# Automating Form-based Processes through Annotation*

Sung Wook Kim[1], Hye-Young Paik[1], and Ingo Weber[1,2]

[1] School of Computer Science & Engineering, University of New South Wales
[2] Software Systems Research Group, NICTA, Sydney, Australia†
{skim,hpaik,ingo.weber}@cse.unsw.edu.au

**Abstract.** Despite all efforts to support processes through IT, processes based on paper forms are still prevalent. In this paper, we propose a cost-effective and non-technical approach to automate form-based processes. The approach builds on several types of annotations: to help collect and distribute information for form fields; to choose appropriate process execution paths; and to support email distribution or approval for filled forms. We implemented the approach in a prototype, called *EzyForms*. We conducted a user study with 15 participants, showing that people with little technical background were able to automate the existing form-based processes efficiently.

## 1 Introduction

Business Process Management Systems (BPMS) enable organisations to automate and continuously improve business processes in order to achieve better performance. 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 long tail of processes, i.e. highly customised processes that are unique to individual organisations, or concern a small number of workers.

In this paper, we particularly focus on the long tail of processes that consists of form documents. Forms provide a low-tech, quick and flexible way to manage processes. However, they impose a good deal of manual labour on the end users, such as searching/downloading required forms, entering the same information repeatedly, printing/faxing forms and so on.

A typical form-based processes exhibit the following characteristics. 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 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

*This work is supported by SDA Project, Smart Services CRC, Australia
†NICTA, http://www.nicta.com.au/about/

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. Although there are other types of form-based processes (e.g., a process involving more than one user), our current work focuses on this model initially.

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. Our contributions in this paper are:

- a prototype named *EzyForms*, a framework that supports the whole life-cycle of form-based process management. It includes the following novel aspects:
  - 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,
  - a WYSIWYG-fashion integration of forms in every stage of the process life-cycle, from form service creation through to modelling and execution.
- evaluation of such a framework through user studies.

Note that a full version of the paper is available at [1].

## 2 Proposed Approach

We aim to enable form owners to automate the form-based processes themselves, and allow end users to execute the process efficiently. This approach consists of four steps: form upload, form annotation, process modelling, and process execution.

### 2.1 Form Upload

In order to fill-in the forms electronically, we convert a form into a Web service. This is done by our previous work, FormSys [2], through which PDF forms[3] are uploaded to the central FormSys repository. 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 $\mathcal{G}(F_i) := \{f_1, f_2, \ldots, f_n\}$.

### 2.2 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 form B). This is, of course, not always feasible or accurate. To bridge the gap, we introduce the form annotation step which is an act of adding metadata to the form. Due to its informality and flexibility in use, annotation

[3] We use AcroForm, a PDF sub-standard, which contains interactive form elements.

has been widely accepted by users for assisting Web browsing or searching, and utilised in many systems like Flickr, Delicious, and Youtube [3, 4]

There are two types of annotations, each assisting different aspects of process automation: annotation for process modelling and for process execution. Also, 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. 2.3).

**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.
- *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.

**Annotation for Execution.**

- *Condition*: This type of annotation specifies conditions under which the form should be used. 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. Details on all execution aspects are given below.
- *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 $E : \mathcal{F} \times A \mapsto \mathcal{E}$, 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.

Note that the collected annotations on different forms by different form owners are centrally managed and shared via *EzyForms* Knowledge Base (KB). Also, adding annotations is an activity separate from process modelling tasks and it is possible that annotation and modelling are done by different people.

### 2.3 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 optional part of the flow (i.e., which form is relevant for the current user).

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

**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 $p := (\mathcal{F}|_p, C|_p, A|_p, E|_p, I|_p, O)$, such that:

- $\mathcal{F}|_p \subseteq \mathcal{F}$ is a projection from the set of all forms to its subset used in $p$.
- $C|_p : \mathcal{F}|_p \mapsto 2^{\mathcal{C}}$ 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 $F_i \in \mathcal{F}|_p$ and $c_{i_1}, \ldots, c_{i_k} \in \mathcal{C}$.
- $A|_p : \mathcal{F}|_p \mapsto 2^{\mathcal{A}}$ 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 $F_i \in \mathcal{F}|_p$ and $a_{i_1}, \ldots, a_{i_k} \in \mathcal{A}$.
- $E|_p : \mathcal{F}|_p \times A \mapsto \mathcal{E}$ 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.
- $I|_p : \mathcal{G}(F_k) \mapsto \{\mathcal{I}, \perp\}$ is defined as a mapping from a form field to zero or one field tag: $I|_p(f_j) = \perp$ (no field tag) or $I|_p(f_j) = i_j$, where $f_j \in \mathcal{G}(F_k)$, the form belongs to $p$: $F_k \in \mathcal{F}|_p$, and $i_j \in \mathcal{I}$.

– $O$ specifies the order of the forms in $p$, and thus is an ordered permutation of $\mathcal{F}|_p : O = (F_{i_1}, \ldots, F_{i_k})$ 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 is 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.

### 2.4 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 $p = (\mathcal{F}|_p, C|_p, A|_p, E|_p, I|_p, O)$, the approach first asks the user to determine the truth of all conditions used in the process (if any), as a set union: $\bigcup_{F_i \in \mathcal{F}|_p} C|_p(F_i)$. 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 $i \in \mathcal{I}$, this set of fields is $\{f \in \mathcal{G}(F) \mid F \in \mathcal{F}|_p, I|_p(f) = i\}$. 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.

## 3 *EzyForms* Implementation

A prototype (Fig. 1) has been implemented and its screencast is available at `http://www.cse.unsw.edu.au/~skim/ezyforms`.

**Form Upload and Annotation.** The forms and their matching Web services are stored in the repository in *FormSys Core Component*. `Matcher` and `Form Annotation Manager` components are responsible for managing tag library and tag recommendations. We use Apache Lucene[4] to create indices on the form's title text, file name, text content as well as the annotation tags.

**Process Modelling and Execution.** `Input Field Mapper` generates mapping candidates for any selected input fields amongst forms during the modelling

[4] http://lucene.apache.org/

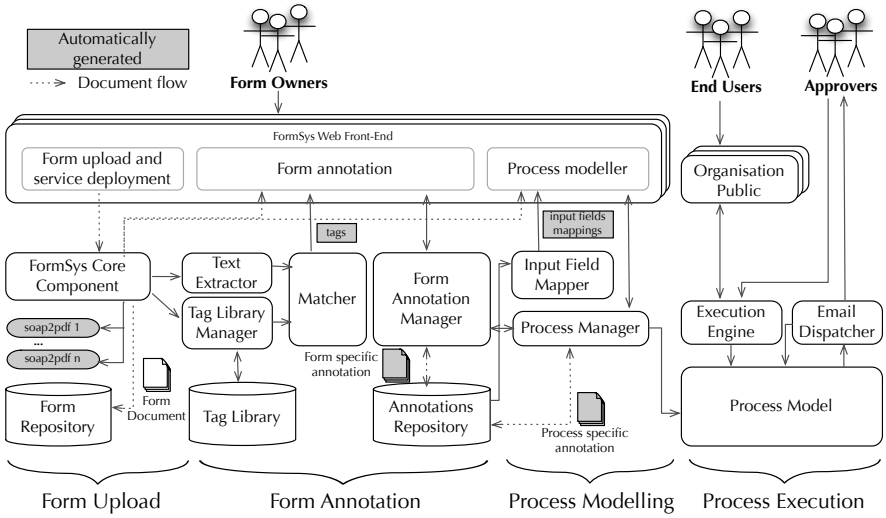



**Fig. 1.** *EzyForms* Architecture

process. `Process Manager` stores and manages all processes created by the form owners. 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, 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 generated.

**Approval Chains.** 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). From the approval chain model, *EzyForms* dispatches approval request emails to appropriate approvers and collate the responses.

**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. For example, the algorithm for recommending form description tags will consider elements such as title text of the form, or free text appearing after the title (which is normally a description/instruction about the form) to derive potentially relevant terms. The details of the tag recommendation algorithms and evaluation results are explained in [5].

**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. To assist, we automatically generate a suggested list for mapping candidates in the chosen forms. 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.

## 4 User Study

In this section, we present a summarised version of our user study results. A full version is available at [1]. The study included 15 participants. The main 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 processes and ii) end users find the automated execution of a form-based process convenient.

The task scenario was based on a university student activity grant application process. The participants were given two paper forms[5] and asked to observe the forms and the process instructions. Then, using our tool, the participants were asked to execute form annotation and process modelling tasks as if they were the form owners. 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. For the process modelling task, the participants used the two form services to create an application process.

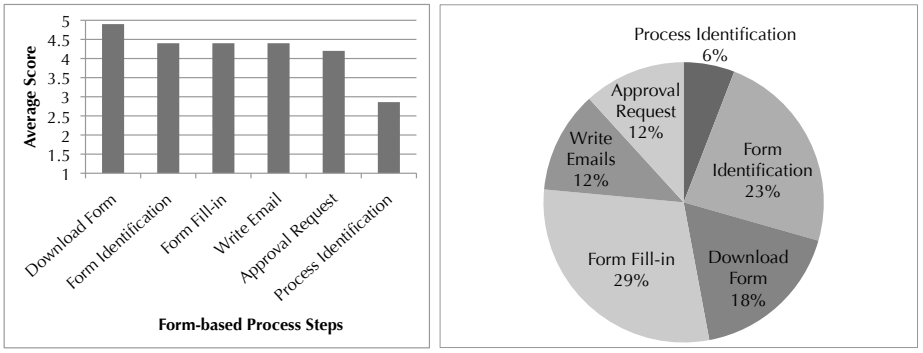
We categorised 7 participants into 'experienced users' and 8 participants into 'novice users'. All fifteen participants were able to use the tool and complete the given set of tasks designed for the form owner's role – without any extra help, regardless of their respective category. The questionnaire results on the tasks showed the tool was easy to use and intuitive (scoring well over 4 in a 5-point scale in all questions). 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.

All participants were able to complete the tasks given for the second role, form end user. Figure 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. 2(b)).

## 5 Conclusion and Future Work

In this paper, we have presented a pragmatic approach for automating form-based processes. In our approach, the form-based process model and execution

[5] www.arc.unsw.edu.au/get-involved/clubs-and-societies/club-forms-and-policy




(a) Process execution steps (end user experience)

(b) Favourite features

**Fig. 2.** Form annotation user study evaluation

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. So far 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. Our preliminary evaluation revealed that form annotation approach for automating form-based processes are applicable to people with little or no technical background, and that automated form-based processes significantly improved the overall user experience especially for form fill-in, form identification and form download tasks.

In future work, we plan to explore the process data management issue 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. We will improve the tooling features including more complex input fields mapping (e.g., concatenating two strings or manipulating dates) and attaching files to the forms.

## References

1. Kim, S.W., Paik, H.Y., Weber, I.: Automating Form-based Processes through Annotation. Technical Report 201221, School of Computer Science and Engineering, The University of New South Wales, `ftp://ftp.cse.unsw.edu.au/pub/doc/papers/UNSW/201221.pdf` (2012)
2. Weber, I., Paik, H.Y., Benatallah, B., Gong, Z., Zheng, L., Vorwerk, C.: FormSys: Form-processing Web Services. In: WWW '10. (2010) pp. 1313–1316
3. Cattuto, C., Loreto, V., Pietronero, L.: Semiotic Dynamics and Collaborative Tagging. Proceedings of the National Academy of Sciences **104** (2007) pp. 1461–1464
4. Golder, S.A.: Usage Patterns of Collaborative Tagging Systems. Journal of Information Science **32** (2006) pp. 198–208
5. Kim, S.W.: Form annotation framework for long tail process automation. In: Workshop on User-Focused Service Engineering, Consumption and Aggregation, (to be published in 2012) (2011)