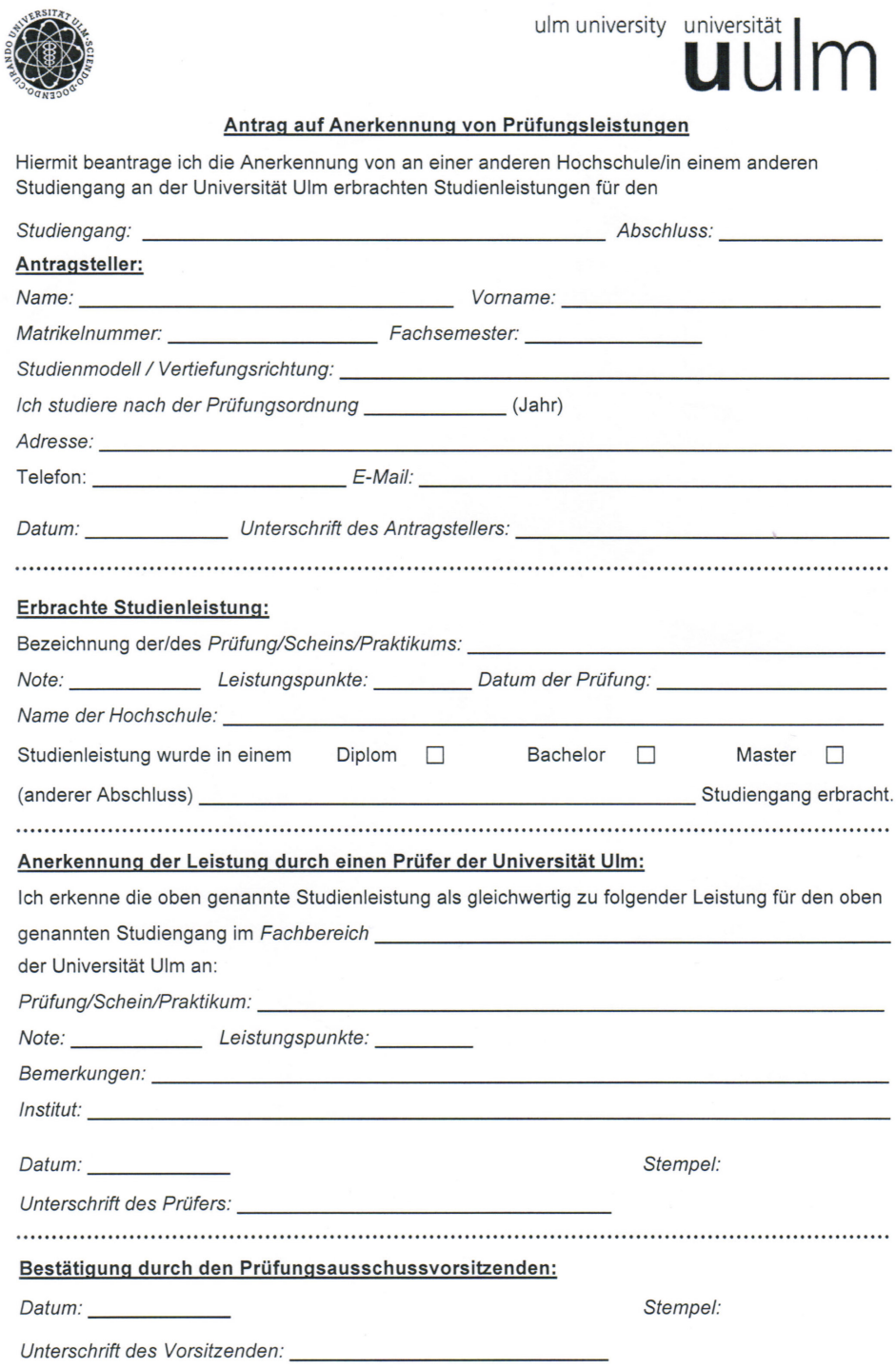
□ Bescheinigung über alle erbrachten Prüfungsleistungen (pdf-Ausdruck) vom ..... (bei jeder Art von Antrag)
□ Prüfungsplan (bei Diplom bei Prüfungsanerkennung)
□ Ärztliches Attest vom ..... über den Zeitraum von ..... bis.....
□ Arbeitsvertrag vom ..... . über den Zeitraum von .....bis.....
□ Bestätigung des Arbeitgebers vom ..... über den Zeitraum von .....bis.....
□ Unterlagen über Prüfungsstoff (bei Prüfungsanerkennung)
□ Gutachten zur Anerkennung externer Prüfungsleistungen
□ Sonstiges .....

□ Dies ist mein erster Antrag an den Prüfungsausschuss
□ Ich habe bereits einen oder mehrere Anträge an den Prüfungsausschuss gestellt, und zwar:
.....
.....

Ich versichere, die vorhergehenden Angaben vollständig und den Tatsachen entsprechend gemacht zu haben.

.....
Datum, Ort, Unterschrift

Figure A.1: Antrag Prüfungsauschuss Informatik



ulm university universität ulm

Antrag auf Anerkennung von Prüfungsleistungen

Hiermit beantrage ich die Anerkennung von an einer anderen Hochschule/in einem anderen Studiengang an der Universität Ulm erbrachten Studienleistungen für den

Studiengang: _____ *Abschluss:* _____

Antragsteller:

Name: _____ *Vorname:* _____

Matrikelnummer: _____ *Fachsemester:* _____

Studienmodell / Vertiefungsrichtung: _____

Ich studiere nach der Prüfungsordnung _____ (Jahr)

Adresse: _____

Telefon: _____ *E-Mail:* _____

Datum: _____ *Unterschrift des Antragstellers:* _____

Erbrachte Studienleistung:

Bezeichnung der/des *Prüfung/Scheins/Praktikums:* _____

Note: _____ *Leistungspunkte:* _____ *Datum der Prüfung:* _____

Name der Hochschule: _____

Studienleistung wurde in einem Diplom ☐ Bachelor ☐ Master ☐

(anderer Abschluss) _____ Studiengang erbracht.

Anerkennung der Leistung durch einen Prüfer der Universität Ulm:

Ich erkenne die oben genannte Studienleistung als gleichwertig zu folgender Leistung für den oben genannten Studiengang im *Fachbereich* _____ der Universität Ulm an:

Prüfung/Schein/Praktikum: _____

Note: _____ *Leistungspunkte:* _____

Bemerkungen: _____

Institut: _____

Datum: _____ *Stempel:*

Unterschrift des Prüfers: _____

Bestätigung durch den Prüfungsausschussvorsitzenden:

Datum: _____ *Stempel:*

Unterschrift des Vorsitzenden: _____

Figure A.2: Antrag auf Anerkennung von Prüfungsleistungen

ulm university universität
uulm

Antrag auf Studiengangwechsel für das

☐ Sommersemester 20____ ☐ Wintersemester 20____

Ende Antragsfrist: 30.05. im Sommersemester, 30.11. im Wintersemester. Für einen Wechsel in einen zulassungsbeschränkten Studiengang wenden Sie sich bitte an die Abteilung Zulassung.

Matrikelnr. _____

Name, Vorname: _____

Bisheriger Studiengang/Fachsemester: _____ / _____

Bisheriger Abschluss: _____

Ich bin Darlehensnehmer (L-Bank) ☐ ja (abgeschlossener Vertrag liegt vor) ☐ nein

Ich beantrage die Einschreibung für folgende/n zulassungsfreie/n Studiengang/-gänge:

Neuer Studiengang/Fachsemester: _____ / _____

2. Studiengang/Fachsemester (nur Lehramt) _____ / _____

Neuer Abschluss: _____

Bei einem Studiengangwechsel in das 2. und höhere Fachsemester ist die Begutachtung durch den zuständigen Prüfungsausschussvorsitzenden erforderlich (bei Lehramtsstudiengängen von beiden Vorsitzenden):

Begutachtung 1. Prüfungsausschussvorsitzende/r

Nach den mir vorgelegten Nachweisen aus dem bisher absolvierten Studium kann der/die Antragsteller/in in dem Studiengang, für den er/sie die Zulassung beantragt hat, höchstens dem ____. Fachsemester zugeordnet werden. Anzurechnende Leistungen werden gesondert bescheinigt.

_____ _____

Datum Unterschrift des Prüfungsausschussvorsitzenden

Begutachtung 2. Prüfungsausschussvorsitzende/r bei Wechsel in Lehramtsstudiengang

Nach den mir vorgelegten Nachweisen aus dem bisher absolvierten Studium kann der/die Antragsteller/in in dem Studiengang, für den er/sie die Zulassung beantragt hat, höchstens dem ____. Fachsemester zugeordnet werden. Anzurechnende Leistungen werden gesondert bescheinigt.

_____ _____

Datum Unterschrift des Prüfungsausschussvorsitzenden

Bei einem Studiengangwechsel im 3. oder höheren Semester ist der schriftliche Nachweis über eine auf den angestrebten Studiengang bezogene studienfachliche Beratung zu erbringen:

Studienfachliche Beratung

Der/die Antragsteller/in war am _____ bei einer studienfachlichen Beratung für den oben angeführten Studiengang.

_____ _____

Datum Unterschrift des Fachberaters

_____ _____

Datum Unterschrift des Antragstellers

Figure A.3: Antrag auf Studiengangwechsel

B Component Source Codes

Listing B.1: PDF Printer Source Code

```
1  package de.uulm.dbis.PDFPrinter;
2
3  import java.awt.Graphics;
4  import java.awt.Graphics2D;
5  import java.awt.Rectangle;
6  import java.awt.print.Book;
7  import java.awt.print.PageFormat;
8  import java.awt.print.Printable;
9  import java.awt.print.PrinterException;
10 import java.awt.print.PrinterJob;
11 import java.io.IOException;
12 import java.nio.ByteBuffer;
13 import java.util.logging.Logger;
14
15 import javax.print.PrintService;
16 import javax.print.attribute.HashPrintRequestAttributeSet;
17 import javax.print.attribute.PrintRequestAttributeSet;
18 import javax.print.attribute.standard.MediaSizeName;
19
20 import com.sun.pdfview.PDFFile;
21 import com.sun.pdfview.PDFPage;
22 import com.sun.pdfview.PDFRenderer;
23
24 import de.aristaflow.adept2.core.runtimemanager.executionenvironments.
      ExecutionEnvironment;
25 import de.aristaflow.adept2.extensions.datatypes.FileUDT;
26 import de.aristaflow.adept2.model.common.ActivityConfiguration;
27 import de.aristaflow.adept2.model.datamanagement.
      InvalidDataTypeException;
28 import de.aristaflow.adept2.model.datamanagement.
      NoSuchParameterException;
29 import de.aristaflow.adept2.model.execution.ActivityInstance;
30 import de.aristaflow.adept2.model.globals.ApplicationErrorCodes;
```

```
31 import de.aristaflow.adept2.model.runtimeenvironment.
       ApplicationEnvironmentException;
32 import de.aristaflow.adept2.model.runtimeenvironment.
       ApplicationFailedException;
33 import de.aristaflow.adept2.model.runtimeenvironment.DataContext;
34
35 /**
36  * Aristaflow component for printing pdfs
37  *
38  * @author Kevin Andrews
39  * @version 1.3 error throwing rework
40  * 1.2 added support for various paper sizes (A3,A5,Letter,Legal)
41  * 1.1 complete rework, now utilising com.sun.pdfview, no more adobe
        reader
42  */
43
44 public class PDFPrinter extends ExecutionEnvironment
45 {
46     public PDFPrinter(ActivityInstance activityInstance)
47     {
48         super(activityInstance);
49     }
50
51     @Override
52     public void run()
53     {
54         DataContext dataContext = sessionContext.getDataContext();
55         ActivityConfiguration activityConfiguration = activityInstance
               .getConfiguration();
56         String printerName;
57         String paperSize;
58         try
59         {
60             FileUDT fileUDT = new FileUDT(dataContext.
                   retrieveUDTParameterValue("PDF"));
61             printerName=activityConfiguration.getString("Printer Name"
                   );
62             paperSize=activityConfiguration.getString("Paper Size");
63             ByteBuffer bb = ByteBuffer.wrap(fileUDT.getData());
64             PDFFile pdfFile = new PDFFile(bb);
```

```
65              PDFPrintPage pages = new PDFPrintPage(pdfFile);
66              PrinterJob pjob = PrinterJob.getPrinterJob();
67              PrintRequestAttributeSet attributes = new
                    HashPrintRequestAttributeSet();
68              if(paperSize==null||paperSize.equals("A4"))attributes.add(
                    MediaSizeName.ISO_A4);
69              else if(paperSize.equals("A3"))attributes.add(
                    MediaSizeName.ISO_A3);
70              else if(paperSize.equals("A5"))attributes.add(
                    MediaSizeName.ISO_A5);
71              else if(paperSize.equals("Legal"))attributes.add(
                    MediaSizeName.NA_LEGAL);
72              else if(paperSize.equals("Letter"))attributes.add(
                    MediaSizeName.NA_LETTER);
73              else attributes.add(MediaSizeName.ISO_A4);
74              PageFormat pf = PrinterJob.getPrinterJob().getPageFormat(
                    attributes);
75              pjob.setJobName(fileUDT.getFileName());
76              Book book = new Book();
77              book.append(pages, pf, pdfFile.getNumPages());
78              pjob.setPageable(book);
79              PrintService[] services=PrinterJob.lookupPrintServices();
80              boolean found=false;
81              for(PrintService i:services)
82              {
83                  if(i.getName().equals(printerName)){
84                      found=true;
85                      pjob.setPrintService(i);
86                  }
87              }
88              if(found==false)Logger.getAnonymousLogger().warning("
                    Printing to default printer");
89              pjob.print();
90
91          } catch (IOException e)
92          {
93              throw new ApplicationFailedException("Piping failure", "",
                    ApplicationErrorCodes.IOEXCEPTION, e);
94          } catch (PrinterException e)
95          {
```

```
96              throw new ApplicationFailedException("Printing failure", "
                    ",ApplicationErrorCodes.
                    INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
97          } catch (InvalidDataTypeException e)
98          {
99              throw new ApplicationEnvironmentException("parameter has
                    wrong data type", ApplicationErrorCodes.
                    PARAMETER_UNEXPECTED_TYPE, e);
100         } catch (NoSuchParameterException e)
101         {
102             throw new ApplicationEnvironmentException("parameter
                    missing", ApplicationErrorCodes.PARAMETER_NOT_EXISTING
                    , e);
103         }
104
105
106
107
108         sessionContext.getRuntimeEnvironment().applicationClosed();
109     }
110 }
111 class PDFPrintPage implements Printable
112 {
113         private PDFFile file;
114
115         PDFPrintPage(PDFFile file)
116         {
117             this.file = file;
118         }
119
120         public int print(Graphics g, PageFormat format, int index)
                throws PrinterException
121         {
122             int pagenum = index + 1;
123
124             if (pagenum >= 1 && pagenum <= file.getNumPages())
125             {
126
127                 Graphics2D g2 = (Graphics2D) g;
128                 PDFPage page = file.getPage(pagenum);
```

```
129                 Rectangle imgbounds = new Rectangle((int) format.
                        getImageableX(), (int) format.getImageableY(), (
                        int)format.getImageableWidth(), (int)format.
                        getImageableHeight());
130                 PDFRenderer pgs = new PDFRenderer(page, g2, imgbounds,
                         null, null);
131                 try
132                 {
133                     page.waitForFinish();
134                     pgs.run();
135                 } catch (InterruptedException e)
136                 {
137                    e.printStackTrace();
138                 }
139                 return PAGE_EXISTS;
140             } else
141             {
142                 return NO_SUCH_PAGE;
143             }
144         }
145
146 }
```

Listing B.2: PDF Filler Source Code

```
1  package de.uulm.dbis.PDFFiller;
2
3  import java.io.ByteArrayOutputStream;
4  import java.io.File;
5  import java.io.IOException;
6  import java.text.DateFormat;
7  import java.util.HashMap;
8  import java.util.Map;
9  import java.util.Set;
10 import com.itextpdf.text.DocumentException;
11 import com.itextpdf.text.pdf.AcroFields;
12 import com.itextpdf.text.pdf.PdfReader;
13 import com.itextpdf.text.pdf.PdfStamper;
14
15 import de.aristaflow.adept2.core.runtimemanager.executionenvironments.
       ExecutionEnvironment;
```

```
16 import de.aristaflow.adept2.extensions.datatypes.FileUDT;
17 import de.aristaflow.adept2.model.common.ActivityConfiguration;
18 import de.aristaflow.adept2.model.common.systemdata.
       BasicSystemDataFormatter;
19 import de.aristaflow.adept2.model.common.systemdata.SystemDataTools;
20 import de.aristaflow.adept2.model.datamanagement.
       InvalidDataTypeException;
21 import de.aristaflow.adept2.model.datamanagement.
       NoSuchParameterException;
22 import de.aristaflow.adept2.model.execution.ActivityInstance;
23 import de.aristaflow.adept2.model.globals.ActivityConstants.AccessType
       ;
24 import de.aristaflow.adept2.model.globals.ApplicationErrorCodes;
25 import de.aristaflow.adept2.model.processmodel.ProcessModelParameter;
26 import de.aristaflow.adept2.model.runtimeenvironment.
       ApplicationEnvironmentException;
27 import de.aristaflow.adept2.model.runtimeenvironment.
       ApplicationFailedException;
28 import de.aristaflow.adept2.model.runtimeenvironment.DataContext;
29
30 /**
31  * Aristaflow component for filling pdfs with forms
32  *
33  * @author Kevin Andrews
34  * @version 2.0 complete input rework (changed to parameterized
        strings)
35  * 1.5 error throwing rework
36  * 1.4 flatten pdf before output
37  * 1.3 support static lifecycle xfa forms
38  * 1.2 variable amount of input parameters through parameter
        configurations
39  * 1.1 IO handling with FileUDT
40  */
41 public class PDFFiller extends ExecutionEnvironment
42 {
43
44     private AcroFields fields;
45     private Boolean usesXFA;
46
47     public PDFFiller(ActivityInstance activityInstance)
```

```
48      {
49          super(activityInstance);
50      }
51
52      @Override
53      public void run()
54      {
55          try
56          {
57              usesXFA = false;
58              ByteArrayOutputStream out = null;
59              FileUDT fileUDT = null;
60              DataContext dataContext = sessionContext.getDataContext();
61              ActivityConfiguration activityConfiguration =
                    activityInstance.getConfiguration();
62
63              String pdfTemplateLocation = activityConfiguration.
                    getString("PDF Template Location");
64              /*
65               * create pdfreader, use it to read pdf from the
                     datacontext's input
66               */
67              String fileLocation = activityConfiguration.getString("PDF
                     Template Location");
68              File file = new File(fileLocation);
69              fileUDT = new FileUDT(file);
70              out = new ByteArrayOutputStream();
71              PdfReader reader = new PdfReader(fileUDT.getData());
72              PdfStamper stamper = new PdfStamper(reader, out);
73              fields = stamper.getAcroFields();
74              usesXFA = fields.getXfa().isXfaPresent();
75              if (usesXFA)
76              {
77                  fields.removeXfa();
78              }
79
80              Set<String> fieldNames = activityConfiguration.
                    getAllEntries();
81              fieldNames.remove(pdfTemplateLocation);
82
```

```
83              Map<String, String> fieldNamesAndParameterizedStrings =
                    new HashMap<String, String>();
84              for (String s : fieldNames)
85              {
86                  fieldNamesAndParameterizedStrings.put(s,
                        activityConfiguration.getString(s));
87              }
88
89              Set<ProcessModelParameter> activityParameters =
                    activityInstance.getParameters(AccessType.READ);
90              Map<String, Object> parsedActivityParameters = new HashMap
                    <String, Object>();
91              for (ProcessModelParameter p : activityParameters)
92              {
93
94                  switch (p.getDataType())
95                  {
96                      case STRING:
97
98                          parsedActivityParameters.put(p.getName(),
                                dataContext.retrieveStringParameterValue(p
                                .getName()));
99
100                         break;
101                     case INTEGER:
102                         parsedActivityParameters.put(p.getName(),
                                dataContext.retrieveIntegerParameterValue(
                                p.getName()) + "");
103
104                         break;
105                     case DATE:
106
107                         parsedActivityParameters.put(p.getName(),
                                DateFormat.getDateInstance(DateFormat.
                                MEDIUM).format(dataContext.
                                retrieveDateParameterValue(p.getName())));
108
109                         break;
110                 }
111
```

```
112             }
113             Map<String, String> fieldNamesAndFilledStrings = new
                    HashMap<String, String>();
114             for (String s : fieldNamesAndParameterizedStrings.keySet()
                    )
115             {
116                 fieldNamesAndFilledStrings.put(s, SystemDataTools.
                        format(BasicSystemDataFormatter.class,
                        fieldNamesAndParameterizedStrings.get(s),
                        parsedActivityParameters));
117             }
118
119             for (String s : fieldNamesAndFilledStrings.keySet())
120             {
121
122                 fields.setField(getRealFieldName(s),
                        fieldNamesAndFilledStrings.get(s));
123
124             }
125
126             stamper.setFormFlattening(true);
127             stamper.close();
128
129             /*
130              * write the filled pdf to a new FileUDT and store it to
                     the
131              * datacontext
132              */
133             fileUDT = new FileUDT(out.toByteArray(), fileUDT.
                    getFileName(), fileUDT.getEncoding(), fileUDT.
                    getMimetype(), out.toByteArray().length);
134
135             out.close();
136             dataContext.storeUDTParameterValue("PDF", fileUDT.getAsXML
                    ());
137
138             sessionContext.getRuntimeEnvironment().applicationClosed()
                    ;
139         }
140         catch (InvalidDataTypeException e)
```

```
141         {
142             throw new ApplicationEnvironmentException("Invalid
                    parameter data type", ApplicationErrorCodes.
                    PARAMETER_UNEXPECTED_TYPE, e);
143         }
144         catch (NoSuchParameterException e)
145         {
146             throw new ApplicationEnvironmentException("Parameter
                    missing", ApplicationErrorCodes.PARAMETER_NOT_EXISTING
                    , e);
147         }
148         catch (DocumentException e)
149         {
150             throw new ApplicationFailedException("PDF could not be
                    read", "", ApplicationErrorCodes.
                    INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
151         }
152         catch (IOException e)
153         {
154             throw new ApplicationFailedException("PDF could not be
                    read", "", ApplicationErrorCodes.IOEXCEPTION, e);
155         }
156
157     }
158
159     String getRealFieldName(String fieldName)
160     {
161         if (!usesXFA)
162         {
163             return fieldName;
164         }
165         for (String i : fields.getFields().keySet())
166         {
167             if (i.contains(fieldName))
168             {
169                 return i;
170             }
171         }
172         return null;
173     }
```

```
174
175 }
```

Listing B.3: Encrypted Mailer Source Code

```
1  package de.uulm.dbis.EncryptedMailer;
2
3  import java.io.FileInputStream;
4  import java.io.FileNotFoundException;
5  import java.io.IOException;
6  import java.security.Security;
7  import java.security.cert.CertificateException;
8  import java.security.cert.CertificateFactory;
9  import java.security.cert.X509Certificate;
10 import java.util.ArrayList;
11 import java.util.Properties;
12 import java.util.Set;
13 import java.util.logging.Logger;
14
15 import javax.activation.DataHandler;
16 import javax.mail.Message;
17 import javax.mail.MessagingException;
18 import javax.mail.PasswordAuthentication;
19 import javax.mail.Session;
20 import javax.mail.Transport;
21 import javax.mail.internet.AddressException;
22 import javax.mail.internet.InternetAddress;
23 import javax.mail.internet.MimeBodyPart;
24 import javax.mail.internet.MimeMessage;
25 import javax.mail.internet.MimeMultipart;
26
27 import org.bouncycastle.cms.CMSAlgorithm;
28 import org.bouncycastle.cms.CMSException;
29 import org.bouncycastle.cms.jcajce.JceCMSContentEncryptorBuilder;
30 import org.bouncycastle.cms.jcajce.JceKeyTransRecipientInfoGenerator;
31 import org.bouncycastle.jce.provider.BouncyCastleProvider;
32 import org.bouncycastle.mail.smime.SMIMEEnvelopedGenerator;
33 import org.bouncycastle.mail.smime.SMIMEException;
34 import org.bouncycastle.operator.OperatorCreationException;
35
```

```
36 import de.aristaflow.adept2.core.runtimemanager.executionenvironments.
       ExecutionEnvironment;
37 import de.aristaflow.adept2.extensions.datatypes.FileUDT;
38 import de.aristaflow.adept2.model.common.ActivityConfiguration;
39 import de.aristaflow.adept2.model.datamanagement.
       InvalidDataTypeException;
40 import de.aristaflow.adept2.model.datamanagement.
       NoSuchParameterException;
41 import de.aristaflow.adept2.model.datamanagement.UDTValue;
42 import de.aristaflow.adept2.model.execution.ActivityInstance;
43 import de.aristaflow.adept2.model.globals.ActivityConstants.AccessType
       ;
44 import de.aristaflow.adept2.model.globals.ApplicationErrorCodes;
45 import de.aristaflow.adept2.model.processmodel.ProcessModelParameter;
46 import de.aristaflow.adept2.model.runtimeenvironment.
       ApplicationEnvironmentException;
47 import de.aristaflow.adept2.model.runtimeenvironment.
       ApplicationFailedException;
48 import de.aristaflow.adept2.model.runtimeenvironment.DataContext;
49
50 /**
51  * Aristaflow component for encrypting mails with a x.509.cer file
52  *
53  * @author Kevin Andrews
54  * @version 1.4 error throwing rework 1.3 made encryption,tls,cc,
        attachment
55  *          optional 1.2 changed to/cc/subject/body to data objects
        1.1 IO
56  *          handling with FileUDT
57  */
58 public class EncryptedMailer extends ExecutionEnvironment
59 {
60
61     public EncryptedMailer(ActivityInstance activityInstance)
62     {
63         super(activityInstance);
64     }
65
66     /* initialize all configuration variables as static for faster
           access */
```

```
67      static String user;
68      static String pw;
69      static String from;
70      static String cc;
71      static String to;
72      static String host;
73      static String port;
74      static Boolean startTLS = false;
75      static String body;
76      static String subject;
77      static String certificateLocation;
78      static Boolean encryption = false;
79
80      @Override
81      public void run()
82      {
83          /* get mail-server settings from activity configuration */
84          ActivityConfiguration activityConfiguration = activityInstance
                .getConfiguration();
85          certificateLocation = activityConfiguration.getString("
                Certificate Location");
86          user = activityConfiguration.getString("User");
87          pw = activityConfiguration.getString("Password");
88          from = activityConfiguration.getString("From");
89          host = activityConfiguration.getString("SMTP-Server");
90          port = activityConfiguration.getString("SMTP-Port");
91          encryption = activityConfiguration.getBoolean("Encrypt Message
                ");
92          encryption = encryption == null ? false : encryption;
93          startTLS = activityConfiguration.getBoolean("Enable TLS");
94          startTLS = startTLS == null ? false : startTLS;
95
96          /* get attachment file and comment string from Data Context */
97          FileUDT fileUDT = null;
98          try
99          {
100             DataContext dataContext = sessionContext.getDataContext();
101             Set<ProcessModelParameter> parameters = activityInstance.
                    getParameters(AccessType.READ);
```

```
102             ArrayList<String> parameternames = new ArrayList<String>()
                    ;
103             for (ProcessModelParameter p : parameters)
104             {
105                 parameternames.add(p.getName());
106             }
107             to= dataContext.retrieveStringParameterValue("to");
108             body = dataContext.retrieveStringParameterValue("body");
109             subject = dataContext.retrieveStringParameterValue("
                    subject");
110
111             if (parameternames.contains("cc"))
112             {
113                 Logger.getAnonymousLogger().warning("found cc");
114                 cc = dataContext.retrieveStringParameterValue("cc");
115             }
116
117             if (parameternames.contains("attachment"))
118             {
119                 Logger.getAnonymousLogger().warning("found attachment"
                        );
120                 UDTValue temp = dataContext.retrieveUDTParameterValue(
                        "attachment");
121                 if (temp != null)
122                 {
123                     fileUDT = new FileUDT(temp);
124                 }
125             }
126
127         }
128         catch (InvalidDataTypeException e)
129         {
130             throw new ApplicationEnvironmentException("Parameter is of
                     the wrong data-type", ApplicationErrorCodes.
                    PARAMETER_WRONG_DATA_TYPE, e);
131         }
132         catch (NoSuchParameterException e)
133         {
```

```
134             throw new ApplicationEnvironmentException("Parameter
                    missing", ApplicationErrorCodes.PARAMETER_NOT_EXISTING
                    , e);
135         }
136
137         /* add bouncycastle to security providers */
138         Security.addProvider(new BouncyCastleProvider());
139
140         /*
141          * load certificate and create a generator for SMIME envelopes
                for later
142          * wrapping
143          */
144         FileInputStream certInput;
145         SMIMEEnvelopedGenerator encrypter = null;
146         if (encryption == true)
147         {
148
149             try
150             {
151                 certInput = new FileInputStream(certificateLocation);
152                 CertificateFactory cf = CertificateFactory.getInstance
                        ("X.509");
153                 X509Certificate cert = (X509Certificate) cf.
                        generateCertificate(certInput);
154                 certInput.close();
155                 encrypter = new SMIMEEnvelopedGenerator();
156                 encrypter.addRecipientInfoGenerator(new
                        JceKeyTransRecipientInfoGenerator(cert).
                        setProvider("BC"));
157             }
158             catch (FileNotFoundException e)
159             {
160                 throw new ApplicationFailedException("Certificate not
                        found", "", ApplicationErrorCodes.
                        PARAMETER_URI_TARGET_MISSING, e);
161             }
162             catch (CertificateException e)
163             {
```

```
164                 throw new ApplicationFailedException("Certificate
                        error", "", ApplicationErrorCodes.
                        INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
165             }
166             catch (IOException e)
167             {
168                 throw new ApplicationFailedException("Certificate can
                        not be read", "", ApplicationErrorCodes.
                        IOEXCEPTION, e);
169             }
170             catch (IllegalArgumentException e)
171             {
172                 throw new ApplicationFailedException("Illegal argument
                        ", "", ApplicationErrorCodes.ILLEGAL_ARGUMENT, e);
173             }
174             catch (OperatorCreationException e)
175             {
176                 throw new ApplicationFailedException("Encrypting error
                        ", "", ApplicationErrorCodes.
                        INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
177             }
178         }
179         /*
180          * create and fill individual body parts (text,attachment),
                add them to
181          * a multipart, encrypt and add to MIMEmessage
182          */
183         MimeMessage message = null;
184         try
185         {
186             MimeMultipart multi = new MimeMultipart();
187             MimeBodyPart text = new MimeBodyPart();
188             text.setText(body);
189             multi.addBodyPart(text);
190             if (fileUDT != null)
191             {
192                 MimeBodyPart attachment = new MimeBodyPart();
193                 attachment.setDataHandler(new DataHandler(fileUDT.
                        getData(), fileUDT.getMimetype()));
194                 attachment.setFileName(fileUDT.getFileName());
```

```
195                 if (fileUDT != null)
196                 {
197                     multi.addBodyPart(attachment);
198                 }
199             }
200 
201             message = new MimeMessage(getSession());
202             message.setFrom(new InternetAddress(from));
203             message.setRecipient(Message.RecipientType.TO, new
                    InternetAddress(to));
204             if (cc != null)
205             {
206                 message.setRecipient(Message.RecipientType.CC, new
                        InternetAddress(cc));
207             }
208             message.setSubject(subject);
209             message.setContent(multi);
210             if (encryption == true)
211             {
212                 MimeBodyPart complete = encrypter.generate(message,
                        new JceCMSContentEncryptorBuilder(CMSAlgorithm.
                        RC2_CBC, 40).setProvider("BC").build());
213                 message.setContent(complete.getContent(), complete.
                        getContentType());
214             }
215         }
216         catch (AddressException e)
217         {
218             throw new ApplicationFailedException("Email address error"
                    , "", ApplicationErrorCodes.
                    INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
219         }
220         catch (MessagingException e)
221         {
222             throw new ApplicationFailedException("Messagingexception
                    in javax.mail", "", ApplicationErrorCodes.
                    INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
223         }
224         catch (IOException e)
225         {
```

```
226             throw new ApplicationFailedException("Attachment IO Error"
                    , "", ApplicationErrorCodes.IOEXCEPTION, e);
227         }
228         catch (SMIMEException e)
229         {
230             throw new ApplicationFailedException("SMIME encryption
                    error", "", ApplicationErrorCodes.
                    INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
231         }
232         catch (CMSException e)
233         {
234             throw new ApplicationFailedException("Encryption error", "
                    ", ApplicationErrorCodes.
                    INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
235         }
236
237         /* actual sending of created+encrypted MIMEmessage */
238         try
239         {
240             Transport.send(message);
241         }
242         catch (MessagingException e)
243         {
244             throw new ApplicationFailedException("Sending error", "",
                    ApplicationErrorCodes.
                    INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
245         }
246         /* end component */
247         sessionContext.getRuntimeEnvironment().applicationClosed();
248     }
249
250     /**
251      * helper method for getting a MIME session with optional tls
            support
252      */
253     private static Session getSession()
254     {
255         SMTPAuthenticator authenticator = new SMTPAuthenticator();
256         Properties properties = new Properties();
257         properties.setProperty("mail.smtp.from", user);
```

```
258         properties.setProperty("mail.smtp.port", port);
259         properties.setProperty("mail.smtp.host", host);
260         properties.setProperty("mail.smtp.auth", "true");
261         if (startTLS)
262         {
263             properties.setProperty("mail.smtp.starttls.enable", "true"
                );
264         }
265         return Session.getInstance(properties, authenticator);
266     }
267
268     /**
269      * implementation of javax.mail.Authenticator creates an
            authenticator with
270      * our user and pw combination
271      */
272     static class SMTPAuthenticator extends javax.mail.Authenticator
273     {
274         @Override
275         protected PasswordAuthentication getPasswordAuthentication()
276         {
277             return new PasswordAuthentication(user, pw);
278         }
279     }
280 }
```

Listing B.4: PDF Signer Source Code

```
1  package de.uulm.dbis.PDFSigner;
2
3  import java.io.ByteArrayOutputStream;
4  import java.io.FileInputStream;
5  import java.io.FileNotFoundException;
6  import java.io.IOException;
7  import java.security.KeyStore;
8  import java.security.KeyStoreException;
9  import java.security.NoSuchAlgorithmException;
10 import java.security.NoSuchProviderException;
11 import java.security.PrivateKey;
12 import java.security.Security;
13 import java.security.UnrecoverableKeyException;
```

```
14 import java.security.cert.Certificate;
15 import java.security.cert.CertificateException;
16
17 import org.bouncycastle.jce.provider.BouncyCastleProvider;
18
19 import com.itextpdf.text.DocumentException;
20 import com.itextpdf.text.Rectangle;
21 import com.itextpdf.text.pdf.PdfReader;
22 import com.itextpdf.text.pdf.PdfSignatureAppearance;
23 import com.itextpdf.text.pdf.PdfStamper;
24
25 import de.aristaflow.adept2.core.runtimemanager.executionenvironments.
       ExecutionEnvironment;
26 import de.aristaflow.adept2.extensions.datatypes.FileUDT;
27 import de.aristaflow.adept2.model.common.ActivityConfiguration;
28 import de.aristaflow.adept2.model.datamanagement.
       InvalidDataTypeException;
29 import de.aristaflow.adept2.model.datamanagement.
       NoSuchParameterException;
30 import de.aristaflow.adept2.model.execution.ActivityInstance;
31 import de.aristaflow.adept2.model.globals.ApplicationErrorCodes;
32 import de.aristaflow.adept2.model.runtimeenvironment.
       ApplicationEnvironmentException;
33 import de.aristaflow.adept2.model.runtimeenvironment.
       ApplicationFailedException;
34 import de.aristaflow.adept2.model.runtimeenvironment.DataContext;
35
36 /**
37  * Aristaflow component for signing pdf files with a keystore +
        certificate
38  * @author Kevin Andrews
39  * @version 1.2 error throwing rework
40  * 1.1 IO handling with FileUDT
41  */
42 public class PDFSigner extends ExecutionEnvironment
43 {
44
45
46     public PDFSigner(ActivityInstance activityInstance)
47     {
```

```
48          super(activityInstance);
49      }
50
51      @Override
52      public void run()
53      {
54
55          /* get pdf file from Data Context */
56          DataContext dataContext = sessionContext.getDataContext();
57          FileUDT fileUDT = null;
58          try
59          {
60              fileUDT = new FileUDT(dataContext.
                    retrieveUDTParameterValue("PDF"));
61          } catch (InvalidDataTypeException e)
62          {
63              throw new ApplicationEnvironmentException("Parameter has
                    invalid data type!", ApplicationErrorCodes.
                    PARAMETER_UNEXPECTED_TYPE, e);
64          } catch (NoSuchParameterException e)
65          {
66              throw new ApplicationEnvironmentException("Parameter
                    missing!", ApplicationErrorCodes.
                    PARAMETER_NOT_EXISTING, e);
67          }
68
69          /* get configuration strings from Activity Configuration */
70          ActivityConfiguration activityConfiguration = activityInstance
                .getConfiguration();
71          String keystoreLocation = activityConfiguration.getString("
                KeystoreLocation");
72          String keystoreAlias = activityConfiguration.getString("
                KeystoreAlias");
73          String keystoreType = activityConfiguration.getString("
                KeystoreType");
74          String keystorePassword = activityConfiguration.getString("
                KeystorePassword");
75          String signatureLocation = activityConfiguration.getString("
                SignatureLocation");
```

```
76          String signatureReason = activityConfiguration.getString("
                SignatureReason");
77
78          /* add bouncycastle security provider */
79          Security.addProvider(new BouncyCastleProvider());
80
81          /* load keystore from file use password and get privatekey
                from ks */
82          KeyStore ks;
83          Certificate[] chain = null;
84          PrivateKey key = null;
85          try
86          {
87              ks = KeyStore.getInstance(keystoreType, "BC");
88              ks.load(new FileInputStream(keystoreLocation),
                    keystorePassword.toCharArray());
89              key = (PrivateKey) ks.getKey(keystoreAlias,
                    keystorePassword.toCharArray());
90              chain = ks.getCertificateChain(keystoreAlias);
91          } catch (KeyStoreException e)
92          {
93              throw new ApplicationFailedException("KeyStoreException",
                    "",ApplicationErrorCodes.
                    INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
94          } catch (NoSuchProviderException e)
95          {
96              throw new ApplicationFailedException("
                    NoSuchProviderException", "",ApplicationErrorCodes.
                    INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
97          } catch (NoSuchAlgorithmException e)
98          {
99              throw new ApplicationFailedException("
                    NoSuchAlgorithmException", "",ApplicationErrorCodes.
                    INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
100         } catch (CertificateException e)
101         {
102             throw new ApplicationFailedException("CertificateException
                    ", "",ApplicationErrorCodes.
                    INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
103         } catch (FileNotFoundException e)
```

```
104             {
105                 throw new ApplicationFailedException("
                        FileNotFoundException", "",ApplicationErrorCodes.
                        IOEXCEPTION, e);
106             } catch (IOException e)
107             {
108                 throw new ApplicationFailedException("IOException","",
                        ApplicationErrorCodes.IOEXCEPTION, e);
109             } catch (UnrecoverableKeyException e)
110             {
111                 throw new ApplicationFailedException("Printing failure", "
                        ",ApplicationErrorCodes.
                        INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
112             }
113
114             /*
115              * create pdfreader, use it to read pdf from the datacontext's
                     input
116              * FileUDT
117              */
118             PdfReader reader;
119             try
120             {
121                 reader = new PdfReader(fileUDT.getData());
122             } catch (IOException e)
123             {
124                 throw new ApplicationEnvironmentException("IOException",
                        ApplicationErrorCodes.IOEXCEPTION, e);
125             }
126
127             /*
128              * create pdf signature and set its look and feel then sign
                    the document
129              * with close()
130              */
131             PdfStamper stp;
132             ByteArrayOutputStream out = null;
133             try
134             {
135
```

```
136             out = new ByteArrayOutputStream();
137             stp = PdfStamper.createSignature(reader, out, '\0');
138             PdfSignatureAppearance sap = stp.getSignatureAppearance();
139             sap.setCrypto(key, chain, null, PdfSignatureAppearance.
                    WINCER_SIGNED);
140             sap.setReason(signatureReason);
141             sap.setLocation(signatureLocation);
142             sap.setVisibleSignature(new Rectangle(100, 100, 200, 200),
                     1, null);
143             stp.close();
144
145         } catch (DocumentException e)
146         {
147             throw new ApplicationFailedException("DocumentException",
                    "",ApplicationErrorCodes.
                    INSTANCE_ABORT_DUE_TO_INTERNAL_ERROR, e);
148         } catch (IOException e)
149         {
150             throw new ApplicationFailedException("IOException","",
                    ApplicationErrorCodes.IOEXCEPTION, e);
151         }
152         /* write the filled pdf to a new FileUDT and store it to the
                datacontext */
153         fileUDT=new FileUDT(out.toByteArray(), fileUDT.getFileName(),
                fileUDT.getEncoding(), fileUDT.getMimetype(), out.
                toByteArray().length);
154         try
155         {
156             out.close();
157             dataContext.storeUDTParameterValue("PDF", fileUDT.getAsXML
                    ());
158         } catch (IOException e)
159         {
160             throw new ApplicationFailedException("IOException","",
                    ApplicationErrorCodes.IOEXCEPTION, e);
161         } catch (InvalidDataTypeException e)
162         {
163             throw new ApplicationEnvironmentException("Output
                    parameter has invalid data type",
                    ApplicationErrorCodes.PARAMETER_UNEXPECTED_TYPE, e);
```

```
164             } catch (NoSuchParameterException e)
165             {
166                 throw new ApplicationEnvironmentException("Output
                        parameter missing", ApplicationErrorCodes.
                        PARAMETER_NOT_EXISTING, e);
167             }
168             /* end component */
169             sessionContext.getRuntimeEnvironment().applicationClosed();
170
171         }
172
173 }
```

Bibliography

[1] ARLBJORN, J. ; HAUG, A. : *Business Process Optimization*. Academica, 2010

[2] DADAM, P. ; REICHERT, M. ; RINDERLE-MA, S. ; GOESER, K. ; KREHER, U. ; JURISCH, M. : Von ADEPT zur AristaFlow BPM Suite - Eine Vision wird Realität: "Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen / University of Ulm. 2009. – Forschungsbericht

[3] DADAM, P. ; REICHERT, M. ; RINDERLE-MA, S. ; LANZ, A. ; PRYSS, R. ; PREDESCHLY, M. ; KOLB, J. ; LY, L. T. ; JURISCH, M. ; KREHER, U. ; GOESER, K. : From ADEPT to AristaFlow BPM Suite: A Research Vision has become Reality / University of Ulm. 2009. – Forschungsbericht

[4] LANZ, A. ; KREHER, U. ; REICHERT, M. ; DADAM, P. : Enabling Process Support for Advanced Applications with the AristaFlow BPM Suite. 2010. – Forschungsbericht

[5] LANZ, A. ; REICHERT, M. ; DADAM, P. : Making Business Process Implementations Flexible and Robust: Error Handling in the AristaFlow BPM Suite. 2010. – Forschungsbericht

[6] MENDLING, J. ; REIJERS, H. ; AALST, W. van d.: Seven Process Modeling Guidelines (7PMG). In: *Information and Software Technology* (2010)

[7] MICHELER, F. : *Konzeption, Implementierung und Integration einer Komponente für die Erstellung intelligenter Formulare*. 2009

[8] WHITE, S. : Introduction to BPMN. In: *IBM Cooperation* (2004)

Name: Kevin Andrews Matrikelnummer: 671626

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Kevin Andrews