Automating form-based processes through annotation

Author:
Kim, Sung Wook

Publication Date:
2013

DOI:
https://doi.org/10.26190/unsworks/16393

License:
https://creativecommons.org/licenses/by-nc-nd/3.0/au/
Link to license to see what you are allowed to do with this resource.

Downloaded from http://hdl.handle.net/1959.4/52948 in https://unsworks.unsw.edu.au on 2023-10-14
Automating Form-based Processes through Annotation

Sung Wook Kim

Supervisor: Dr. Hye-Young Paik

School of Computer Science and Engineering
University of New South Wales
Sydney 2052, Australia

A thesis submitted in fulfillment of the requirements
for the degree of Masters of Engineering
August 2012
Abstract 350 words maximum: (PLEASE TYPE)

Business process is a collection of tasks performed to achieve a certain business objective within an organisation. Due to the complexity and variability of business processes, organisations use information technology to support their business processes. Business Process Management Systems (BPMS) are widely accepted software systems that enable organisations to automate and continuously improve business processes. Although the benefits of BPMS have been widely recognised, there is still a large portion of business processes that are not adequately addressed by these systems. These types of processes make up the so-called the Long Tail of processes, i.e., highly customised processes that are unique to individual organisations or a small number of workers. Many organisations do not see significant Return On Investment on customising BPMS, or implementing solutions to automate Long Tail processes. So they often rely on using paper-based forms for running their processes due to its low cost, flexibility, and easy-of-use. However, using paper-based forms puts a burden of doing tedious manual work to the end users, restraining them from focusing on more important tasks.

In this thesis, we propose a pragmatic approach for automating form-based processes, in which annotation is used to help collect necessary information from the users to ascertain basic data flows and process execution paths. We have implemented the approach in the system called EzyForms, a framework for supporting form-based processes. The architecture supports an end-to-end life-cycle of forms, starting from its creation, annotation, and ultimately to its execution in a process.

The proposed approach and framework can help administration officers with little or no technical background to automate the Long Tail of the processes which in turn helps the end users to execute the process in a more efficient manner.
ORIGINALITY STATEMENT

'I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.'

Signed

Date

22/10/2013
COPYRIGHT STATEMENT

'I hereby grant the University of New South Wales or its agents the right to archive and to make available my thesis or dissertation in whole or part in the University libraries in all forms of media, now or here after known, subject to the provisions of the Copyright Act 1968. I retain all proprietary rights, such as patent rights. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

I also authorise University Microfilms to use the 350 word abstract of my thesis in Dissertation Abstract International (this is applicable to doctoral theses only).

I have either used no substantial portions of copyright material in my thesis or I have obtained permission to use copyright material; where permission has not been granted I have applied/will apply for a partial restriction of the digital copy of my thesis or dissertation.'

Signed ..........................................

Date .................. 21/ Oct. /2013 ..........................................

AUTHENTICITY STATEMENT

'I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis. No emendation of content has occurred and if there are any minor variations in formatting, they are the result of the conversion to digital format.'

Signed ..........................................

Date .................. 21/ Oct. /2013 ..........................................
I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project’s design and conception or in style, presentation and linguistic expression is acknowledged.

(Signed)_____________________________

(Date)_____________________________
COPYRIGHT STATEMENT

“I hereby grant the University of New South Wales or its agents the right to archive and to make available my thesis or dissertation in whole or part in the University libraries in all forms of media, now or here after known, subject to the provisions of the Copyright Act 1968. I retain all proprietary rights, such as patent rights. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation. I also authorize University Microfilms to use the 350 word abstract of my thesis in Dissertation Abstract International (this is applicable to doctoral theses only). I have either used no substantial portions of copyright material in my thesis or I have obtained permission to use copyright material; where permission has not been granted I have applied/will apply for a partial restriction of the digital copy of my thesis or dissertation.”

Signed: ................................

Date: ................................

AUTHENTICITY STATEMENT

“I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis. No emendation of content has occurred and if there are any minor variations in formatting, they are the result of the conversion to digital format.”

Signed: ................................

Date: ................................
Acknowledgements

When I first started as a master of engineering student at the School of Computer Science and Engineering in the University of New South Wales (UNSW), I was hoping to get the real taste of research. Looking back the past few years, I certainly feel that I have achieved what I came for and much more. There are people who helped me along the way to achieve these goals, people who I have built a strong relationship over the years both professionally and personally, and I would like to thank them.

First and foremost, I would like to thank my supervisor, Dr. Hye-Young Paik for being a great teacher, supporter and colleague during all stages of this research. I appreciate all her hard work and commitment to provide me with everything I needed to complete this study. Without her support, this work would not have been possible.

I would like to thank Dr. Ingo Weber for his valuable advices and active involvement in this work. His insightful reviews and expertises in the field have nourished this research at every stage. I would also like to thank Dr. Dac Dat Hoang for his friendly supports. We have had some great time together during coffee breaks and he has shared with me his past experiences as a research student which made my life at UNSW much easier. In addition, I want to express my gratitude to Smart Service CRC for offering me the opportunity to work on this project.

Last but not least, I would like to thank my parents who have always been my source of inspiration and motivation. Their sacrifices and encouragements have made who I am today. Thank you for being there for me through the good and the bad times. I would not have made this far without your support and love.
Abstract

Business process is a collection of tasks performed to achieve a certain business objective within an organisation. Due to the complexity and variability of business processes, organisations use information technology to support their business processes. Business Process Management Systems (BPMS) are widely accepted software systems that enable organisations to automate and continuously improve business processes. Although the benefits of BPMS have been widely recognised, there is still a large portion of business processes that are not adequately addressed by these systems. These types of processes make up the so-called the Long Tail of processes, i.e., highly customised processes that are unique to individual organisations or a small number of workers. Many organisations do not see significant Return On Investment on customising BPMS, or implementing solutions to automate Long Tail processes. So they often rely on using paper-based forms for running their processes due to its low cost, flexibility, and easy-of-use. However, using paper-based forms puts a burden of doing tedious manual work to the end users, restraining them from focusing on more important tasks.

In this thesis, we propose a pragmatic approach for automating form-based processes, in which annotation is used to help collect necessary information from the users to ascertain basic data flows and process execution paths. We have implemented the approach in the system called EzyForms, a framework for supporting form-based processes. The architecture supports an end-to-end life-cycle of forms, starting from its creation, annotation, and ultimately to its execution in a process. The proposed approach and framework can help administration officers with little or no technical background to automate the Long Tail of the processes which in turn helps the end users to execute the process in a more efficient manner.
List of Publications


## Contents

1 Introduction  
   1.1 Illustrating Scenarios ........................................... 2  
   1.2 Form-based Process ............................................. 4  
   1.3 Contribution ..................................................... 6  
   1.4 Thesis Organisation ............................................. 9  

2 Preliminaries  
   2.1 Long Tail of the Business Processes ............................. 10  
   2.2 Web Services ..................................................... 13  
      2.2.1 XML ............................................................ 13  
      2.2.2 Web Services Description Language (WSDL) ................. 14  
      2.2.3 Simple Object Access Protocol (SOAP) ......................... 14  
   2.3 FormSys Forms Manager ........................................ 15  

3 Literature Review  
   3.1 Supporting Ad-hoc Business Processes ......................... 19  
      3.1.1 Mashup ....................................................... 20
8.3 Directions for Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . 80
List of Figures

1.1 A form-based process expressed in BPMN for submitting new applications ........................................... 4

2.1 Product demand curve [15] ......................................................... 11

2.2 Long Tail of process automation [52] ................................. 12

2.3 WSDL Specification .......................................................... 15

2.4 a SOAP 1.1 request to search for “shrdlu winograd maclisp teletype” 16

2.5 FormSys Forms Manager Overview [63] .......................... 17

3.1 Screen capture of FormSys Process Designer [64] .............. 22

4.1 A life-cycle of forms in EzyForms and the schema for form annotations 33

4.2 Adding annotations to a form ............................................. 38

4.3 Example relationships between forms, process, annotation, and execution (fields omitted for clarity). 40

5.1 The position of texts relative to the text fields of personal details category ............................................ 46
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>Mapping input fields between two different forms using mapping recommendation</td>
<td>53</td>
</tr>
<tr>
<td>6.1</td>
<td><em>EzyForms</em> Architecture</td>
<td>54</td>
</tr>
<tr>
<td>6.2</td>
<td>Possible mappings in the input fields: two forms and an email template</td>
<td>57</td>
</tr>
<tr>
<td>6.3</td>
<td>Process executed by the end users</td>
<td>58</td>
</tr>
<tr>
<td>6.4</td>
<td>Example data flow showing how email input data is passed to fill in a form via Web service invocation at process execution stage</td>
<td>59</td>
</tr>
<tr>
<td>6.5</td>
<td>Process execution summary showing a list of approvers, forms associated with them, and email preview of the selected form</td>
<td>61</td>
</tr>
<tr>
<td>6.6</td>
<td>Email Dispatcher component in detail</td>
<td>62</td>
</tr>
<tr>
<td>7.1</td>
<td>Scores from the questionnaires on form owner’s tasks - rated on a 5-point Likert scale (1 being worst, 5 being best)</td>
<td>74</td>
</tr>
<tr>
<td>7.2</td>
<td>Form annotation user study evaluation</td>
<td>75</td>
</tr>
</tbody>
</table>
List of Tables

1.1 Summary of forms used for CSE travel request ........................ 3

3.1 Comparison of approaches to form-based process automation .... 25

5.1 Different categories of input fields: Personal Details vs. Process Specific 44

5.2 Types of reused personal details Input fields ........................... 44

5.3 Overview of function findTagRecommendation .......................... 47

5.4 Weight values for calculating MS .......................................... 49

5.5 Overview of function findFormDescriptionTags .......................... 50

5.6 Algorithm for input field mapping recommendation ....................... 52

6.1 Applicant Type ................................................................. 63

6.2 Approver Type ............................................................... 67

6.3 Model Type ................................................................. 67

6.4 Node Type ................................................................. 68

6.5 Joint Type ................................................................. 68

6.6 Different states of nodes ................................................. 68
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Evaluation results for our tag recommendation strategy for personal details input fields</td>
<td>70</td>
</tr>
<tr>
<td>7.2</td>
<td>Types of reused personal details input fields</td>
<td>75</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Business process is a collection of tasks performed to achieve a certain business objective within an organisation. Due to the complexity and variability of business processes, organisations use information technology to support their business processes. Business Process Management Systems (BPMS) are widely accepted software systems [65] that enable organisations to automate and continuously improve business processes in order to achieve better performance.

While all business processes work collectively to support business operations, BPMS solutions typically aim to support uniform and repetitive processes that will have significant impact on the performance of an organisation when automated [51], or processes that require constant monitoring and optimisation to be able to adjust to rapidly changing circumstances [29].

Although the benefits of BPMS have been widely recognised [26, 27], there is still a large portion of business processes that are not adequately addressed by these systems. These types of processes make up the so-called the Long Tail of processes [48], i.e., highly customised processes that are unique to individual organisations [56] or a small number of workers.
Many organisations do not see significant ROI (Return On Investment) on customising BPMS, or implementing solutions to automate Long Tail processes [37, 36]. Furthermore, a full suite of BPMS is too technical and sophisticated for people with little or no technical background to use to build an automated solution of their own.

Due to the above reasons, many organisations often rely on using paper-based forms for running their processes. Forms are the main artefacts of the Long Tail processes that enable cost-effective and simple ways of running and managing business processes. However, the use of the forms requires the end users like employees or students to do more manual work such as downloading the forms, typing in same information repeatedly, and getting approvals from several people from different organisation units. Moreover, the end users are often not familiar with the process and this costs them a lot of time and energy just to figure out which forms to fill in. The following real-world scenario outlines some of the drawbacks of using a form-based process.

### 1.1 Illustrating Scenarios

Long Tail processes tend to be implemented as a series of manual tasks (e.g., filling in the forms, emailing for approval requests). Let us take an example. Academics and research students in the School of Computer Science and Engineering (CSE) at UNSW must complete a travel request process for work-related travels (e.g., attending a conference).

The procedures and policies are described in a web page\(^1\). There are many different forms involved in the process as shown in Table 1.1. Depending on the employment status or position held in the school, people need to choose a different set

Table 1.1: Summary of forms used for CSE travel request

<table>
<thead>
<tr>
<th>Form Name</th>
<th>Available From</th>
<th>Type</th>
<th>People Involved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Travel Notification form</td>
<td>CSE Finance Unit</td>
<td>pdf</td>
<td>Traveler, Supervisor, HoS</td>
</tr>
<tr>
<td>Statement of authorship</td>
<td>UNSW Grants Unit</td>
<td>rtf</td>
<td>Traveler, HoS, Authors</td>
</tr>
<tr>
<td>Teaching arrangement</td>
<td>CSE Finance Unit</td>
<td>pdf</td>
<td>Traveler, Assoc HoS, HoS</td>
</tr>
<tr>
<td>Approval for use of vehicle</td>
<td>UNSW Finance Unit</td>
<td>pdf</td>
<td>Traveler</td>
</tr>
<tr>
<td>Travel diary</td>
<td>CSE Finance Unit</td>
<td>pdf</td>
<td>Traveler, HoS</td>
</tr>
<tr>
<td>Reimbursement claim</td>
<td>CSE Finance Unit</td>
<td>pdf</td>
<td>Traveler</td>
</tr>
<tr>
<td>Petty cash reimbursement</td>
<td>CSE Finance Unit</td>
<td>pdf</td>
<td>Traveler, Supervisor, HoS</td>
</tr>
<tr>
<td>Vehicle mileage claim</td>
<td>CSE Finance Unit</td>
<td>pdf</td>
<td>Traveler, Supervisor, HoS</td>
</tr>
</tbody>
</table>

of forms (e.g., students are not required to fill in the Teaching Arrangement Form). Furthermore, those forms are available from and managed by different business organisations in within the university (e.g., some available in the school Web page, some in UNSW finance department).

In the current scenario, a traveller is expected to read a web page for instructions and identify and download relevant forms, fill them out and submit for approvals.

Through this example, we observe typical drawbacks of Long Tail processes. Not only it takes a fair amount of effort to understand what is required, but also takes time to find necessary forms, print, fill out multiple forms and submit. In addition, these forms contain multiple fields that are duplicate across them, such as a title, name and address, which have to be repeatedly entered by the end users. And since these are manual tasks, the whole process cannot be automatically re-played later.

Given this situation, it would certainly be of value to provide a coherent set of tools to support the automation of such processes. To this end, we have to analyse...
form-based processes to identify informations that are associated with forms. Then we have to formalise necessary steps that are involved in automating the form-based process. We also have to develop an intuitive approach that can be used by the form owners who we expect to be non-technical. Finally, when the processes are modelled and deployed, it needs be executable by the end users.

1.2 Form-based Process

The aim of this thesis is to propose a framework that supports the whole life-cycle of form-based processes from form management to process execution. In this section, we describe the type of form-based processes we focus on in the thesis by articulating the activities and typical characteristics of the processes.

![Form-based Process Diagram](image-url)

Figure 1.1: A form-based process expressed in BPMN for submitting new applications

Figure 1.1 depicts the type of form-based processes we aim to automate, which is generalised from many form-based processes that we observed. Although there are other types that do not conform to the shown model (e.g., a process involving more than one user), it is our observation that typical form-based processes exhibit the following characteristics. Our current work focuses on these initially.

First, a form-based process consists of one or more paper-based forms which are...
eventually to be submitted to an administration unit to trigger or record an organisational process (e.g., a trip request). Second, a form-based process is initiated and executed by a single user (e.g., a trip requestor), and the user is normally responsible for finding information about the process (e.g., reading instructions on a web page, searching/downloading forms). Third, a form-based process involves obtaining zero or more approvals on completed forms, which could mean the user having to write multiple email requests and coordinating the chain of approval manually.

To elaborate on Fig. 1.1 further, in a form-based process, the user goes through the following activities:

**Process Identification.** One of the problems associated with form-based processes is that the user needs to figure out how to conduct the process. With form-based processes, this is often done by finding a web page that describes the process. This can be a time consuming process if there are overwhelming amount of information you have to read in understanding, and especially if you are going through the process for the first time.

**Form Identification.** The form-based process may consist of more than one form, and each form can be associated with a certain condition(s) that only the end user who meets the condition is required to fill in. For example, if the user is applying for a travel, teaching arrange form is required if s/he is a academic staff member, but the statement of authorship form is mandatory for every travel request.

**Download Forms.** When the end users need to download the forms, it happens that the form documents used in a same process exists in a different websites. This is due to the fact that form documents are not centrally managed and the owner of the forms may belong to different departments or organisations.
**Form Fill-in.** For the processes that consists of more than one form document, it is likely that the end user is required to enter same information repeatedly. This is due to the fact that the forms are not designed to be used in a same process initially, or they require different approvers from different departments, so it is required to repeat the same information such as your name, address, phone number, and so on in each form.

**Write Emails.** Once the form documents have been completed, they need to be submitted for approvals. This process includes writing number of approval request emails to designated approvers. In order to request an approval, the filled-in form must be attached to the email before they are dispatched to approvers. Often the users will print and submit the forms in person as well.

**Obtaining Approvals.** Obtaining a final approval may involve going through a chain of signatures.

### 1.3 Contribution

Based on our observation, we believe the key aspects of form-based process automation should include the following:

- centrally managed processes - it should be as easy as a single click to identify a required process and obtaining forms

- execution path routing based on end user’s condition - the relevant forms should be automatically identified and presented

- reuse of input data across different form documents - all repeated fields (e.g., personal details) should be reused and not asked to re-enter
Filled-in form dispatch for approval request - the forms should be automatically dispatched for signatures according to the approval requirements.

In the framework proposed in the thesis, we aim to take an ‘as-is’ situation of a form-based process and transform it into an automated process which has the properties outlined above. We realise this through simple techniques such as annotations and data mappings to implicitly model and execute a process.

For example, using our framework, we are able to transform the previously illustrated scenario as follows:

an admin officer\(^2\) at the CSE school uploads the eight forms into the system and matching Web services are created. The officer annotates the forms, adding details such as employment status conditions, approver’s name and email. The same officer (or another officer who is familiar with the travel request process) creates a process model which consists of the five forms, data flow mapping between forms and extra annotations if needed (e.g., email templates). The model is then deployed, ready for use by a school member\(^3\). Now the execution of the travel request process means that all forms are in one place, any optional forms are automatically identified via the conditions, no need for repetitive entering of the same data and automated dispatch of filled forms for approvals as needed.

The overall approach of the framework is implemented in a prototype system called \textit{EzyForms} which provides tools/environments for automating form-based processes which are traditionally executed manually. Some of the automation is made

\(^2\)We will refer to this class of users as ‘form owners’ in the rest of the thesis.

\(^3\)We will refer to this class of users as ‘end users’ in the rest of the thesis.
possible by engaging end users who are not necessarily technically savvy, in annotating the form services.

Our approach is to enable the form users to model and deploy their existing form-based processes into a service-enabled framework without requiring technical knowledge or the cost of re-engineering. The key concept of our proposal is in various types of annotations on forms and effective applications of such information during the modelling and execution phases of form-based processes. In particular, we:

- Explore a feasibility and applicability of an annotation technique as a way of automating the form-based processes.
  - identification of different types of annotation for simple, implicit modelling and execution of form-based processes,
  - perform analysis on the forms being used in real-world and use it to generate tag recommendations,
  - design and implementation of tag recommendation strategy; specifically, for input field and form description annotation,
  - evaluation of input field tag recommendation strategy.

- Develop a prototype named EzyForms, a framework that supports the whole life-cycle of form-based process management. The framework includes the following novel aspects:
  - smart applications of the annotations to support basic data flows and execution patterns in form-based processes,
  - a WYSIWYG-fashion integration of forms in every stage of the process life-cycle, from form service creation through to modelling and execution.
Evaluate such a framework in terms of applicability and usability through user studies.

1.4 Thesis Organisation

The rest of the thesis is structured as follows:

Chapter 2 provides an extensive overview of relevant technologies. We first describe Long Tail of the processes. Then, we describe Web services technology and previous research works that our current research continues to build on in more details.

Chapter 3 provides literature reviews to articulate the problem field and an extensive and critical overview of the state-of-the-art commercial products to which this work is relevant to.

Chapter 4 describes an approach for applying annotation technique to automate the form-based processes. We first analyse real-world form-based processes and identify annotation artefacts for the form documents. We then provide annotation model and illustrate end-to-end process execution.

Chapter 5 presents number of user supporting features that aims to help users in form annotation, process modelling and process execution.

Chapter 6 describes the architecture and implementation of the tool support, which has been developed for illustrating the feasibility and usability of the proposed approach for form-based process automation.

Chapter 7 concludes the thesis by summarising the findings and contributions of our work and the future directions of this research.
Chapter 2

Preliminaries

In this chapter, we present several concepts that are important and form the background knowledge in the thesis.

We first introduce a brief history of the Long Tail theory and how they are applied in the business process automation domain (Sect. 2.1). We also describe the characteristics of the Long Tail processes in more detail. In section 2.2, we introduce Web service technology and its building blocks as they are the core technology used for service-enabling form documents. In section 2.3, we present FormSys Forms Manager which is a Web-based system that provides a form repository and the capability to model and deploy new processes with the service-enabled forms.

2.1 Long Tail of the Business Processes

The phrase “Long Tail” was first coined by Chris Anderson in 2004 [15]. The Long Tail theory is that the demand for niche products with low sales volume that make up the Long Tail of an overall product demand curve (Fig. 2.1) can, in aggregate, exceed the demand for a relatively small number of best selling products.
In order to maximise the profit, traditional stores with a limited shelf space have been focusing on selling the mainstream products at the head of the demand curve due to the high costs of inventory and distribution that makes serving the niche products at the Long Tail unprofitable.

But with the power of Internet and Web-enabling technologies, many retailers begin to serve their customers digitally which significantly reduces the distribution and promotion costs, and with the real time information about buying trends, online retailer companies are now serving the niche product market profitably in addition to the blockbusters.

In recent years, many BPM researchers started looking at the Long Tail theory applied in the BPMS context. As shown in Fig. 2.2, the traditional BPMS solutions have typically been targeted towards supporting high demand processes that are roughly similar across organisations. On the right end of the spectrum, we see more ad-hoc, situational applications that focus on day to day problems of end users, and satisfy the needs of small number of individuals. There are not many solutions that
are designed for the Long Tail end, partly due to the fact that customising BPMS to support unique organisational processes are costly and many cases not feasible.

As a consequence, many organisation-specific processes remain low-tech (e.g., spreadsheets, forms) and heavily dependent on human manual activities such as emailing and faxing forms. And we can find this type of processes everywhere still. For example, when opening a new account at the bank, or when applying for a driver’s license, or applying for a reimbursement, and so on.

![Figure 2.2: Long Tail of process automation [52]](image)

In fact, applications in the Long Tail do not handle large scale data and have low demands on quality or proper governance. Rather, the emphasise is on solutions that are rapid, lightweight and flexible [36]. We believe enabling the users to build a solution themselves by providing a lightweight process automation framework that
has low technical barriers would help alleviate some of the challenges in the Long Tail processes.

2.2 Web Services

Web service can be defined as a software system designed to support interoperable machine-to-machine interaction over the Web. Using Web service technology, a service providers can expose their services to be accessible via the Web by registering the services on the service registry, and the service requesters can search and invoke the service, or even compose different services together to provide higher level functionality and value-added services [49]. Good example of Web service technology is Amazon EC2 Web service [5] which provides developers with various operations to access relational database service, load balancing service, Amazon EC2 service, and so on.

2.2.1 XML

The increased popularity of the Web service technology is largely due to its ability to interoperate between different systems running on a variety of platforms and/or frameworks. Although there are many ways to build and use Web services, XML plays a key role in Web service technology due to several key reasons.

- Nearly every language has support for XML and XML based standards (parsers).
- Because it is text based, XML is easy to read and understand.
- The major vendors who in the past have supported binary protocols such as DCOM, CORBA, or EJB/RMI have evolved to work with SOAP.
XML is extensible.

XML data can be transported using a variety of protocols (e.g., HTTP, FTP, SMTP, etc.) because XML only defines the document not the transport.

Web service is based on three XML open standards which are Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI).

2.2.2 Web Services Description Language (WSDL)

Web Services Description Language (WSDL) [16] is a language for describing Web services. The information described in WSDL includes the operations that make up the service and the messages that are exchanged between the service requester and provider. WSDL can be used to generate the proxy (client stub) to the Web service that acts as a access point between the service and the client. Any proprietary integration solutions can be used to implement the Web service application (e.g., CORBA, JMS, COM, COBOL) as long as both the service requester and provider agrees on the service description (WSDL file).

The WSDL can be divided into two parts as shown in Fig 2.3. The service themselves are described abstractly and then bound to a concrete network protocol and message encoding format. This allows Web service design to be separated from Web service deployment environment details.

2.2.3 Simple Object Access Protocol (SOAP)

The client invokes services via proxy sending and receiving SOAP [13] messages. SOAP describes how information should be packaged into an XML document. SOAP
message consists of header and body parts where application payload is defined in the
body part and an additional protocols such as security and transaction is defined in
the header part. Figure 2.4\(^1\) shows sample SOAP message for Google’s Web service
interface using SOAP 1.1.

### 2.3 FormSys Forms Manager

FormSys Forms Manager [63] is a Web-based system that service-enables form doc-
uments. FormSys Forms Manager manages PDF forms in the central FormSys
repository.

When a form is uploaded to the system, FormSys automatically generates two
matching Web services. The creation of Web service is based on WSDL/SOAP
standards and the procedure is completely automated and hidden to the users.

\(^{1}\)http://www.w3.org/2004/06/03-google-soap-wsdl.html
Figure 2.4: a SOAP 1.1 request to search for “shrdlu winograd maclisp teletype”

- **soap2pdf**: receives data from an application, fills a form with it, and returns a form via email or a URL where the filled form is available.

- **pdf2soap**: extracts the data from a filled form, assembles a corresponding SOAP message, which it sends to an application.

An active **soap2pdf** service is a full-fledged Web service offering a WSDL interface with a single operation, **fillForm**. The input to this operation is mainly the input data for a form document. After the service invocation, the filled form is provided at a URL or emailed back to a given email address. **soap2pdf** services can be flexibly used in (Web) applications or part of BPEL orchestrations.

**pdf2soap** implements a read form function. When a filled form is uploaded by an end user through the pdf2soap front-end component, the data is extracted and packaged in a SOAP message according to the mapping in the database, and sent to an the associated consuming service.

The architecture design of FormSys is shown in Fig. 2.5 and has the following
components: a manage-form service for uploading, administering, archiving, and deleting form templates; a management front-end which is a Web-based component used by form administrators; a format converter container, where converters from arbitrary formats to the central forms format (in the implementation: PDF) can be included as plug-ins; a form-to-data service (i.e., pdf2soap) handling uploaded filled forms (and possibly an Optical Character Recognition (OCR) pre-processor for using pdf2soap functionality on filled paper forms); a data-to-form service (i.e., soap2pdf) for instantiating and filling forms; a database for form templates, field mappings, and instance data; and a core component controlling the whole system including form-to-data and data-to-form functionality.

FormSys Forms Manager allows the users to manage single PDF forms. The work proposed in the thesis uses a set of APIs provided by FormSys Forms Manager...
to build a system that support creation and execution of form-based processes. In particular, our prototype relies on the Forms repository and soap2pdf service part of FormSys Forms Manager.
Chapter 3

Literature Review

In this chapter, we review literatures on relevant topics to our work. The first part covers automating flexible and ad-hoc processes. This includes mashups, the research work that was previously conducted by Service-Oriented Computing Research Group\(^1\) in CSE UNSW called FormSys Process Designer, and other commercial BPM tools that are available in today’s market. We also review the artifact-centric approach in process modelling. The second part covers literatures on annotations as annotation technique is our proposed approach to automate the form-based processes.

3.1 Supporting Ad-hoc Business Processes

Automating the form-based process is an effort to automate one of the frequently appearing ad-hoc process in our daily life, and in this section we look at other research and commercial work that has been conducted in this area.

\(^1\)http://soc.cse.unsw.edu.au/
3.1.1 Mashup

Mashup is a way of providing new service by combining data, presentation or functionality from different sources of services. Mashup has played an important role in the evolution of Web 2.0 in that mashup composition tools are usually simple and intuitive to be built by end users who do not have programming skills [6].

Service providers have exposed variety of services through Application Programming Interfaces (APIs), most notably Google Map API \(^2\) that is used to create mapping mashups and Flickr API \(^3\) that is used to create photo mashup applications. Other type of service exposure is data feeds such as RSS and ATOM. Many tools have been developed to handle mashup between various data sources such as Damia [8, 61], Exhibit [30] and MashMaker [21].

These tools aim to overcome the challenge in structural and semantics diversities of schema, or data manipulation such as conversion, filtering, format transformation and etc [39]. Although form-based processes can benefit from such data manipulations, the applicability of the mashup tools are subject to number of requirements such as availability of API to expose data from the form service. Moreover, such tools do not provide comprehensive set of solutions to model new process or to automate the approval processes.

3.1.2 FormSys Process Designer

FormSys Process Designer [64] is an extension to the FormSys Forms Manager with an intention to generate BPEL code via modelling processes using the form-filling Web services other SOAP-based Web services. FormSys Process Designer aims to

\(^2\)https://developers.google.com/maps/
\(^3\)http://www.flickr.com/services/api/
provide a user-friendly service composition environment by presenting services in a graphical representation format that the business domain experts feel comfortable, and providing automated verification technique to ensure the correct combination of the modelled control and data flow.

For example, Fig. 3.1(a) shows the icon the “Find News Data” service. The technical details about the service is described in a standardised WSDL which contains invocable WSDL operations. These operations have an input and optional output messages and they are represented as forms which reflects the services’s running user interface. Figure 3.1(b) shows an example of the input message as a form. The names of the form fields which corresponds to the service’s input and output parameters can be set to something meaningful to the users, and can be used during search and composition of services.

The coloured boxes in Fig. 3.1(b) corresponds to the data fields from the message and are used to enable automatic execution of designed processes, that is, data fields from one message is mapped to data fields of another message.

Although FormSys Process Designer provides simple and intuitive composition between distinct and diverse services, the data mapping is done entirely manually by the process modeller who knows the services being composed well enough. The form-based processes contains lots of duplicate data fields across multiple form documents, and hence the manual mapping is not efficient. Moreover, modelling end-to-end form-based process using FormSys Process Designer is not a trivial task given that this requires implementation of the extra services such as email sender and receiver, and careful composition of these services. Our thesis is a departure from the approach used in FormSys Process Designer. Our experience with this work showed that the experience that explicit process modelling (i.e., asking users to build sequence, if-else type constructs) is limited and impose higher entry barrier
for people with low technical skills [62]. One of the contributions in the thesis is that we use annotations implicitly construct simple process models.

3.1.3 Commercial Tools

Microsoft SharePoint [1] suite provides a rapid development environment for, among other things, custom design forms and workflows. Its tool for managing forms allows the users to create forms and apply conditional formatting, actions and validations. Its workflow design tool is able to support creation and execution of sophisticated approval workflows. Its focus is on the easy creation of the forms and embedding the form in any SharePoint website pages, while our proposed solution focuses on the restricted access to the forms input fields and binding meaningful context information to the form. This is due to the fact that form-based processes already have forms to work with (i.e. Acroform), and it is better to reuse this form as a presentation layer rather than having creating a new form from a scratch. However, it requires
deep understanding of the tool in order to be able to model the process. Liferay [2] suite (e.g., Social Office and Portal) focuses on the synchronisation of the work and communication between the team members. However, it has limited support for process automation. For example, its workflow engine Kaleo is used to control creation, approval and rejection by a person of one “asset” (e.g., a document), whereas a form-based process may involves multiple of such “asset”. Another downside is that the workflow developer needs to write a customised script in a programming language to support if there were actions outside of the standard one that need to be performed on your asset using supported languages such as Javascript, Ruby, or Python.

There are many other tools (e.g., Adobe LiveCycle ES2 [3], SAP Gravity [4]) in this category we can discuss, but the strong difference is that our focus is in supporting form-based processes in its “AS-IS” state (i.e., low cost, time and effort), which means we do not require the forms or user interfaces to be designed and created from scratch. We use the forms that are in situ and the users will always see them as the main interacting components for the processes. Our view is that this approach can provide an effective solution for processes that are 'in transition' to full automated systems.

To highlight salient features of our system, we first compare EzyForms with existing commercial products with regard to their characteristics and fitness for automating form-based processes. We take two most well-known commercial tools in this area which are Microsoft(MS) SharePoint and Adobe LiveCycle, and summarised our findings in Table 3.1.

MS SharePoint is designed to be a collaboration tool for enterprises. It can dynamically interact with internal users using customised forms which are created using MS InfoPath. However, when dealing with existing paper-based forms such as
MS Office documents, its capability is limited to storage, content management and routing of the forms. The forms are persisted in a database on which a workflow associated with the form is triggered under certain conditions. A workflow can be created in MS Visio using process specific diagram notations, or in MS SharePoint using a textual syntax. Either way, the process is explicitly designed, which is different from our implicit approach. MS Sharepoint also lacks in a functionality to reuse input data across different forms in the same process.

Adobe LiveCycle is an enterprise solution for business process management and document services. It supports PDF, SWF, and HTML forms both inside and outside the organisation. It can assemble multiple PDF forms into a form package for end-users' convenience, and employs data extraction and pre-population. However, in order to automate the process with existing forms, a non-trivial amount of effort is required to develop a component that can transform data from one form to the other.

Commercial enterprise suite solutions meets diverse needs of users. However, they impose management overheads such as installation requirements of of many programs, integration with legacy system, staff training, etc., to organisations and form users. Therefore the cost and risk of adopting such systems are not preferable from the organisations’ perspective unless they have strong motivations and visible ROI to do so. EzyForms on the other hand, is not as robust as commercial tools (being only a research prototype), but provides light-weight and cost-effective solution to a specific problem that is prevalent in today’s business environment. The main point of difference between EzyForms and other commercial tools is that the interest of EzyForms lies in enhancing the end user experience in form-based processes with minimal risk and cost, and not necessarily driven by the ROI.
Table 3.1: Comparison of approaches to form-based process automation

<table>
<thead>
<tr>
<th></th>
<th>MS SharePoint</th>
<th>Adobe LiveCycle</th>
<th>EzyForms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of form</td>
<td>MS InfoPath,</td>
<td>PDF,</td>
<td>AcroForm**</td>
</tr>
<tr>
<td></td>
<td>MS Office</td>
<td>SWF, HTML</td>
<td></td>
</tr>
<tr>
<td>Document processing</td>
<td>Persistent</td>
<td>Persistent</td>
<td>Transient</td>
</tr>
<tr>
<td>Modelling approach</td>
<td>Process flow design</td>
<td>Process flow design</td>
<td>Annotation, Data mapping</td>
</tr>
<tr>
<td>Input data storage</td>
<td>Server-side</td>
<td>Client-side</td>
<td>Client-side</td>
</tr>
<tr>
<td>Target users</td>
<td>Enterprise</td>
<td>Enterprise,</td>
<td>Enterprise, Public</td>
</tr>
<tr>
<td></td>
<td>Public</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>Integration with Legacy System</td>
<td>Required</td>
<td>Required</td>
<td>Not necessary</td>
</tr>
<tr>
<td>Open Source</td>
<td>No</td>
<td>No</td>
<td>Possibility</td>
</tr>
</tbody>
</table>

**: Although we currently support AcroForm only, we believe, conceptually, auto-generating a matching Web service from a form to read/write data is a generic idea that can be applied to a wide range of form formats.
3.1.4 Other Forms Related Work

[10] developed a formal model that associates meaningful segments of documents with process activities which allows tight control over document related processes as well as customisation of document monitoring and content based interaction between WFMS and other applications. In [45], the authors propose active documents [7, 19] as a method for automating business processes based on form documents. Their research derives new concepts of active documents by examining meanings implied in paper form documents and explicitly express them in a knowledge representation language in which machines can understand and process. While their work enhances document processing with regard to the process execution, they require form designer tool to generate new forms which is stored as XML document, and the workflow engine to processes them.

3.1.5 Artifact-Centric Process Modelling

An artifact is defined as a key business-relevant information that is concrete, identifiable, and self-describing [46]. The artifact-centric process model consists of such business artifacts, the collection of their life-cycle and the interaction between them. The construct of the model can vary to suit the business needs. For example, the data model might be constructed as name value pair [46] or by providing pre- and post-conditions [12, 18]. The artifact-centered approach allows business processes to be more flexibly represented, easier to make changes and manage applications. The focus of the artifact-centric approach is to enable the organisations to adapt to the new requirement and expand in linear scale. With form-based processes, many of the benefits of artifact-centric approach can be incorporated in process model. In this thesis, our focus is on providing a programatic approach to automating the
existing process using existing form document.

Our contribution is on automating the ‘as-is’ process with minimal impact on the existing processes than reinventing or enhancing the process. This is because we believe a solution based on this ‘as-is’ approach (i) can help lower the barrier of acceptance from the users and (ii) can minimise the need for also changing other processes that rely on the outcome of the existing process.

3.2 Annotation

The second topic is annotation. Annotation is our approach to automating the form-based processes. In this section, we review literatures on different aspects of annotation technique including motivation of uses, recommendation approaches, and tools for annotations.

3.2.1 Web Service and Business Process Annotation

Our main idea for EzyForms is that, it will become an environment where the life-cycle of the system works to ensure the knowledge contributions from the users are reflected in the continual improvement of the system. This starts with supporting easy-to-use annotation mechanism. Hence, we see the research work in Web service annotation as an important and relevant work we can learn from and contribute to. METEOR-S framework [50, 53] is a Web service annotation framework which takes existing Web service technologies and semi-automatically marking up service descriptions with ontologies (e.g., expressed in RDF [38]) using schema matching techniques that is based on linguistic and structural similarity. [35] presents practical method for semantically annotating collections of XML Schemas and Web service
interfaces which adopts incremental methodology for building ontology. Many research work in this category of annotation assumes an existing (formal) ontology which may not be feasible in many Long Tail process scenarios. In [17], the authors proposed an approach to embrace user-centric aspect to enhance Web service discovery by providing collaborative tagging-based environment to end users. In this approach, Web services are tagged with different keywords from different users and the tag weight which refers to the occurrence of a specific tag associated to a Web service is used to rank Web services found after the discovery process. The role of annotation in semantic Web service is similar to the role of annotation in form-based processes in that they are both used for mapping of data. They are complementary to our work in that one possible direction our work is to use ontology to improve the recommendation and data mapping accuracy.

The focus of Web service is B2B/EAI. Most of research addresses integration and search in this context. However Long Tail processes are B2C. Annotation in B2B has problems such as i) the user has to browse available ontologies to find the relevant ontologies, and ii) the size of the Web service description and the size of the ontology or vocabulary (METEOR-S addresses this problem). Automatically generated WSDL elements may not contain a valid name but some randomly generated alphanumerics. So METEOR-S is not applicable in this case.

### 3.2.2 Online Resource Annotation

Annotation of resources and its tooling support has been an active field of research.
Motivation for Annotation

A recent study by Ames et al. [9] on the motivation for annotation in Flickr and the role of tag suggestion in the system has provided number of implications for the design of annotation system in general such as making the annotation pervasive and multi-functional, and not forcing the users to annotate. Their work also reveals that easy annotation features and relevant tag suggestions encouraged the tagging and gave users direction as to the sort of tags they should use. Marlow et al. [42] provides some insights for design decisions in architecting new tagging systems. They point out the importance of studying the incentives for driving participation, and the level of system support to embrace or limit these motivations. The incentives for annotation of online contents such as photos are a well studied area [9, 47]. [9] suggests that the biggest motivation of tagging for most of the users were for general public (photo pools, search, self-promotion). Self-organisation (adding tags for later retrieval) [58, 22], self-communication (memory and context) [54] and social communication (adding context for friends, family, and the public) [41, 32] were also common motivations for online photo tagging.

This motivation is hardly applicable to form annotators since the social impacts of form annotation is much smaller than sharing photos or bookmarks on websites such as Flickr or Delicious, where millions of users upload and share new contents daily. However, we expect that form annotation can contribute greatly to semi-automating the process modelling in public or private organisations at a low cost, in addition to the form organisation and retrieval benefits.
Tools for Annotation in HCI

One of the active field of research in human-computer interaction is on providing tools for annotation of media [23, 34, 59]. These research focuses on maximising the benefits of annotating resources (e.g., photo retrieval) [31], but most people still do not bother with annotation of their photos [54]. This indicates that the potential benefits of annotation do not overcome the investment [33] even with the most advance annotation systems. The research effort is mainly focused on desktop-based applications whereas our intended system is web-based which is expected to be much more cost-effective and scalable.

3.2.3 Tag Recommendation

In some cases, the tag suggestions can inspire the users to tag their photos and give them guidance for how best to annotate, it can encourage tagging and give users ideas about possible tags [9].

Type of Strategies

There are number of strategies for automatically annotating online resources. In [11], the authors provide automatic image annotation system that uses statistical modelling and optimisation methods to establish probabilistic association between images and words. A large collection of example images are used to educate the system. In [60], the authors analyses the tagging behaviour and the kind of tags the Flickr users provide to the 52 million publicly available photos. Based on the analysis, the authors present tag recommendation process that is based on the tag co-occurrence and tag aggregation and ranking. Tag co-occurrence is used to derive a list of candidate tags from the user-defined tags. Large quantity of Flickr user-
generated data is used to support this strategy to work reliably. Building ontologies and term hierarchies are common approach for smart tag recommendation [44, 55]. However, in our work, we assume that we have preloaded tag library and focus on selecting accurate tag recommendations from the candidate tags.

3.2.4 Collaborative Tagging System

Collaborative tagging is most widely adopted by contemporary Websites as a way of classifying large amount of content (URL, photo, etc.). In [42], the authors provide key dimensions in tagging systems’ design taxonomy and possible implications. Some of the key dimensions includes Tagging Rights, Tagging Support and Aggregation Model. According to the taxonomy defined in [42], EzyForms is a system that is self-tagging where users can only tag their own resources, viewable and suggestive-tagging that allows users to view the tags already attached to a resource and recommends possible tags to the user, and Set-Model aggregation which avoids multiplicity of a tag around the same resource. Their study on Flickr in comparison with the work of Golder and Huberman [25] provided an initial evidence that the system’s place within this dimensions, the incentives driving participation, and the extent to which the system supports or limits these motivations can vary the dynamics of the system such as the relationship between the tags usage and other types of input. After thoroughly analysing [42, 25, 57, 43], we believe that collaborative tagging systems can collect domain knowledge of the form owners to enhance the metadata of the form documents and hence facilitate the modelling of form-based processes.
3.2.5 Data Mapping

Data mapping is a critical component in automating the form-based business processes. Similar challenges also arise in deep-web community where mediating related data sources of the same domains is becoming essential. This normally involves mapping HTML form fields to the underlying data sources. In [28], the authors propose a framework that finds a consistent model amongst a set of input schemas. Their approach takes holistic matching of schemas associated with deep web query interfaces. Their research focuses on synonym discovery which is closely related with the input field mapping between the forms. However in statistical matching of input schemas, each schema contains a unique set of attributes whereas in our approach for the form input field mapping, we support several synonyms to be added to each input field and use them as an attribute of the input field for mapping. In [40], the authors present an approach to predict annotation label for data units so that they are machine processable. Their approach involves aligning the data units into different semantic groups and annotating it from different aspects and aggregating the different annotation to predict the final annotation label. The data alignment utilizes several feature which are not necessarily applicable in form input field mapping. For example, Presentation Style, Tag Path, Adjacency between the units all relies on the structure of the form document which is not available in every types of forms (e.g., PDF form does not expose format structure information to third parties). However, this approach would be interesting to explore further if the process that needs to be automated consists of structured forms only. In [20], the authors present a set of common sense design rules that emerges from heterogeneous query interfaces. Although these rules are derived from the observation of HTML web forms, they can be a complementary for deriving more accurate input field tag recommendations for various other types of forms such as the PDF forms.
Chapter 4

Forms and Annotations

Our approach aims to enable form owners with little or no technical background to automate the form-based processes, and allows end users to execute the process in more efficient manner. Our proposed approach consists of four steps: form upload, form annotation, process modelling, and process execution.

Figure 4.1: A life-cycle of forms in EzyForms and the schema for form annotations

Figure 4.1 depicts the life-cycle of forms in EzyForms, and the association between the form and the metadata that EzyForms uses to automate the form-based processes. The following sections describes each stage of the form life-cycle in more details.
4.1 Form Upload

In order to fill-in the forms electronically, we convert a form into a Web service. This is accomplished by the previous work FormSys [63], through which PDF forms\(^1\) are uploaded to the central FormSys repository. All forms in the repository are available to other form owners to design and deploy new form-based processes. When a form is uploaded to the system, FormSys automatically generates two matching Web Services:

- **soap2pdf**: receives data from an application, fills a form with it, and returns a form via email or a URL where the filled-in form is available.
- **pdf2soap**: extracts the data from a filled-in form, assembles a corresponding SOAP message, which it sends to an application.

The creation of Web Service is based on WSDL/SOAP standards and the procedure is completely automated and hidden to the users. The rest of the life-cycle of forms in *EzyForms* extends these basic functions of FormSys.

Formally, we define \( \mathcal{F} := \{ F_1, F_2, \ldots \} \) as the set of all forms in the system, where each form has a set of fields \( \mathcal{G}(F_i) := \{ f_{i_1}, f_{i_2}, \ldots, f_{i_n} \} \).

4.2 Form Annotation

We recognise the fact that, to remove the need of BPM/IT professionals’ involvement, the system must ascertain necessary information by itself as much as possible (e.g., which form is relevant for process X, which fields in form A are duplicated in

---

\(^1\)To be precise, we use AcroForm, a sub-standard of PDF which contains editable and interactive PDF form elements.
form B). This is, of course, not always feasible or accurate. To bridge the gap, we introduce the form annotation step.

The form annotation is an act of adding metadata to the form. Adding metadata such as keywords or tags is an effective way of categorising a large pool of resource for both personal and public purposes [9, 14]. Due to its informality and flexibility in use, annotation has been widely accepted by users for assisting Web browsing or searching, and utilised in many systems like Flickr, Delicious, and Youtube [14, 24].

There are two types of annotations, each assisting different aspects of process automation: annotation for process modelling and for process execution. Also, note that the annotation is defined on two levels: the form-library level (i.e., the annotation is defined on the form generically, without regards to the processes in which the form is involved) and the process level (starting from the form-library level, the annotation can be refined within the context of a process – see Sect. 4.3).

### 4.2.1 Annotation for Modelling

This annotation is used to help form owners when modelling a new process.

- **Form description tag**: these are descriptive or representative tags that can describe the form. For example, the travel reimbursement form in our usage scenario may have form descriptions tags like travel, reimbursement, travelcost.

We formalize the system-wide set of form description tags as $\mathcal{T} := \{t_1, t_2, \ldots\}$, and the annotation function $T : \mathcal{F} \mapsto 2^\mathcal{T}$ as a mapping from the forms to the description tags which apply to this form: $T(F_i) = \emptyset$ or $T(F_i) = \{t_{i_1}, \ldots, t_{i_k}\}$ for $F_i \in \mathcal{F}$ and $t_{i_1}, \ldots, t_{i_k} \in \mathcal{T}$ where $k$ corresponds to the number of form description tags added to the form.
These tags contribute to the search and discovery of forms. Although not currently implemented, form description tags can also be used to prefetch or recommend forms during the process modelling phase.

- **Input field tag:** these are synonyms, alias or any descriptive tags for an input field in a form. For example, name field in the travel reimbursement form may have tags such as `name`, `staffname`, `fullname`.

  Formally, we write $\mathcal{I} := \{i_1, i_2, \ldots\}$ for the set of input field tags available in the system. The respective annotation function $I : \mathcal{G} \mapsto 2^\mathcal{I}$ is defined as a mapping from the form fields to the field tags: $I(f_j) = \emptyset$ or $I(f_j) = \{i_{j_1}, \ldots, i_{j_k}\}$ for $f_j \in \mathcal{G}$ and $i_{j_1}, \ldots, i_{j_k} \in \mathcal{I}$ where $k$ corresponds to the number of tags added to the input field.

  These tags contribute to ascertain simple data flow between forms. That is, by comparing the tags associated with input fields from each form, as well as their respective text labels, we can postulate if any given two input fields share the same input data.

### 4.2.2 Annotation for Execution

- **Condition:** This type of annotation specifies conditions under which the form should be used. They are used to determine if the form should be filled, in the context of the running process instance.

  We define the system-wide set of conditions as $\mathcal{C} := \{c_1, c_2, \ldots\}$, and the condition annotation function $C : \mathcal{F} \mapsto 2^\mathcal{C}$ as a mapping from the forms to the conditions which apply to this form: $C(F_i) = \emptyset$ or $C(F_i) = \{c_{i_1}, \ldots, c_{i_k}\}$ for $F_i \in \mathcal{F}$ and $c_{i_1}, \ldots, c_{i_k} \in \mathcal{C}$ where $k$ corresponds to the number of conditions associated with the form.
The conditions on a form are a template for conditions in a process. Process-
level conditions determine if the form should be filled by a particular end user
at process execution stage (e.g., if a form should be filled in if the user is a
student). Details on all execution aspects are given in Section 4.4.

- **Approver**: This annotation type describes the names and email addresses of
  people who are responsible for approving some form (e.g., travel requests may
  need approval from the Head of School), and used when dispatching approval
  request emails.

  Formally, we write $\mathcal{A} := \{a_1, a_2, \ldots\}$ for the set of approvers stored in the
  system, and $A : \mathcal{F} \mapsto 2^\mathcal{A}$ is a function mapping from the forms to the approvers
  which apply to this form: $A(F_i) = \emptyset$ or $A(F_i) = \{a_{i_1}, \ldots, a_{i_k}\}$ for $F_i \in \mathcal{F}$ and
  $a_{i_1}, \ldots, a_{i_k} \in \mathcal{A}$ where $k$ corresponds to the number of approvers associated
  with the form.

- **Email Template**: This annotation type specifies email templates for creating
  email content to be sent to approvers, where the filled-in form gets attached.

  The email templates available in the system are formally referred to as
  $\mathcal{E} := \{e_1, e_2, \ldots\}$, and the email annotation function as $E : \mathcal{F} \times A \mapsto E$, a mapping
  from the forms and their respective approvers to the email template which
  should be sent to this approver for this form: $E(F_i, a_j) = e_k$ iff $a_j \in A(F_i)$,
  else undefined.

Figure 4.2 shows the main user interface for adding annotations to a form. Note
that the collected annotations on different forms by different users are in fact cen-
trally managed and shared via *EzyForms* Knowledge Base (KB), so that they are
visible to different form owners and can be reused. We use one of the common
methods known in tag aggregation called ‘Set-Model’ (as opposed to ‘Bag-Model’).
which manages the collective tags in a set to ensure all tags are unique within the KB [42]. It is also noted that adding annotations is an activity separate from process modelling tasks and it is possible that annotation and modelling are done by different people.

### 4.3 Process Model

The process model is designed based on the following characteristics of form-based processes:

- they are purely form-to-form processes, that is, it is possible to describe the processes as multiple steps of fill-form activities
- they are a single sequential flow where conditions are used to determine op-
tional part of the flow (i.e., which form is relevant for the current user).

In this section, we describe the formal model for the association between form annotation and the process model.

### 4.3.1 Process Definition

The annotations associated with the form documents in a process are translated into the process model. A *process model* is defined as a 6-tuple \( p := (F|_p, C|_p, A|_p, E|_p, I|_p, O) \), such that:

- \( F|_p \subseteq F \) is a projection from the set of all forms to its subset used in \( p \).
- \( C|_p : F|_p \mapsto 2^C \) is a function mapping from the forms in \( p \) to the conditions which apply to this form: \( C|_p(F_i) = \emptyset \) or \( C|_p(F_i) = \{c_{i_1}, \ldots, c_{i_k}\} \) for \( F_i \in F|_p \) and \( c_{i_1}, \ldots, c_{i_k} \in C \).
- \( A|_p : F|_p \mapsto 2^A \) is a function mapping from the forms in \( p \) to the approvers which apply to this form: \( A|_p(F_i) = \emptyset \) or \( A|_p(F_i) = \{a_{i_1}, \ldots, a_{i_k}\} \) for \( F_i \in F|_p \) and \( a_{i_1}, \ldots, a_{i_k} \in A \).
- \( E|_p : F|_p \times A \mapsto E \) is a function mapping from the forms in \( p \) and their respective approvers to the email template which should be sent to this approver for this form: \( E|_p(F_i, a_j) = e_k \) iff \( a_j \in A(F_i) \), else undefined.
- \( I|_p : G(F_k) \mapsto \{I, \bot\} \) is defined as a mapping from a form field to zero or one field tag: \( I|_p(f_j) = \bot \) (no field tag) or \( I|_p(f_j) = i_j \), where \( f_j \in G(F_k) \), the form belongs to \( p \): \( F_k \in F|_p \), and \( i_j \in I \).
- \( O \) specifies the order of the forms in \( p \), and thus is an ordered permutation of \( F|_p : O = (F_{i_1}, \ldots, F_{i_k}) \) where \( k \) corresponds to the number of elements in
\( \mathcal{F}^p \), and \( i_j \neq i_l \) for any \( j, l \in \{1, \ldots, k\} \).

Almost all process-specific annotations are projections of their respective forms-library level counterparts. The exception are the field tags: where on the form library level sets of field tags can be annotated, on the process-specific level at most one field tag can be assigned to each field. Note that we do not require the annotations on the process level to be subsets of the library level.

Figure 4.3: Example relationships between forms, process, annotation, and execution (fields omitted for clarity).

Figure 4.3 depicts the formation of process execution model from forms that were initially annotated independent of any processes, and then composed into a process with a process-specific annotation.
4.4 Process Execution

We now explain how a process model in our approach is executed. When an end user starts an instance of some process model \( p = (F|_p, C|_p, A|_p, E|_p, I|_p, O) \), the approach first asks the user to determine the truth of all conditions used in the process (if any), as a set union: \( \bigcup_{F_i \in F|_p} C|_p(F_i) \). This means that conditions which are shared between forms are only evaluated once. The user selects whether the condition applies to her case, i.e., is true for this instance, or not.

Next, the forms without conditions, or whose conditions all were marked to be true, are executed. That is, a form \( F_i \) is only shown to the user if \( C|_p(F_i) = \emptyset \) or \( c_j \) is true for all \( c_j \in C|_p(F_i) \). The execution takes place in the order specified by \( O \). Form execution here means that each form is shown to the user, who can enter values for the fields. The user can go back and forth between the forms.

The process-level field tags (zero or one per field) are hereby interpreted as data mapping: all fields tagged with the same tag are set to the same value. For a fixed but arbitrary \( i \in I \), this set of fields is \( \{ f \in \mathcal{G}(F) \mid F \in F|_p, I|_p(f) = i \} \). This value equality is applied whenever the value for a field in this set is changed.

After filling all desired values into the forms, the user can trigger the next step in the process, where all filled forms can be downloaded in their original format. Finally, all approval emails are sent out for each form \( F_i \) without annotated conditions \( (C|_p(F_i) = \emptyset) \) or where all conditions \( C|_p(F_i) \) are true.
Chapter 5

User Supporting Features

As a way to help the form owners and end users with their tasks in-hand in various stages of the form life-cycle, we built a number of user supporting features throughout the system.

5.1 Recommendation for Adding Tags

For form owners, manually creating tags for a form (e.g., for input fields) can be time-consuming and tedious. To assist, our system includes recommendation algorithms for suggesting tags. This way, a tag can be chosen from a system-suggested list, or newly created by the user if none from the list is suitable. The new tag is added to the knowledge base for sharing and reuse.

We employ different recommendation strategies for different annotation types. For generating a tag recommendation list, we use the actual text appearing in the forms, as they are the most likely terms the form owner is going to find relevant for annotations. To understand the text extraction issue better, we studied 30
publicly available forms from 6 different organisations: UNSW FIPRAS\(^1\), New South Wales Government Licence, Ohio State University Human Resources, Government of Ontario, Bank of New Zealand, and Australian Research Council.

**Input Field Annotation:** Our approach to extracting relevant texts for each input field is i) Analyse real-world forms to find out what kind of information gets frequently reused and ii) Find out how these frequently reused input fields are structured within the form. Our approach to generating a list of tag recommendations from the tag library is by using various heuristics (e.g. which department this form belongs to, and availability of the tag in the tag library, etc).

We first observe that the input fields in forms can be categorised as follows:

- **Personal Details:** Input fields that require personal information such as a name, address, or phone number of the form user.

- **Process Specific:** Input fields that are relevant to the business process the form is used for. For example, in a CSE travel form, this category includes fields such as a location, name, or a date of the conference.

- **Approval Related:** Input fields reserved for approval steps of the process. This includes the name of the approver and the date of approval.

Of all input fields available in these forms, we are only interested in the input fields that are completed by the applicant only, and the input fields that can be completed at the first go. And we assume that the approval related data should not be reused, so the input fields that belong to the approval related category are also omitted from this analysis.

\(^1\)The Finance, Procurement and Assets Unit within School of Computer Science and Engineering
Table 5.1: Different categories of input fields: *Personal Details* vs. *Process Specific*

<table>
<thead>
<tr>
<th></th>
<th>Distinct</th>
<th>~Reused</th>
<th>Reused</th>
<th>#Reused</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal Details</td>
<td>98</td>
<td>42</td>
<td>56</td>
<td>184</td>
<td>1.88</td>
</tr>
<tr>
<td>Process Specific</td>
<td>909</td>
<td>870</td>
<td>39</td>
<td>101</td>
<td>0.11</td>
</tr>
</tbody>
</table>

~: Not, #: Number of times

Table 5.2: Types of reused personal details Input fields

<table>
<thead>
<tr>
<th></th>
<th>Forms</th>
<th>Reused</th>
<th>Textfield</th>
<th>Checkbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNSW FIPRAS</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>NSW Licence</td>
<td>3</td>
<td>11</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>OSU HR</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Ontario</td>
<td>3</td>
<td>14</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>BNZ</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>ARC</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

*Personal Details* vs. *Process Specific* – The analysis shows (Table 5.1) that 56 out of 98 distinct input fields in the *Personal Details* category appear in other forms 184 times, which means that each input field were reused 1.88 times on average, compared to 0.11 times for the the input fields in the *Process Specific* category. This tells us that personal detail fields are much more likely to be reused when there are more than one form involved in a single process. Therefore the heuristics for generating a tag recommendation list target input fields in the *Personal Details* category in order to maximise the benefit.

*Different Input Types* – There are various types of input fields in forms including a textfield, checkbox, and radio button. The next analysis is about investigating what kind of input fields are used for personal details, and see how we can extract relevant texts from these input fields. Table 5.2 shows a number of times a textfield and a checkbox were reused for personal details input fields. It shows that majority of the
input field being reused are of type textfield. This indicates that we can narrow
down the extraction candidates to textfields related to personal details.

*Positioning Text Labels* – Next, we investigated where the relevant texts can be
found within the forms for the input fields we are interested in. For each form, we
analysed the positioning of the text we want to extract relative to each textfield
used to for personal details.

The result (Fig. 5.1) shows that some organisations tend to be consistent in the
label position they use. For example, reused personal input fields in UNSW FIPRAS
forms always had text labels on the left (Fig. 5.1(a)), and for the OSU HR, majority
of the input fields had a text label at the bottom (Fig. 5.1(b)). Of course, it is not
always true for every organisations. For example, the text labels for NSW Licence
department forms (Fig. 5.1(c)) were evenly placed on top, bottom, and left of each
input field.

**Form Directives Annotation:** The form directive annotation we focus on for
this paper is form description tag. There are two sources of information used to
generate the list of tag form description tag recommendation which are: i) the words
extracted from the form and the number of times the word appears in the document,
and ii) the meta-data associated with the forms which contains information about
the title and subject of the form.

**Annotation Knowledge Base:** Based on the observations, we first extract vo-
cabularies that will form our initial “tag library” for input field annotation which
are the labels related to personal details fields. This forms a part of KB that *Ezy-
Forms* uses to select a list of input field tag recommendation. When an input field
is annotated with a new tag that did not exist previously in the tag library, it is
5.1.1 Input Field Annotation

Once we have the text extracted for each input fields, we need to select a list of tags from the tag library to generate a tag recommendation list. The function `findTagRecommendation` listed in Table 5.3 returns a tag recommendation list for a given input field. The list is ranked using MS (Match Score).

The MS is calculated based on three different measures: Label Match (`LabelMatch`), Position Match(`PosMatch`), and Id Match(`IdMatch`). `LabelMatch` provides linguistic similarity between the extracted texts and the tags in the tag library, `Pos-
Table 5.3: Overview of function findTagRecommendation

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>findTagRecommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUTS</td>
<td>$t_i \in T$, $w_o \in W$</td>
</tr>
<tr>
<td>where, $T$ is the set of all tags in the tag library, $T = {t_1, t_2, t_3, \ldots, t_n}$ and $W$ is the set of words extracted from the form by the text extractor denoted by $W = {w_{o_1}, w_{o_2}, w_{o_3}, \ldots, w_{o_m}}$</td>
<td></td>
</tr>
<tr>
<td>OUTPUT</td>
<td>$R_i = {(t_i, MS_i), (t_j, MS_j), (t_k, MS_k), \ldots, (t_l, MS_l)}$</td>
</tr>
<tr>
<td>where, $R_i$ is the list of tag recommendation for input field $i$ and $MS_i$ is the Match Score calculated for the tag $t_i$ ($MS_i \in [0,1]$)</td>
<td></td>
</tr>
</tbody>
</table>

$Match$ takes into account the position of the extracted texts relative to the input field, and $IdMatch$ provides linguistic similarity between the PDF Acroform ID of the input field and the tags in the tag library. The MS is calculated as the weighted average of these three measures as shown in Equation 5.1.

$$MS_i = w_1 * LabelMatch_i + w_2 * PosMatch_i + w_3 * IdMatch_i$$

where, $(0 \leq w_1 \leq 1)$ $(0 \leq w_2 \leq 1)$ $(0 \leq w_3 \leq 1)$ $(w_1 + w_2 + w_3 \leq 1)$

$(PosMatch \in [0,1])$ (5.1)

Weight $w_1$, $w_2$, $w_3$ indicates the contribution of $LabelMatch$, $PosMatch$, and $IdMatch$ respectively. Table 5.4 describes various conditions that were used to calculate the MS. It is noted that currently, we determine the weight values intuitively based on our observation of the forms we have studied, which is explained below. However, it is also possible to determine the optimal value of each weight by experiments.

By default, $IdMatch$ is assigned twice as much weight than $LabelMatch$ and
PosMatch: This is because the field id does not always have a meaningful name, but if the creator of the PDF form has assigned a meaningful id to the fields (e.g. words describing the field instead of randomly generated alphanumerics), it is a very useful source of information we can use. Therefore $w_3$ is assigned twice as much weight than $w_1$ and $w_2$ by default. It is for the same reason that the weight of the IdMatch is set to 1 even when the outcome of the LabelMatch and PosMatch is 0, but the IdMatch is greater than 0.

If the IdMatch is greater than 0, and found a tag with LabelMatch equal to 1, the weight is split between the two equally setting $w_1$ and $w_3$ to 0.5.

When the outcome of LabelMatch is 1, and it is supplemented by PosMatch equal to 1 or IdMatch equal to 1, it is very likely that the tag is a proper tag for the given input field. Therefore we set the final MS for the given tag to 1 by setting $w_1$ to 1.

Finally, if LabelMatch and IdMatch are both equal to 0, the final MS for the given tag is set to 0. This is because PosMatch is only a supplementary score that is meaningful when there is either LabelMatch or IdMatch score is greater than 0.

Different combination of conditions and weights can be experimented other than what’s defined in Table 5.4 to enhance the accuracy of recommendation. As noted, the current combination is by no means complete or proven to be optimal, but rather, a starting point for testing the applicability of our approach in recommending input field tags.

LabelMatch: The extracted texts are first tokenised based on the new line character. Then it removes unnecessary words from the list of string tokens, using a stop-word list. After the extracted texts are tailored, EzyForms enumerates all tags in the tag library and matches string tokens with the tags using Ngram algorithm. The Ngram algorithm calculates the similarity by dividing number of common N-
grams by the total number of N-grams.

**PosMatch:** We take an advantage of the fact that some business units tends to put the text label for personal information input fields in a consistent position for the forms they manage (as shown in Fig. 5.1). This predominant position each business unit uses to position the text label of the input field is referred to as a regular position for a given organisation. For example, the UNSW FIPRAS tends to put the text label on the left side of the textfield(Fig. 5.1(a)), therefore its regular position would be left. The term irregular position is used to denote the text labels that are not positioned in the regular position. For example, input fields that have a text label on the left of the textfield in case of OSU HR(Fig. 5.1(b)) are referred to as positioned in a irregular position. We assume that KB has information about where the regular position is for each business units so that we determine the regular position of each form by checking its form owner’s business unit.

\[
\text{PosMatch}_i = \begin{cases} 
1 & \text{if } \text{pos} \in \text{regular} \land 0 < \text{labelmatch}_i \\
0 & \text{if } \text{pos} \not\in \text{regular}
\end{cases}
\]

where, pos = position of text extracted

regular $\in \{\text{left, top, bottom}\}$ \hspace{1cm} (5.2)
Table 5.5: Overview of function findFormDescriptionTags

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>findFormDescriptionTags</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUTS</td>
<td>wo_i \in W</td>
</tr>
<tr>
<td>where, W is the set of words extracted from the form by the text extractor denoted by W = {wo_0, wo_1, wo_2, \ldots, wo_m }</td>
<td></td>
</tr>
<tr>
<td>OUTPUT</td>
<td>R = {(wo_i, MS_i), (wo_j, MS_j), (wo_k, MS_k), \ldots, (wo_l, MS_l)}</td>
</tr>
<tr>
<td>where, R is the tag recommendation list for form descriptions and MS_i is the Match Score calculated for the word wo_i</td>
<td></td>
</tr>
</tbody>
</table>

The Equation 5.2 shows that for a given input field, PosMatch function returns a score of 1 if the tag matches a text extracted from a regular position, and returns 0 otherwise.

**IdMatch:** The input field Id generated during the PDF form creation time is used to provide extra information for tag recommendation since we expect that many form developers will use reasonable Id that is proximate to the text label. IdMatch function also takes same approach as LabelMatch function. The mapper enumerates each textfield in the form and applies these rules to select tag recommendation from the tag library.

### 5.1.2 Form Directives Annotation

The function findFormDescriptionTags listed in Table 5.5 returns the tag recommendation list for form descriptions for a given form. It extracts words from the form, and then excludes unnecessary words using a stop-word list. After the extracted texts are tailored, EzyForms enumerates each word in the list and calculates the MS (Match Score) for each word.
The MS for form description tag is calculated based on 3 different measures which are Subject Match (SubMatch), Title Match (TitleMatch), and Count Score (CountScore). The SubMatch and TitleMatch matches the word set W with the subject and title metadata of the form. A SubMatch score of 1 is given to the word that is found in the subject metadata, and a TitleMatch score of 1 is given to the word that is found in the title metadata. The CountScore is calculated based on the number of times a word is found in the form. The MS is calculated as the weighted average of these 3 measures as shown in Equation 5.3.

\[
MS_i = w1 \times \text{SubMatch}_i + w2 \times \text{TitleMatch}_i + w3 \times \frac{\text{CountScore}_i}{\text{MaxCountScore}}
\]

where, \( w1 = 0.4, w2 = 0.4, w3 = 0.2 \)

(SubMatch, TitleMatch \( \in [0,1] \)) \hspace{1cm} (5.3)

Weight \( w1, w2, w3 \) indicates the contribution of SubMatch, TitleMatch, and CountScore respectively. The subject and title metadata is not always present, but if the form developer has assigned a meaningful metadata to the fields, it is most significant source of information we can use. Therefore \( w1 \) and \( w2 \) is assigned a bigger weight than \( w3 \) by default. The CountScore for each word is divided by the maximum count score (MaxCountScore) obtained out of all available words in order to normalise a score between 0 to 1.

### 5.2 Candidate Generation for Mappings

For form owners, the mapping task in form-based process modelling involves identifying input fields that share the same value across forms so that the value is only request once from the user. During the modelling phase, the form owners will be asked to identify the mappings by examining forms. To assist, we automatically generate a suggested list for mapping candidates in the chosen forms. Each input
Table 5.6: Algorithm for input field mapping recommendation

Let $SIF$: a selected input field
$tags$: annotated tags for the SIF
$IFs$: a list of available input fields in the process except $SIF$
$t(f)$: annotated tags for an input field $f$
$n$: number of common tags
$m<key,val>$: mapping recommendations ordered by size of $val$

Get\_Mapping\_Recommendation($SIF$)
Foreach $f$ in $IFs$ do {
    \hspace{0.5cm} n=0;
    \hspace{0.5cm} Foreach tag in $t(f)$ do
    \hspace{1cm} if($tags$.contains(tag)) n++;
    \hspace{0.5cm} $m$.add($f$, n);
}
return $m$;

field has zero or more annotation tags associated with it. We take the tags in an input field with those appearing in other input fields. We then calculate the number of common tags between any two input fields. Based on the common tags, we generate a ranked list of mapping candidates. The larger the number of common tags, the more likely the input fields are to share the same input data. The algorithm achieving this is shown in Table 5.6. The larger the number of common tags, the more likely the input fields are to share same input data. *EzyForms* presents the potential mappings to the process modeller and provides tooling support to easily apply new mappings between input fields. Figure 5.2 shows the screen capture of process modelling where the process modeller is mapping Club Name input fields between two different forms using the data mapping recommendation. A mapping
means that the input data entered into one mapped field will be copied to all other mapped fields.

![Mapping input fields between two different forms using mapping recommendation](image)

Figure 5.2: Mapping input fields between two different forms using mapping recommendation

### 5.3 Saving and Loading of Process Execution Data

For the end users, it may be useful to be able to save the form data they typed in during the execution of a process, and be able to reuse the same data next time the same process is required. To assist, we allow the end users to save the entire form data entered into a process into a local system file, after the execution. The file can be loaded into *EzyForms* when the same process needs to run. The input data is saved as *key-value* pairs. This technique relieves the end users from re-entering data for each process instance, while circumventing security/privacy concerns through local storage on the end user’s machine.
Chapter 6

Architecture and Implementation

A prototype has been implemented to evaluate the feasibility of our approach and its screencast is available at http://www.cse.unsw.edu.au/~hpaik/ezyforms.

![EzyForms Architecture](image)

Figure 6.1: *EzyForms* Architecture

The overall architecture is shown in Fig. 6.1, which is partitioned into form upload, form annotation, process modelling, and process execution components.
6.1 Form Upload

The forms uploaded into the system are stored into the repository by the FormSys Core Component. It is also responsible for soap2pdf service generation [63]. Forms are central to EzyForms. All of the actions by the human actors take place on the form itself. While the concept of automating the form-based process with form annotation is generic, we used specific technologies for the purpose of demonstrating the concept, like using Adobe PDF/AcroForm as the form document format. The decision was made based on the fact that Adobe PDF is the de-facto Web standard for exchanging read-only fixed-layout documents and it provides rich application programming interfaces (APIs) to interact with forms. It is also responsible for soap2pdf service generation that is based on standard Web service technology. Apache CFX\(^1\) is used for all aspects related to Web services, including Web service implementations.

When the form is first uploaded into the system, FormSys Core Component also generates a basic XML schema file in the local directory that contains information about each form documents uploaded to the system. This form XML file contains list of input field elements, approver elements and condition elements. Since no approvers or conditions have been annotated at this stage, these elements contains no children, however, as forms get annotated at the form annotation stage, the elements will get child elements appended.

6.2 Form Annotation

Text Extractor uses the iText\(^2\) library to extract raw text from the PDF form doc-

\(^1\)http://cxf.apache.org/
\(^2\)http://itextpdf.com/
ument and passes it to the Matcher component that uses the tag selection algorithm to generate tag recommendation list from the tag library. Tag Library Manager component is responsible for managing tag library and tag recommendations. For prototype implementation, the tag library initially contains number of strings that are commonly used for personal information such as name, address, email address, phone number and so on. In Form Annotation Manager, forms are indexed using Apache Lucene\(^3\) when they are uploaded into the system. The indices are created on the form’s title text, file name, text content as well as the annotation tags. Form Annotation Manager is also responsible for interacting with the form XML file to add or update approvers, conditions, or input field tags as the form owners annotate the form document that was uploaded at the form upload stage.

6.3 Process Modelling

Based on the annotated input fields, Input Field Mapper generates mapping candidates for any selected input fields amongst forms during the modelling process. Process Manager stores and manages all processes created by the form owners.

A process model is created through three steps of activities: form selection, data mapping, refine annotations as necessary. All these activities are supported through a graphical user interface which features forms as first class citizens. The resulting process definition is expressed in XML. Figure 6.2 shows snippets of the model documents which is expressed in XML format for forms with id attribute 260 and 251. The approver inputfield in form 260 and if.10727 inputfield in form 251 contains two identical input field tags which makes them a potential mapping candidate for data reuse. Another potential mapping is within form 251 where if.10727 inputfield

\(^3\)http://lucene.apache.org/
shares one identical tag with template input field with id attribute 1.

Figure 6.2: Possible mappings in the input fields: two forms and an email template

If the modeller decides to map two input fields, the rep element value will be set to the same between the two input fields, and the execution engine will fill these input fields with the same input data.

6.4 Process Execution

When an end user instantiates a process, Execution Engine first interacts with the end user to evaluate any conditions in the process, determines forms to be filled-in and presents the forms for input and executes the form services. Finally, it interacts with Email Dispatcher to send the forms according to the approval chain
Executing a form-based process involves three phases: *pre-processing*, *form service execution*, and *post-processing*. The form-based processes that are supported by *EzyForms* are transient processes that are short-lived and request-response style processes. The input data from an end user starting a process are handled by the front-end of the system during *pre-processing*: forms that are applicable to the current end user are retrieved, the end user’s input data is retrieved, and based on the data mapping the input data is to forwarded to the corresponding input fields across the forms.

**Figure 6.3: Process executed by the end users**

Figure 6.3 shows the process executed by the form user. When a process is deployed successfully, *EzyForms* provides a link that the form user can click to initiate the process. *EzyForms* provides an execution environment as if the end user is filling in the actual form document by displaying the form image in the background and overlaying textfields in appropriate positions for the form user to input data.
EzyForms displays a single page at a time and provides a left and right button(2) to flip through the pages. When the first form of the process is completed, the form user can move on to the next form document by clicking next form button(3) or go back to the previous form by clicking the previous form button(4). When the form user moves to the next form, the input data entered in the previous form(5) is passed on to the next form and is used to fill in the input field with same tag name(6). Once all forms are completed (submit button(7)), the execution engine invokes each form service to fill in the form. Once the process is completed successfully, a link(8) to the filled in form documents is returned to the form user.

Figure 6.4: Example data flow showing how email input data is passed to fill in a form via Web service invocation at process execution stage

Form service execution phase is where the actual form filling operations occurs. Figure 6.4 illustrates how input data is passed to the form service, using the example of the end user’s email address. The name of HTML input field element that is presented to the end user is determined by the representative tag name of the input field associated with process-specific annotation. These HTML input fields are mapped to the form-specific annotation before invoking the form filling service.

An active soap2pdf service is a full-fledged Web service offering a WSDL interface with a single operation: fillForm. When the end user completes the data entry,
a soap2pdf service is invoked for each form with end user’s inputs as input to the fillForm operation; each service returns a URL to the filled form (if successful), which are displayed to the end user in the next phase.

The end user can review the filled in forms during post-processing, and may trigger further actions. Figure 6.5 shows the summary of the filled forms and the actions the end user may take. It contains a list of approvers and the forms associated to them, the preview of an email template with the place-holders filled in with the end user’s input data, and the link where the filled-in form can be reviewed. If the end user chooses to trigger the approval, it is handled using approval queues for forms where approvers are annotated. EzyForms currently supports a sequential approval pattern, as follows. Approval requests are emailed one at a time; subsequently, EzyForms awaits the approver’s response via email. In case of a rejection, the system sends a rejection notification to the original requestor. In case of approval, the system moves on to the next approver or, if all approvers accepted, sends an acceptance notification to the original requestor.

Another important aspect of post-processing is enabling end user to store current input data as a file locally, so that it can be reused the next time when executing the same process. The input data is saved as key-value pairs in which the representative tag is used as a key. When the input data file is loaded for another process execution, EzyForms matches the key to the name of form input field and fills the previous input data.

---

4The first line of the email body is expected to contain a certain keyword (e.g., “approved” or “rejected”). As mentioned before, email replies are used in lieu of digital signatures.
6.5 Approval Chains

The Email Dispatcher component is composed of 3 main components which are Approval Chain Models, Sender, and Polling Receiver as shown in Fig. 6.6. From the approver annotations, we auto-generate an approval chain model that supports a multi-level hierarchy (e.g., supervisor, then Head of School, then the Dean) as well as “parallel-splits” (e.g., signed by two supervisors)\(^5\).

From the approval chain model, EzyForms dispatches approval request emails to appropriate approvers and collate the responses. The current implementation reads the content of the email to determine if the request has been approved, that is, if the reply text of the email contains approved or yes, the request is considered approved and the next email is dispatched to the next approver in the chain. If declined, the process is terminated and the notification email is sent to the applicant.

\(^5\)Note that it is possible to plug-in an algorithm which takes different interpretation of the given approver annotations
6.5.1 Approval Chain Model

The approval chain model is the basis for the Sender and Polling Receiver component to execute. In this implementation, the approval chain model is described in XML which is based on the approval chain model schema, and is always generated before the execution of an approval process. The XML file that represents each approval chain has an unique name that acts as an identifier for each instance of the process. There are three data types defined in the approval chain model schema which are the applicant, the approver, and the chain model.

The Applicant

The Applicant element in the schema is a complex data type that contains the necessary information of the applicant. Currently, only one applicant is allowed for each process, and represented with the email address as shown in Table 6.1.
Table 6.1: Applicant Type

\[
\begin{array}{l}
< \texttt{xsd}: \texttt{complexType} \texttt{typeName} = "\texttt{Applicant}" > \\
< \texttt{xsd}: \texttt{sequence} > \\
\quad < \texttt{xsd}: \texttt{elementname} = "\texttt{Email}" > \\
\quad \quad < \texttt{xsd}: \texttt{simpleType} > \\
\quad \quad \quad < \texttt{xsd}: \texttt{restrictionbase} = "\texttt{xsd}: \texttt{string}" > \\
\quad \quad \quad \quad < \texttt{xsd}: \texttt{patternvalue} = "\texttt{\w+ ([-+]\w+) * @\w+ ([-+]\w+) * \w+ ([-+]\w+) *" } > \\
\quad \quad \quad < /\texttt{xsd}: \texttt{restriction} > \\
\quad \quad < /\texttt{xsd}: \texttt{simpleType} > \\
\quad < /\texttt{xsd}: \texttt{element} > \\
\quad < /\texttt{xsd}: \texttt{sequence} > \\
\quad < /\texttt{xsd}: \texttt{complexType} > \\
\end{array}
\]

The Approver

The Approver element is a complex data type that contains the necessary information about the approvers who are involved in the processes. The information includes the email address of the approver, the subject and content of the email template used for each approver, and the attachment that gets sent to the approver denoted by \(< \texttt{Email} >, < \texttt{Subject} >, < \texttt{Content} > \) and \(< \texttt{Attachment} > \) as shown in Table 6.2. More than one approver can be included in the approval process, and each approver is identified with unique id attribute.

The Chain Model

The Approver element in the schema is a complex data type that specifies the process flow of the approval chain. Each approval chain model contains one Model element and each Model element consists of several Node element as shown in Table 6.3.
The Node element has four attributes which are id, approverID, state and joint. The id attribute is an unique identifier for each Node element, approverID is used to map between the node and the Approver element, state attribute is used to control the flow of approval process, and joint attribute is used to distinguish whether the node is a simple node or a complex node. The type of state that is used to control the flow of approval process is shown in Table 6.6.

6.5.2 Sender

The Sender component is implemented using Java Mail API and is responsible for dispatching an email to the specified email address. The Sender component is triggered by Execution Engine as the end user submits the request, and also when the Polling Receiver retrieves the approvers’ responses. Based on the approval chain model, the Sender component dispatches more emails to the approvers in the next level, or if the approval process is finished, it dispatches a notification email to the applicant.

6.5.3 Polling Receiver

The Polling Receiver continuously polls the inbox of the Gmail account created for EzyForms to check if the approvers’ responses have arrived. We use Java Mail API to communicate with the Gmail Server and the polling mechanism is implemented with Servlet’s ServletContextAttributeListener class which will run while the application (i.e., EzyForms) is running.

Once the approver’s response has arrived, the Polling Receiver will first check the “In-Reply-To” field in the email message header. The format used for this header is <messageID.nodeID@EzyForms>. The messageID is the name of the approver
chain model file, and the nodeID refers to the node in the approval chain model that this message corresponds to.

When the Polling Receiver retrieves a response from the email server, it will look for the approval chain model file in KB using the messageID. If the approval chain model file is found, the content of the response is evaluated to check if the approval request has been “approved” or “declined”. Depending on the message content, the “messageID” and “nodeID” is added to either the approved list or the declined list. If the message does not contain a valid content, the replies are added to the resend list.

Approved List.

For each item in the approved list, the state of the node whose id is same as the nodeID is inspected. If the current state is “1”, the Polling Receiver will move on to the next approver if there is any. Otherwise, the node has already been processed and the incoming response is a duplicate message. Once the nodeID has been identified and the status is checked, the Polling Receiver will identify the nodes involved in the next step according to the approval chain model file. If the node is a simple node, the Sender component will be invoked to dispatch an approval request email to the approver that the current node represents, and the state of this node is updated to “1”. If the node is a complex node with a joint, then the Polling Receiver will update the state to “2” and increase the accepted token number. If the operator is “and”, and accepted token number has reached to the max token number, the Polling Receiver will move on to the next nodes until the next node is a simple node or the accepted token number and the max token number is different. If the operator is “or”, then the immediately move on to the next nodes until the next node is a simple node or the accepted token number
and the max token number is different. When the last node is reached, the **Polling Receiver** will update state of the node to “2” and send the notification email to the applicant.

**Declined List.**

For denied list, **Polling Receiver** checks the state of the node whose id is same as the nodeID just as with the approved list. After that, the **Polling Receiver** will check the next node according to the approval chain model file. If the node is simple node or the node is a joint node but the operator is “and”, then the **Polling Receiver** will update its state to “3” and at the same time update the state of the last node to “3”. If the node is complex node with joint and the operator is “and”, then the **Polling Receiver** will increment the denied token number. If the denied token number is incremented up to the max token number, the **Polling Receiver** will update the state of the last node to “3” and send the failure message to the applicant.

**Resend List.**

For the resend list, the **Polling Receiver** will dispatch the approval request email again and remind approvers to reply with valid content (i.e., *approved* or *declined*).
Table 6.2: Approver Type

```xml
<xs:complexType name="Approver">
  <xs:sequence>
    <xs:element name="Email" type="xs:string">
      <xs:restriction base="xs:string">
        <xs:pattern value="\w+([-+.]\w*)*\@\w+([-+.\w]*\@\w+([-+.\w]*\@\w+)*"/>
      </xs:restriction>
    </xs:element>
    <xs:element name="Subject" type="xs:string"/>
    <xs:element name="Content" type="xs:string"/>
    <xs:element name="Attachment" minOccurs="0" maxOccurs="unbounded" type="xs:string">
      <xs:restriction base="xs:string">
        <xs:pattern value="(\.){1,256}.pdf"/>
      </xs:restriction>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

Table 6.3: Model Type

```xml
<xs:complexType name="Model">
  <xs:sequence>
    <xs:element name="Node" minOccurs="unbounded" maxOccurs="unbounded" type="Node"/>
  </xs:sequence>
</xs:complexType>
```
Table 6.4: Node Type

```xml
<xsd:complexType name="Node">
  <xsd:sequence>
    <xsd:element name="Joint" minOccurs="0" type="xsd:Joint"/>
    <xsd:element name="Next" minOccurs="unbounded" type="xsd:int"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:int"/>
  <xsd:attribute name="approverID" type="xsd:int"/>
  <xsd:attribute name="state" type="xsd:int"/>
  <xsd:attribute name="joint" type="xsd:boolean"/>
</xsd:complexType>
```

Table 6.5: Joint Type

```xml
<xsd:complexType name="Joint">
  <xsd:attribute name="operator" type="xsd:string"/>
  <xsd:attribute name="deniedToken" type="xsd:int"/>
  <xsd:attribute name="acceptedToken" type="xsd:int"/>
  <xsd:attribute name="maxToken" type="xsd:int"/>
</xsd:complexType>
```

Table 6.6: Different states of nodes

<table>
<thead>
<tr>
<th>State number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The node is yet to be processed</td>
</tr>
<tr>
<td>1</td>
<td>The node is waiting for a response from the approver</td>
</tr>
<tr>
<td>2</td>
<td>The approver represented by this node has approved the form</td>
</tr>
<tr>
<td>3</td>
<td>The approver represented by this node has declined the request</td>
</tr>
</tbody>
</table>
Chapter 7

Evaluation

In this chapter, we present the results from the evaluation that was conducted to test the applicability of our proposed approach. The first evaluation tests how well tag recommendation feature works to help the form owners with the input field annotation. Being able to recommend tags accurately for each input field can significantly increase the performance of tagging activity in terms of speed and usability, and hence, number of different forms from different organisations were used to evaluate the accuracy. The second evaluation was the user study that was conducted with fifteen participants. In this user study, the participants were asked to play two roles which are the form owner and the end user. For each role, they were given a task to complete and were asked to rate the system based on their experience.

7.1 Evaluation of Tag Recommendation Accuracy

For the evaluation of tag recommendation accuracy, we focus on evaluating how well our text extraction and matching score works in selecting a proper tag from the tag
Table 7.1: Evaluation results for our tag recommendation strategy for personal details input fields

<table>
<thead>
<tr>
<th>Organisation</th>
<th>Number of forms</th>
<th>Input fields</th>
<th>S@1</th>
<th>S@5</th>
<th>AMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNSW FIPRAS</td>
<td>6</td>
<td>31</td>
<td>90%</td>
<td>97%</td>
<td>0.81</td>
</tr>
<tr>
<td>NSW Licence</td>
<td>3</td>
<td>30</td>
<td>47%</td>
<td>73%</td>
<td>0.41</td>
</tr>
<tr>
<td>OSU HR</td>
<td>5</td>
<td>33</td>
<td>61%</td>
<td>84%</td>
<td>0.65</td>
</tr>
<tr>
<td>Ontario</td>
<td>3</td>
<td>42</td>
<td>64%</td>
<td>86%</td>
<td>0.66</td>
</tr>
<tr>
<td>BNZ</td>
<td>7</td>
<td>49</td>
<td>88%</td>
<td>98%</td>
<td>0.80</td>
</tr>
<tr>
<td>ARC</td>
<td>7</td>
<td>37</td>
<td>100%</td>
<td>100%</td>
<td>0.99</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>76%</strong></td>
<td><strong>90%</strong></td>
<td></td>
<td></td>
<td><strong>0.68</strong></td>
</tr>
</tbody>
</table>

library. We make an assumption that we have a populated tag library that contains input field labels for personal information for every form document we test, and use common sense to decide whether we have proper tag recommendations or not. We adopted two metrics, that captures the accuracy of tag recommendation at different aspects:

**Success at rank k** (**S@k**) represents the probability of finding a good descriptive tag among the top k recommended tags. For this evaluation, we used two values of k: S@1 and S@5.

**Average Match Score** (**AMS**) represents the match score of first relevant tag returned by the system, averaged over all input fields that found a relevant tag. An AMS value is calculated per organisation.

Based on the evaluation result shown in Table 7.1, we observed that 76% of the time, the proper tag for each personal details input field received the highest score amongst a list of recommended tags, and 90% of the time, the proper tag was found amongst top 5 tag recommendations. The Average Match Score for input fields in 30 forms from 6 different organisations were 0.68. In observing input fields
that did not retrieve a proper tag, we have found three main reasons why *EzyForms* were not able to generate a proper tag recommendation. First of all, it was due to *irregular* position of the text label. When it’s difficult to determine which is the *regular* position for a certain organisation (i.e., positions of the text labels are not consistent), it is difficult for *EzyForms* to decide which tag is more relevant than the other when there are raw texts extracted from more than one position. Second, some input fields have rather descriptive text labels (e.g., Address (*licence address must be within NSW*)), and since input field tag recommendation strategy relies on the *Ngram* algorithm to compare the strings, more descriptive text label tends to hamper finding a proper tag recommendation. The third reason is somewhat relevant to the previous point. It is the size of the rectangle we use to extract raw texts. Since the length of the text labels is all different, the size of the rectangle could be too small that it did not extract enough information to determine a proper tag, or it could be too big that the size of the raw texts extracted is too big for accurate recommendation.

### 7.2 User Study

A user study with fifteen participants was conducted using *EzyForms* to evaluate our approach. The goals of the study were to evaluate whether: i) our form annotation approach can be applied to people with little technical background (especially in BPM) to automate form-based processes, ii) user supporting features were helpful, and iii) end users find the automated execution of a form-based process convenient.
7.2.1 User Study Method Overview

We asked the participants to self-rate their knowledge on IT skills including business process modelling and automation. Later, we used the answers to categorise the evaluation results into ‘experienced users’ and ‘novice users’ groups.

All participants were given approximately 15 minutes of overview/induction session on the user study and the tool. The task scenario was based on a university student activity grant application process and the tasks were identical for all participants. We first introduced the participants the manual ‘as-is’ situation of the process. The participants were given two paper forms\(^1\) and asked to observe the forms and the process instructions. Then, using our tool, the participants played two different roles:

**form owner:** in this role, the participants performed form annotation and process modelling tasks. For the form annotation task, they first identified the form fields that they thought would frequently appear across other forms, and annotated those fields with descriptive tag names. The conductor of the study made observations on the tagging behaviour and usage of tag recommendation feature. For the process modelling task, the participants used the two forms to create an activity grant application process. The conductor of the study made observations on the use of data mapping options and mapping candidate generation feature. At the end of each task, the following questions were asked:

- How easy was it to understand and perform each task? - (Understanding how it works)

\(^1\)http://www.arc.unsw.edu.au/get-involved/clubs-and-societies/club-forms-and-policy
Evaluation

- How easy was it to understand the purpose of each task? - (Understanding how it is used)
- How easy was it to complete the task with a given tool? - (Tool usability)
- How helpful did you find the user supporting features for completing the task? - (User supporting features for form owner tasks)

**form end user:** in this role, the participants were asked to execute the process created using *EzyForms* and evaluate the experience against the ‘as-is’ situation. In order to emulate the execution of manual form-based process, we have prepared cards, each describing the steps in the form-based process (as described in Sect. 1.2), and asked the participants to put them in the order that they would perform if they were carrying out the process manually. Then, they completed the same process (but automated) using our tool. The participants were asked to evaluate how much improvement they saw compared to the manual process.

### 7.2.2 Results and Discussion

Out of the total fifteen, we categorised 7 participants into ‘experienced users’ and 8 participants into ‘novice users’.

**On the feasibility of the annotation approach.**

All fifteen participants were able to use the tool and complete the given set of tasks designed for the form owner’s role – without any extra help, regardless of their respective category. The questionnaire results on the tasks as the form owner (Fig. 7.1) show that the understandability of tasks they have to perform, and the usability of the tool are scored high (well over 4 in a 5-point scale) across the user groups.
Also, we can observe that the results show little differences between the two groups across the tasks. Hence, overall, we believe our proposed approach to automating the form-based process (and its implementation in EzyForms) is applicable to both groups and not bound to any process modelling experience.

![Graphs showing scores from questionnaires on form owner's tasks.](a) on Form Annotation Task (b) on Process Modelling Task (incl. Data Mapping)

Figure 7.1: Scores from the questionnaires on form owner’s tasks - rated on a 5-point Likert scale (1 being worst, 5 being best).

**On user supporting features (form owner perspective).**

We also collected ratings for the user supporting features intended for form owners, namely the input field tag recommendation (during form annotation tasks) and mapping candidate generations (during process modelling tasks).

The participants scored 4.8/5 and 5/5 for tag recommendation, and mapping candidate generations features respectively. Table 7.2 shows that, on average, the input fields which did not have any tag recommendations had 1.17 tags, compared to 2.09 tags on those with tag recommendations. Out of 2.09 tags, 2.04 were selected from the recommended tag list. This observation shows that the participants were likely to add annotations (tags) more actively when there were recommendations
Table 7.2: Types of reused personal details input fields

<table>
<thead>
<tr>
<th>Input field type</th>
<th>Average number of people selected</th>
<th>added from recommendation</th>
<th>added manually</th>
<th>Total average</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o recommendations</td>
<td>3.67</td>
<td>0</td>
<td>1.17</td>
<td>1.17</td>
</tr>
<tr>
<td>w/ recommendation</td>
<td>12.11</td>
<td>2.04</td>
<td>0.05</td>
<td>2.09</td>
</tr>
</tbody>
</table>

made from the tool (about 80% increase in the number of tags) This also indicates that better recommendations may lead to better usability at the process modelling phase.

On process execution (end user perspective).

(a) Process execution steps (end user experience) - rated on a 5-point Likert scale

(b) favourite features

Figure 7.2: Form annotation user study evaluation

All participants were able to complete the tasks given for the second role, form end user. Figure 7.2(a) shows the scores on the questionnaires which asked to rate the amount of improvement they saw at each steps of the form-based process, compared to the manual ones. All participants commented that EzyForms allowed them to conduct the process in more efficient manner. This is largely due to the fact that
most manual work was either completely removed or significantly reduced (e.g., identifying which forms to fill-in, downloading forms to fill-in). Finally, each participant selected three favourite features from our approach without specific order. It shows that the most popular point was that they no longer had to fill-in same information repeatedly, closely followed by the point that they did not have to identify required forms by themselves (the conditions annotation in our tool automates that aspect) (Fig. 7.2(b)).
Chapter 8

Conclusion

Business process management has become an integral part of today’s business environment for organisations that deal with large number of processes. Despite the advances in process management solutions and approaches, they have typically been focusing on automating the processes that are highly repetitive and have significant impact on the performance of the organisations.

The processes that are not adequately addressed by these systems are often left manual, leaving most of the work to be done by the end users. Organisations do not see significant ROI for building new solution or customising a commercial tools to automate these type of processes because they usually have less impact in terms of business performance. These type of processes, which we call Long Tail of the processes can be seen everywhere, within organisations, academic institutes, governments, and so on. The research work and products that aims to automate the Long Tail of the processes are quite technical that requires re-engineering of the process or experienced process modeller.

In this thesis, we claimed that there is a need to provide more practical approach for automating such processes, an approach that non-technical people can easily
adapt and for organisations to adopt without much burden in investment. Our focus was on automating the form-based process in its “AS-IS” state which means that we aim to use existing assets, people, and not try to change the current process flow in order to ensure smooth transition for both organisations and the end users.

8.1 Summary of Work Undertaken

The thesis has presented a pragmatic approach for automating form-based processes. In this approach, the form-based process model and execution are deliberately kept plain so that deriving their definitions for automation is attainable through tags and other simple form of annotations by the form owners. In this work, we have identified the types of annotations that can be used to support the end-to-end life-cycle of form-based processes. We have developed a fully working prototype named EzyForms as proof of concept, and conducted a user study to evaluate its applicability.

In summary, the following contributions have been made as a result of this work.

- Analysis on the forms being used in real-world for tag recommendation strategy; specifically, for input field and form description annotation, and the evaluation of input field tag recommendation strategy.

- Identification of different types of annotation for simple, implicit modelling and execution of form-based processes.

- Smart applications of the annotations to support basic data flows and execution patterns in form-based processes.

- Design and development of process model and runtime environment that supports the whole life-cycle of form-based process management.
• Demonstration and evaluation of the automated form-based processes in terms of applicability and usability through user studies.

8.2 Summary of Findings

Our preliminary evaluation based on user study with 15 participants revealed that form annotation approach for automating form-based processes are applicable to people with little or no technical background, that the form annotation and process modelling task was intuitive enough to understand and complete. Part of the credit goes to the tooling support and user supporting features such as tag and mapping recommendation. For input field tag recommendation, a proper tag was found amongst top 5 recommendations using our proposed tag recommendation, and as a result, the participants found it useful and made active use of it. Furthermore, we found that the tag recommendation significantly increases the number of tags added to each input field, which in result leads to better input field mapping recommendations.

The evaluation also showed that automated form-based processes have improved the overall user experience. Our survey revealed that each form-based process steps have improved to a certain degree. The participants rated the amount of improvement they saw in each step, and the Form Download step scored the highest improvement (4.9 out of 5) followed by Form Identification, Form Fill-In, Write Email, Approval Request, and Process Identification. The participants were also asked to select their favourite feature from the automated form-based processes, and the result showed that the participants were most favourable to the removal of duplicate inputs, closely followed by not having to identify the required form for each process.
8.3 Directions for Future Work

Although our research work has provided an initial set of results in applying annotation to automating the form-based processes, we would like to see EzyForms being expanded to support more diverse scenarios of form-based processes. This includes form-based processes that involve more than one user, form documents in various formats such as Microsoft Word and asynchronous processes.

In addition, process data management issue can be explored further so that EzyForms can deal with process specific data (e.g., id number of the project that will fund the travel) and sharing of such data between users. The tooling features to include more complex input fields mapping (e.g., concatenating two strings or manipulating dates) and file attachment to the forms are another possible future work. Applying semantic technologies to the annotations would be an interesting area to explore. We anticipate that the quality of recommendation for tagging and data mapping can improve significantly if the tags are semantically associated with each other. Another interesting issue to look into is homonym. Being able to address homonym properly would further increase the accuracy of recommendation and usability of the system.
Bibliography


