Personal Process Management: Design and Execution for End-Users

Ingo Weber Hye-Young Paik Boualem Benatallah
Corren Vorwerk Zifei Gong Liangliang Zheng
Sung Wook Kim

{ingo.weber | hpaik | boualem}@cse.unsw.edu.au

Technical Report
UNSW-CSE-TR-1018
September 2010

THE UNIVERSITY OF
NEW SOUTH WALES

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

Abstract

In many cases, it is not cost effective to automate given business processes. Those business processes often affect a small number of people and/or change frequently. In this report, we present a novel approach for enabling end-users to model and deploy processes they encounter in their daily work. We herein describe the current status of our research and prototype development on personal process management.

In our approach the processes are modelled exclusively from the viewpoint of a single user, and hence avoid many complicated constructs. Therefore, the modelling can rely on simple process representations, which can be as easily understood as a cooking recipe or an audio playlist. The simplicity is achieved by allowing only few activity types in the process: filling forms and manual tasks. The process models can be translated to an executable format and be deployed, including an automatically generated Web interface for user interaction.

1 Introduction

A business process is "a set of logically related tasks performed to achieve a defined business outcome for a particular customer or market" [1]. Examples of business processes are hiring a new employee or ordering goods from a supplier. Business process management (BPM) refers to a management discipline as well as a broad category of software suites that automate, improve, and optimize business processes across the full range of process activity [2]. Business processes with a well-defined structure and high degree of repetition provide the highest potential gains from full automation [3] using BPM.

Despite the success in this area, the reality is that today many processes are in fact *not* automated. First, among other reasons, BPM products are not suitably equipped to deal with processes that are ad-hoc and dependent on heavy human interactions [4]. Second, there are costs and high skills involved in implementing automated processes. This affects primarily the "long tail of processes" [5], i.e. processes that are less structured, or that do not affect many people uniformly, or that are not considered critical to an organization: those are rarely automated. In addition, according to [6], among the organisations who use BPM software suites only 12% choose to use BPM automation components. One of the consequences of this state is that still today organisations rely on templates and paper-based forms to manage the long tail processes.

This work is focused on *personal processes*, i.e., business processes as experienced and described by a single person. By restricting the scope of our work, we can provide meaningful support while reducing the primitives to model these processes. For example, one person hardly ever pursues multiple tasks in parallel. Another area that addresses long-tail demands in service composition is centred around *mashups*. However, the focus of current mashups is different from ours: data harvesting and visualization, composition of existing data and user interfaces (UIs), and altered views or UIs for existing services are common patterns in (popular) mashups [7, 8]. While this may facilitate certain processes, we believe it does not straight-forwardly apply to personal process modelling and execution: the predominant composition paradigm in mashups are event-based synchronization [9], not process flow.

In manual personal processes, end-users often fill the same information into multiple forms redundantly. This is where our approach comes into play: with our tool, end-users can automate the processes around some of their daily tasks themselves. The starting point is a set of artefacts, like forms. By using some of our earlier work [10], we can automatically create Web services from these artefacts. These Web services then can be used as activities in our personal process management approach. The prototypical part of the solution presented in this paper is a light-weight Web-based tool where end-users can manage their personal processes. In fact, our prototype is a specialized and radically simplified version of a process modelling environment: the processes are modelled using simple process representation, either textual or visual, and are subsequently translated to BPEL[1] for deployment and execution. Naturally, by way of specialization, we restrict the set of processes that can be modelled. The exact restrictions and their ramifications are discussed in detail in the body of this

[1]While any execution language with the appropriate expressive power could be used, we chose BPEL here for two reasons: (i) it is a widely accepted format to express Web service orchestrations, and (ii) the availability of a mature tool with an end-user runtime front-end.

paper. After evaluating the approach with use cases, we believe this trade-off is valid: it allows the simplification that makes the approach applicable for end-users, while not being overly restrictive in terms of expressivity.

The contributions of this paper are the following:

- an approach and architecture for enabling end-user design and execution of personal processes.
- a conceptual solution comprising: a language for capturing personal processes; an approach for determining the required input and the produced output for a given personal process; and a dynamically generated data store to re-use available static data about a user.
- a tool implementing the above as a highly interactive Web application, and a preliminary evaluation based on two use cases.

The paper is structured as follows: We present our approach in Section 2. Then we discuss our technique for process flow modelling and data handling in Section 3. This is followed by the architecture and implementation in Section 4 and the evaluation in Section 5. Finally, we discuss related work in Section 6, before concluding the paper with Section 7.

2 Approach to Personal Process Management

We aim at enabling end-users to capture every-day processes which are so far not automated from their personal perspective. The goal is to achieve a partial automation of these personal processes, such that ideally the same data never has to be entered more than once. However, the automation is only from the user's perspective, using the same artefacts the user usually handles manually. As such, the goal is by no means to achieve heavy-weight integration of large-scale information systems. The assumption is that the end-user does *not* have deep technical knowledge.

In order to achieve this goal, we limit the modelling primitives from which the user can choose. Firstly, the control flow is limited to sequential processes and conditional execution of subprocesses, concepts with which non-IT-professionals are often familiar. Secondly, the available activities are limited to manual tasks[1] and a set of tasks that can be executed automatically. The latter set is extensible, and currently covers filling PDF forms. The artefacts used in these activities (i.e., the PDF forms) have to be provided to the tool before or during process modelling. Once the process has been designed, the user can specify data mappings, i.e., which of the data fields used in the activities will have the same content. Finally, the process can be converted to an executable notation and deployed to an execution environment.

We now go into more detail on each of these aspects.

[1] We here refer to a manual task in the sense of a reminder for the user to perform a certain task, and to inform the system once it has been completed. Only then the system continues the execution of the process.

2.1 Automating Form and Template Handling with FormSys

In earlier work [10], we investigated how to make PDF forms programmatically accessible. By PDF forms we refer to Adobe PDF's sub-standard *AcroForm*, which features editable fields [11]. This resulted in the tool *FormSys*, which has one feature of high relevance for the work here: an end-user can upload a PDF form, specify and rename the fields, and create a Web service for filling the fields. The input message of this service accepts data for the fields, and when invoked the service creates an instance of the template where the fields are filled with the data provided in the message. For instance, the standard form for requesting a driver license in Queensland, Australia, has fields for given names, family name, postal address, residential address, etc. As an AcroForm, these fields have internal names. Our tool uses these names to construct an XML schema for all fields in the forms, as well as a WSDL document where this schema type is the input of the *fillForm* operation. The tool then creates a Web service implementation for this WSDL document. An invocation of the Web service returns the URL to a copy of the original PDF form, where the fields have the values given by the input message from the Web service invocation.

Hence, end-users can actually create Web services handling some types of their every-day-artefacts themselves, using tools they are familiar with (such as word processors and Adobe Acrobat), in addition to the FormSys Web application. However, in this prior work we found that consuming these Web services is still a job for a developer. That is, once the Web service is deployed, the end-user usually is not skilled to consume it: the interface then is given by a WSDL document. With the work presented here, the end-users can in contrast design processes using these services themselves and deploy them for execution.

2.2 End-user Process Modelling

From analysing a number of use cases for personal processes, and from implementing these processes in a traditional way, we derived the following main requirement: *the process modelling language needs to be simple enough that it can be understood by non-technical users.* We found that expressing sequences of activities plus conditional execution are sufficient in terms of expressing control flow. Based on this observation, we designed a conceptual language for personal process modelling, as well as two representations of it. These are described below; the exact limitations of the control flow aspect are discussed in Section 3.2.

On the one hand, we propose a text-based process representation similar to the snippet shown in Fig. 2.1, as well as a visual representation, discussed in the next section. Both should be sufficient and easier to understand for end-users than traditional modelling techniques. To support our argument, compare the process in Fig. 2.1 with the BPMN diagram for the same process, shown in Fig. 2.2. This BPMN diagram is in fact the basis for an implementation of this process in the Intalio BPM suite[2]. More details are discussed in the next section.

[2]In this work we used the community edition of Intalio|Designer and Intalio|Server, both in version 6.0.1, from `http://www.intalio.com`

```
Fill in form F3000 (Driver Licence Application / Renewal).
If the driver has a medical condition, then
   fill in form F4355 (Medical Condition Notification) and
   fill in form F3195 (Private and Commercial Vehicle Driver's
        Health Assessment) and
   fill in form F3712 (Medical Certificate for Motor Vehicle
        Driver).
If a statement of residence is required, then
   fill in form F4208 (Queensland Residency Declaration).
If a Multi-Class Combination license is requested, then
   fill in form F3272 (Multi-Combination Driving Experience
        Declaration).
```

Figure 2.1: Targeted textual process representation: requesting a driver licence in Queensland. The applicant always has to fill in form F3000, and up to five more forms depending on whether certain conditions apply to him/her.

2.3 Data Classification

From the use cases we concluded that data in forms and templates falls into one of three categories, with respect to personal processes: user-static, process-static, and process-instance-specific data.

- *user-static:* this is the type of information in a form or template that rarely changes (i.e., is usually stable) between process instances for the same person. For example, in a travel approval and reimbursement process, the name, title and address of 'John Smith', the traveller, are not likely to change from one trip (i.e., an instance) to the next.
  In our approach, we want to support this kind of information by providing a *transparent data store*. That is, when the information is entered the first time, it is stored in a database. For subsequent process executions, the data is retrieved from the database and reviewed by the user. If any changes are made, the data store is updated. If applied correctly, this transparent data store achieves that the user does not have to enter the same data more than once.

- *process-static:* this type of information rarely changes between instances of the process, regardless of person executing them. For example, in a travel approval and reimbursement process for a certain department, the head of the department rarely changes, while certain fields must be left blank.
  This kind of information is supported by allowing the process designer to assign static values to data fields. Hence, the user starting an instance of the process is never asked to enter this data.

- *process-instance-specific:* this is the type of information that changes from one instance to another, even for the same person. For example, the destination and dates of the trip are specific to the instance of a travel process, but change from one instance to the next.
  In our approach, this information is usually entered at the beginning of

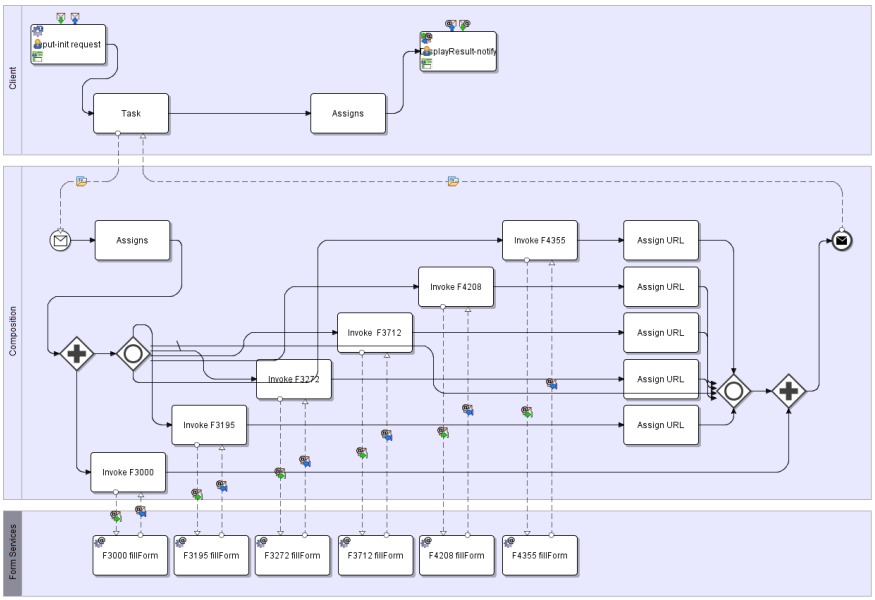

Figure 2.2: The same process as in Fig. 2.1 represented in BPMN (in the Intalio process modeling tool[2]).

the process by the user. One possible extension to our current approach would be to allow some of the information to be entered only when needed. We did not investigate how this can be done in a user-friendly way yet. A specific subtype of information is *process-step-specific*, i.e., information which is only required in a single step of a process, as opposed to being used by multiple steps of the process. For example, bank details are only required for the reimbursement step in a travel process. In contrast, the travel dates are carried over throughout each travel approval and reimbursement process instance. However, no specific handling of this type of data is required in our use cases.

Data fields that are required for more than one process step can be mapped to one another in our approach. For instance, the *surname* in travel approval is the same as *last name* in travel reimbursement. In order to determine the necessary input for the process, our solution combines all inputs for the various steps and provides a consolidated input format to the process. That is, the input message that triggers an instance of the travel process will only ask for *surname* or *last name*, but not both. To allow end-users to provide this input and start a process instance, a Web interface can be created automatically. More details are given in the next section.

3 Modelling Personal Processes

Our goal to automatically produce executable processes will yield the following benefits: (i) end-users without knowledge of the process can just use the implementation, rather than having to learn the process and surrounding policies;

(ii) user-static data can be kept in the transparent data store, so it has only to be entered when it is first used; and (iii) process-instance-specific data that is used in multiple process steps has to be provided only once to the instance, rather than at each step where it is used.

In this section we describe our conceptual solution in more detail. This solution achieves the above benefits with personal process modelling and execution, and is to be operated by end-users throughout.

3.1 Control Flow

As stated, our approach targets creating a simple language to capture personal process models, alongside two representations: a textual one similar to that shown in Fig. 2.1, and a visual representation, shown in Section 4.2. Our modelling method is set up to handle an easily extensible set of activity types. Currently, the following activity types are supported:

- **Manual task:** a named activity with a description, that will block the execution of a process until the user signals its completion.
- **Filling in a PDF form** (via FormSys, cf. Section 2.1): the data fields in a PDF form can be filled from the process; the resulting PDF can be emailed to a given set of addresses, or is made available at some URL.

The control flow primitives available to the user, as derived from our use case investigation, are: sequential ordering, and conditional execution of a sequence of steps. Sequential ordering is simply expressed by the order of the activities in the process' textual model. Conditional execution is expressed as "if `condition` then ...". Conditions can be entered as free text, in which case they become boolean inputs to the process, or as a statement over the values of fields of artefacts used elsewhere in the process.

3.2 Discussion of the Control Flow Expressiveness

The expressive power of process modelling approaches is often investigated, which can be done by analyzing to which degree the *workflow patterns* are supported in a given approach [12]. Generally, stronger expressive power is regarded as a positive feature of a modelling approach. However, already in their seminal paper [12], van der Aalst et al. discuss the issue of suitability, i.e., that the modelling approach needs to be well suited for the problem to which it is applied. The trade-off chosen here is the attempt to achieve this suitability: achieving the simplicity required for end-user process modelling, without sacrificing necessary expressive power. In the following we discuss which patterns are supported by our approach, and which patterns are less relevant given the scope and target of our approach.

Out of the basic workflow patterns we do not support parallel split and synchronization. This is not necessary for two reasons: (i) users rarely do any two tasks concurrently; and (ii) the automated activities listed above execute in split-seconds, and parallel execution would yield very little measurable advantage. Exclusive split is supported only to a degree, in that process branches can be executed or skipped. That means that executing A or B needs to be handled with two separate conditional branches. However, this construct is in

our opinion more naturally suited to the use cases, and easier to understand. Since the conditions are by default interpreted as independent, this implements the multi-choice pattern. The more complicated split/join patterns are even hard to understand at first for IT professionals, and are clearly out of scope for our purposes.

Our execution semantics assume implicit termination (pattern 12 in [12], i.e., once all activities are completed, the process instance terminates.

Loops or multiple instantiation are rarely needed in the scenarios we consider: activities don't fail, and manual tasks can be repeated until successful completion without the process engine asking the user to do so; there are rarely personal processes that relate to multiple instances of some object, e.g., performing an activity for each line item in a purchase order; finally, going back in a process instance to correct something and then execute a number of steps is interesting, but left for future work (cf. Section 7). Interleaving sequences (pattern 17) can only occur within parallel execution. Deferred choice (pattern 18) requires events, which we see as too advanced for end-users at this point.

Interleaved parallel routing (pattern 19), i.e., the execution of a set of activities in an arbitrary order, could be relevant for some scenarios with a larger number of manual tasks. However, this is usually not where our approach adds value: a person is usually well capable to schedule tasks in this manner by herself; a simple checklist appears to be more appropriate such situations than a workflow execution environment. Including interactive checklists in our approach could be done in the future; meanwhile this can be done in manual tasks which state a list of subtasks, and where the user signals completion only to all of the subtasks by marking the whole task as completed.

Cancellation of a process instance can be handled by the underlying workflow system. In our implementation (cf. Section 4) we rely on Intalio's process server[2], which supports cancellation through an administration console.

Inter-workflow synchronization (patterns 23-26) is obviously not required for personal processes with stateless activities.

In [13], zur Muehlen and Recker investigated the usage of the BPMN modelling elements in 120 BPMN models. Among their analyses is the occurrence frequency of BPMN constructs in these models. In the ranking of occurrence frequency, the first construct that could be supported[1] by our approach but is not, is "Parallel Fork/Join" on rank 10; it is used in less than 40% of the models. In comparison, data-based exclusive split/join is used in between 20% to 80% of the models, depending on the source of the diagram. Although BPMN is used for many purposes, we see the fact that most of the commonly used constructs in general-purpose process modelling are supported in our approach as an indication that the expressive power is likely to be sufficient for our limited purposes.

Finally, we note that the PICTURE approach [14, 15], discussed in Section 6, has been successful in practice, in a related setting with an expressive power slightly below ours, from the viewpoint of the single process participant.

[1] Pools are of limited use in personal process models, and message flow is implicit in our approach.

3.3 Data Flow

The above form-filling services all have defined input data, in contrast to manual tasks which have only a process-static description. For the form-filling services, the process model needs to express which information is user-static, process-static, and which data is entered at the start of the process. For the latter part, the model needs to specify which of the data fields for different process steps will contain the same information (cf. Section 2.3). The latter information is called *data mapping* (or just *mapping*) in our approach. Note that, in contrast to many other BPM systems, we here only deal with the mapping of the inputs of one service to the inputs of other services. This is due to the fact that the form-filling services considered here only require input from the user or the transparent data store, since the output of the form-filling services is only the URL to the filled form. When relaxing the assumption that all services are "input-only" to more general classes of inputs, the approach also needs to be able to handle output-input data mappings, i.e., the output of one service becoming the input to another. However, this will be subject only to future work.

The data mapping facilities in our approach can express simple equality (e.g., `last-name = surname`), fixed assignment (`employer = "UNSW"` or `funding-agency = ""`) and concatenation (`given-names + surname = full-name`). It could be extended to support various types of complex mapping rules (e.g., `if X is empty, then Y = X`). However, in order to retain the focus of this work on simplicity, we decided against more complicated cases.

There are several ways to make this knowledge available to the process model: the information can be provided manually by the user; the mapping of fields to other fields can be supported by tools [16, 17]; or the information about the nature of the data could be derived from analysis of other processes or from a knowledge base. We acknowledge that end-users will not find it easy to specify the data mapping manually, and that there may be significant value in additional support, e.g., by automatically suggesting a mapping, but leave the detailed investigation of this topic for future work. Here we assume the mappings are given, and our implementation (cf. Section 4.2) currently only supports the manual mapping.

The input data that should be provided to each process instance is given by the data required by its activities. That is, the union of all data fields of the input messages to the steps minus the fields which are given by the mappings. In our approach, this data set is determined automatically:

- Fixed assignment to a set of fields means those fields are fully specified.
- Equality of two or more fields leads to only one of the fields being required as input.
- User-static fields will still be input fields, but will be pre-filled with information from the transparent data store.
- Concatenation of a set of fields equalling another set of fields means that only the former set is required.

Similarly, the output of the process is simply the union of all the information provided by the activities – i.e., the URLs of all PDFs, filled by the process in-

stance calling (a subset of) the form-filling services. Implementing this strategy for input and output derivation is straight-forward, given the process model.

3.4 PPML – a Language for Personal Process Models

As part of our approach, we suggest the *Personal Process Modelling Language (PPML).* We represent this language with a human-readable syntax. The grammar for it is shown in Fig. 3.1. For certain implementation purposes, we also created an XML representation, which closely corresponds to the human-readable representation. The XML representation is e.g., better suited for translations from or to other XML-based languages such as BPEL.

```
process ::= PROCESS name START processsteps MAPPINGS mappings
            DATACONFIGURATION dataconfiguration END

name ::= letter | letter characters
letter ::= A | B | C | ... | Z | 'a' | 'b' | 'c' | ... | 'z'
characters ::= characters character | /* empty */
character ::= letter | number | '_'
number ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

processsteps ::= processsteps processstep | /* empty */
processstep ::= activity | ifthen
activities ::= activities activity | /* empty */
activity ::= fillform | manualtask
fillform ::= FILLFORM object
manualtask ::= MANUALTASK name DESCRIPTION characters
ifthen ::= IF key operator value THEN activities FI

object ::= name
group ::= name
operator ::= 'eq' | 'ne' | 'gt' | 'lt'
key ::= name
value ::= characters

mappings ::= mappings mapping SEMICOLON | /* empty */
mapping ::= parts
parts ::= '(' concatparts ')' '==' parts | parts '==' part | part
concatparts ::= concatparts AND part | part
part ::= key | object '.' key | object '.' group '.' key
dataconfiguration ::= USERSTATICDATA datalist
                      FIXEDDATA datavaluelist
datalist ::= datalist data | /* empty */
datavaluelist ::= datavaluelist data = '('characters')'
                  | /* empty */
data ::= key | object | object '.' key | object '.' group
         | object '.' group '.' key
```

Figure 3.1: PPML grammar in Backus-Naur Form

A process in PPML has a name, a sequence of process steps, a number of data mappings and a data configuration. A process step can be an activity or an if-then construct. The then-part of an if-then can, in turn, contain a set of activities. The available activities are the ones mentioned in Section 3.1: manual task and filling a PDF form. At this point, the language can be easily extended with other activities. The order of the process steps and activities expresses the control flow order in the process.

The data mappings are stored as a set of mapping rows, which each specify that a set of parts is equal to one another. A part may just refer simply to data field's name (in the language called a *key*), or to an object (like a whole PDF form), or a group of an object (e.g., all fields in the group "personal details"). There may be one concat-part per row, i.e., where the values of multiple fields are concatenated in a mapping. Since the activities all only consume data and produce status output, there is no need to specify source or target of a mapping: the source for all mapped fields is user input, and all the parts set to be equal are the targets. The mappings reduce the number of required input fields. While a richer set of mapping techniques may be desired in some scenarios, we do believe that the simplicity here is the key to achieve end-user engagement.

Finally, some data fields or sets of fields may be marked as being user-static or set to a fixed (possibly empty) value. For the fixed value assignment, the field is listed together with the respective fixed value it is given. Fields can be declared user-static by listing them in the respective data configuration part. The configuration is interpreted hierarchically, i.e., a setting of a higher-level element is applied to a lower-level element unless specified otherwise. The default setting is process-instance-specific, which applies if a field is not listed as user-static or process-static. User-static fields are added to the transparent data store.

4 Architecture and Implementation

In order to demonstrate the feasibility of our approach, we specified an architecture and implemented a proof-of-concept tool. The prototype is a significant extension to our earlier tool [10], hence called *FormSys Process Designer*. A demonstration paper about this implementation has been accepted for publication[18]. Also, a screencast video is available[1]. The architecture and implementation are described in this section. We evaluated our work based on this tool, as discussed in the next section.

4.1 Architecture

The architecture design for our solution is shown in Fig. 4.1. There are two user roles: process owner and end-user. In terms of skills required, our goal is that the process owner can be a non-technical end-user as well. However, he can share the deployed processes with peers or other users, so that they can make use of the implementation as well (role: End-user).

The process owner designs the process model in a Web front-end. This front-end interacts with FormSys Process Designer's core component. Through that, the access is achieved to the database for process models and to FormSys

[1] `http://www.cse.unsw.edu.au/~FormSys/process/`

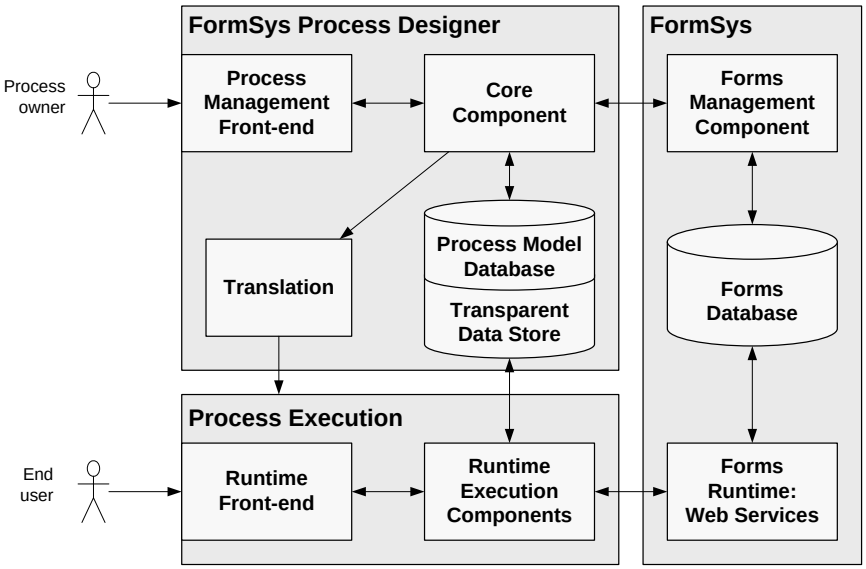

Figure 4.1: The architecture of FormSys Process Designer and the surrounding components.

for generating and accessing form-filling Web services used in processes. Also, the transparent data store for static user information is prepared for runtime. Through the front-end, the user can trigger the translation to and deployment of an executable process model in an execution environment. Due to the prevalent availability of such environments, we use existing third-party tools for the runtime. Once deployed, end-users can start instances of the processes, using Web front-ends of the execution environment. The process execution makes use of the transparent data store for users' static information, as well as FormSys Web services, offering the form-filling functionality described in the previous sections.

4.2 FormSys Process Designer: a Tool for Modelling Personal Processes

The implementation of the above architecture is done in PHP with the Symfony framework[2] and jQuery[3]. As a runtime environment, we use Intalio|Server[2], since it offers a BPEL-compliant execution engine as well as a Web interface for user interaction.

The most original component in our implementation concerns the process design, and we focus our description on it in the following. Screenshots of the graphical user interface (GUI) can be seen in Figures 4.2-4.4. This Web-based GUI allows the user to specify valid PPML processes using a dynamic text-based (Fig. 4.2) or a visual (Fig. 4.3) modelling methodology.

In the *textual editing mode* (Fig. 4.2), a process model's steps offer a few functions by hovering with the mouse over the "+" symbol on the right-hand

[2] `http://www.symfony-project.org/`
[3] `http://jquery.com`

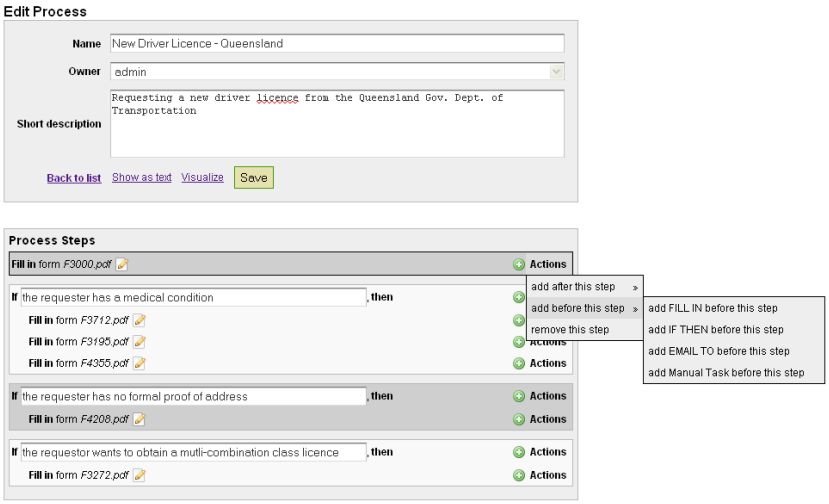

Figure 4.2: Textual process modelling in the FormSys Process Designer.

side of a line. This opens a context menu where the chosen step can be removed or new steps can be added before or after it. The settings and values detailing process steps can be edited by clicking on the respective value. That opens up a drop-down box or a text field, depending on the field type. The mappings can be edited in a similar fashion. Through the context menus, the user's choices are limited to valid changes in PPML, i.e., a valid model can only be changed so that the resulting model is valid again.

In the *visual editing mode* (Fig. 4.3), all activities are represented with a thumbnail image or an icon. Form-filling services are represented by a thumbnail of the form's first page. The list of available activities is presented in a *content flow* (the black area in Fig. 4.3), similar to cover flows in popular music library programs. The process designer can drag-and-drop activities from the content flow to the *Process Steps Workspace* (the grey area in Fig. 4.3). Here the control flow of the process is expressed with arrows for sequencing (left to right, top to bottom); and dark-grey boxes for conditional execution, e.g., "if a residence statement is required", the form contained in the respective box will be executed. Re-ordering steps can be done by dragging-and-dropping them to a new position, including into and out of boxes with conditions.

Besides specifying the control flow, the process designer needs to define the data flow. As explained in the previous section, this means mapping input fields from different forms to each other (equality or concatenation), declaring static values for fields, or declaring fields as user-static. While the latter two features are not implemented in the prototype yet, the former can be done textually, via drop-down menu selections (not shown), or visually (cf. Fig. 4.4). The user can define mappings between individual fields of forms by first dragging two forms from the *Process Steps Workspace* to the *Mapping Workspace* (in Fig. 4.4 this has been done for Form 3000 and 4355 from the Queensland Driver License process), then selecting "Add new mapping", and then clicking on the fields in the forms that are mapped to one another. This will colour the borders around those fields with the colour of the mapping.

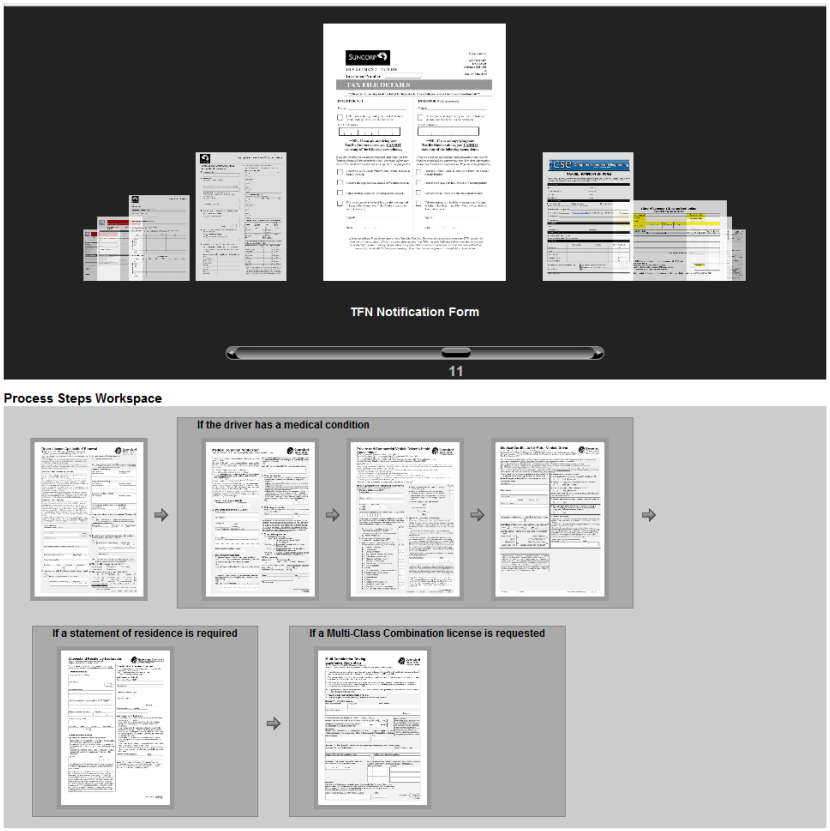

Figure 4.3: Visual process modelling in the FormSys Process Designer.

The list of declared mappings is shown as a list of squares in their colours below the Add and Delete buttons on the top of the Mapping Workspace, and the square of the currently active mapping has a white fill. This list remains stable when replacing one or both of the forms in the Mapping Workspace. Hence, the user can specify mappings of, e.g., the `last_name` fields of more than two forms. When selecting more than one field for a mapping within one form (shown for the dark purple mapping in Fig. 4.4), this is interpreted as concatenation of those two fields. This is indicated by the numbers shown inside the fields: the fields labelled "1" and "2" in the left-hand form are concatenated to a field labelled "1+2" in the right-hand form.

In addition to the model editor, there are overview pages for listing all available processes, as well as allowing user administration and role assignment. Processes can also be viewed, instead of edited. This way, a process is presented without the editing symbols, which results in two views: a visual view similar to the grey area in Fig. 4.3, and a textual one that resembles Fig. 2.1 rather closely.

Process translation to BPEL can be triggered from the overview list. This creates both a BPEL file and WSDL file containing the consolidated input and output message types, as described in Section 3.3. For process execution we use the above-mentioned Intalio|Server. Hence, in addition to the BPEL and

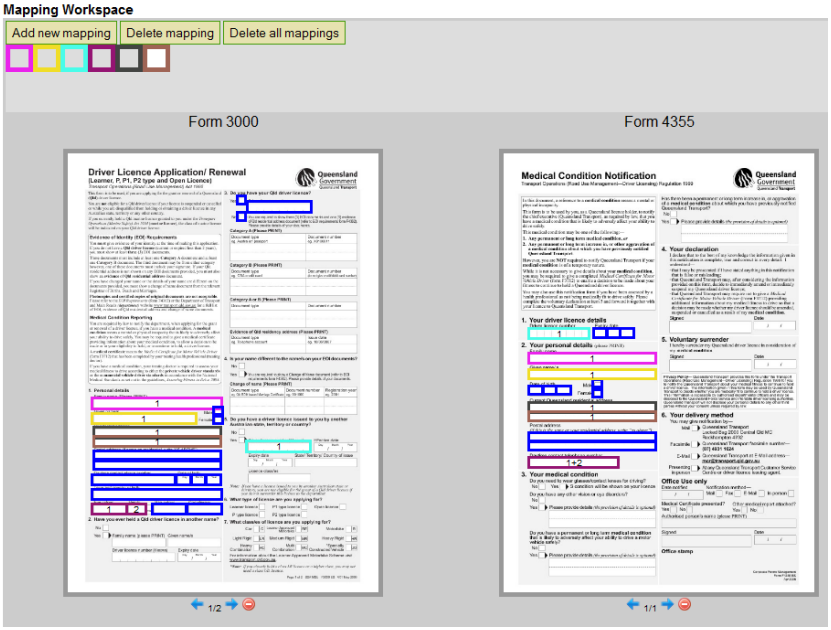

Figure 4.4: Visual data mapping in the FormSys Process Designer.

WSDL files, further artefacts are required: a deployment descriptor, an SVG graph of the process model in BPMN, and user interface pages (AJAX Web forms) for input (triggering a process instance) and output (viewing the results of the instance), and more. All these artifacts are automatically created when the translation is requested. The result is a deployable package of the files, which can be uploaded to an Intalio|Server. A screenshot of an automatically created input Web form is shown in Fig. 5.1. It should be noted that, while usable as such, in real-world usages most likely this form would most likely only form a starting point, and the process designer would be allowed to change its design. Since the focus of this work is not on UI generation, we only implemented this as a proof-of-concept solution.

5 Evaluation

After our analysis and implementation in the previous sections we provide a first, small evaluation our work in this section with two case studies. Examples from both case studies have been used as examples throughout the paper already. The first case study is about applying for a driver licence in Queensland. The second is the travel request and reimbursement process of the School of Computer Science and Engineering (CSE) of the University of New South Wales (UNSW). Both case studies are real-world scenarios from large organizations. Nevertheless, they are still form-based and to date no automation is offered to end-users that are involved in these processes. In the following, we explain the processes, and how their implementation with FormSys and the FormSys Process Designer helps in easing the burden for the end-user. Subsequently we discuss our findings.

5.1 Case study 1: Queensland Drivers License Application

The license request or renewal process involves up to six forms an applicant has to fill in. Which forms she has to fill in is described in a number of Web pages and has to be understood before filling any form. As an example, an additional form has to be filled if the the applicant has a medical condition. Furthermore, all forms request – among other things – the same personal information like name, address, date of birth and more. We analysed the process starting from the main guideline for citizens[1] and collected all forms. We uploaded the forms to FormSys and deployed corresponding Web services for each of them. In the FormSys Process Designer, we then modelled the process (cf. Figures 4.2 and 4.3), mapped the data fields (cf. Fig. 4.4) and deployed the resulting process in Intalio|Server[2]. The end-user (citizen) can start the process through a Web page (cf. Fig. 5.1), where she enters the repetitive fields only once before starting an instance of the process. Additionally she answers the questions derived from the conditions of the process, according to which it will be determined which forms have to be filled. Depending on this information, the process instance fills all forms or a just a subset of them, and returns a message linking to the URLs where these forms are available. The user can then print the forms and follow their instructions on where to submit them.

Specifying the control flow of this process was simple, once the instructions on the website were understood. The data flow mapping is not hard to do, as the forms largely follow a common naming scheme. However, quite a few fields had to be mapped, where automatic mapping would have likely provided good results.

5.2 Case study 2: CSE (UNSW) Travel Request and Reimbursement Process

This case study is about the travel request and reimbursement of CSE, UNSW. Ignoring for this study the special cases of using a private car, getting a cash in advance etc., a traveler has to fill three forms for each trip he wants to make:

- a request for a trip and total cost estimate;
- a travel diary, listing the activities during the trip; and
- a reimbursement request after the trip is finished.

We modelled the process using FormSys Process Designer, as shown in Fig. 5.2. In contrast to the other case study, this process is split in two parts. The request is done before the trip and can lead to an early termination of the process when denied. The second part is executed after the trip, and handles the reimbursement and travel dairy. After the approval, the process instance stops and waits for the manual task to be marked as completed. When the trip is finished, the traveler confirms his return by marking the manual task as completed, and gets the pre-filled reimbursement form and, depending on the nature of the trip ("if the trip is to a destination more than 100kms from the school"), also the dairy form. The data mapping for this process is partly

[1] `http://www.transport.qld.gov.au/Home/Licensing/Driver_licence/Getting_a_licence/Car/Open_licence/`

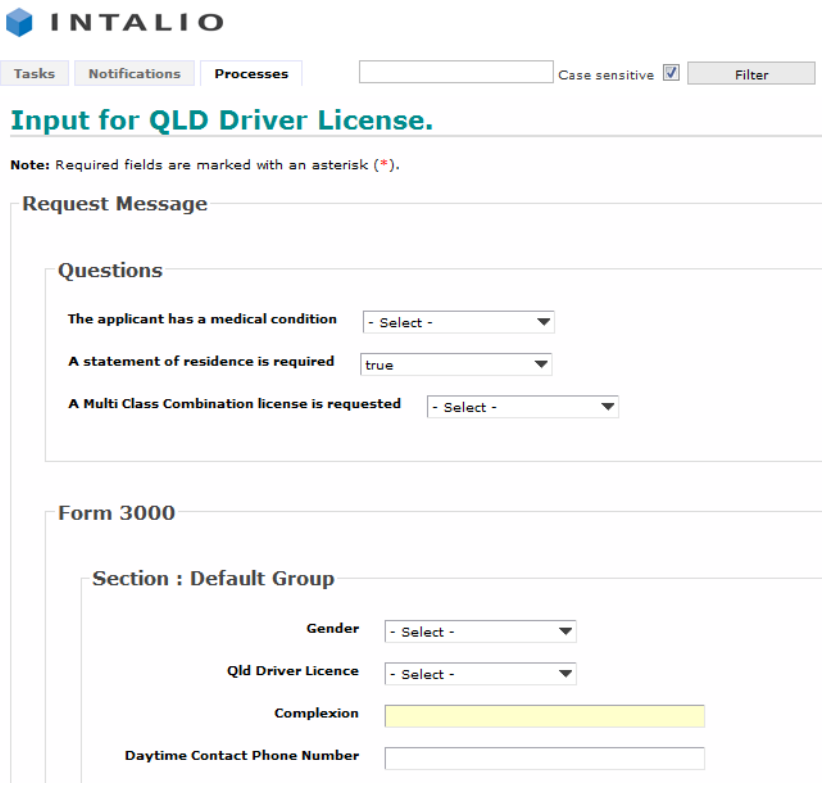

Figure 5.1: Part of the input Web form to start the Queensland Driver Licence process.

shown in Fig. 5.3; its creation required less effort than for the first case study, mainly because only three forms were involved. However, the mapping was not as straight-forward.

5.3 Findings

We now discuss our findings regarding the prototype and our case studies. We implemented these real-world cases with our tool, as a first step towards evaluating its suitability for supporting personal processes. Each process' control flow was created within a few minutes after the form-filling Web services were deployed through FormSys. From the tool side, a user familiar with assembling a music play-list should be able to design the control flow of a personal process in our tool. However, the user still needs to have an understanding of how process models relate to instances.

The more effort-intensive part is the mapping: for forms with many fields or processes with many forms, the mapping consumes a significantly larger share of time than the control flow modelling. Besides that, we think the concept behind the data mapping may be confusing to non-technical users who are not familiar with process modelling methodologies in the first place, and may require good explanation.

After finishing a process model, deployment can be done automatically. In-

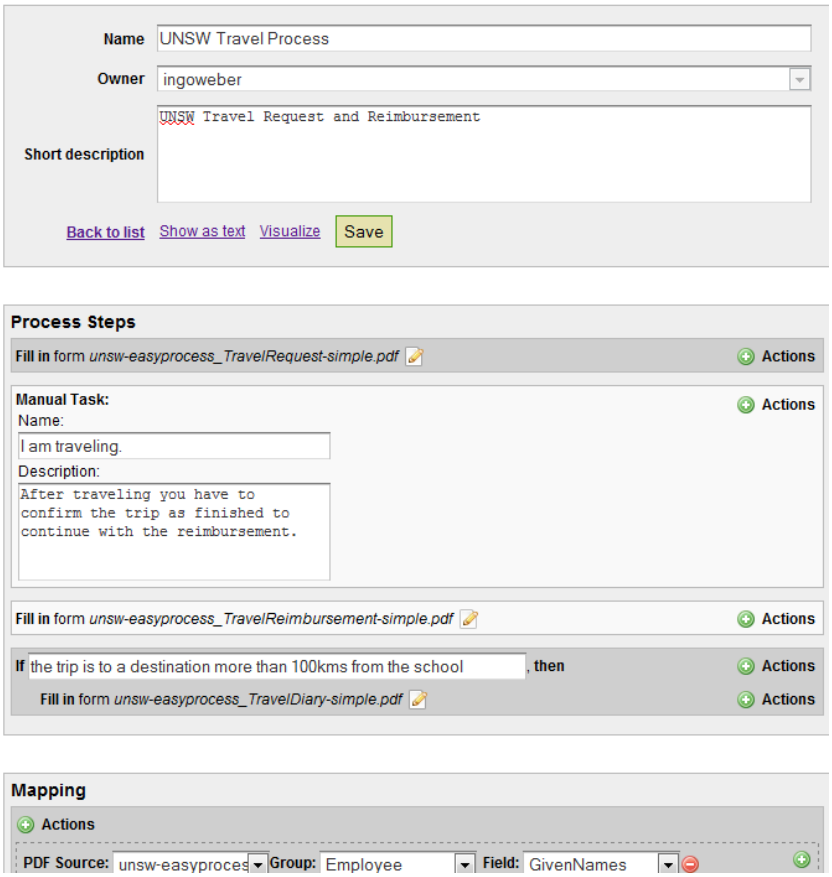

Figure 5.2: Textual process model of the UNSW travel process.

talio executed all processes without problems. However, the presented Web interface for end-user input to the processes is somewhat simplistic and offers potential for improvement.

6 Related Work

The introduction discussed the relation of this work to works on mash-ups and traditional business process execution or Web service orchestration. To our knowledge, all related end-user process modelling tools do not feature the creation of executable processes. We will discuss a number of these below.

Very recently, *collaborative business process modelling* for various types of users gained attention, e.g., [19, 20]. The relation to this work is that most of these approaches aim at collaborative discovery and re-design of processes, where non-IT-professionals and users without heavy training in process modelling are part of the collaboration. We will discuss these approaches below.

Software AG offers the social networking BPM tool *ARISalign*[1] [20]. The focus of this tool is to involve stakeholders and experts from within an organi-

[1] `http://www.arisalign.com/`

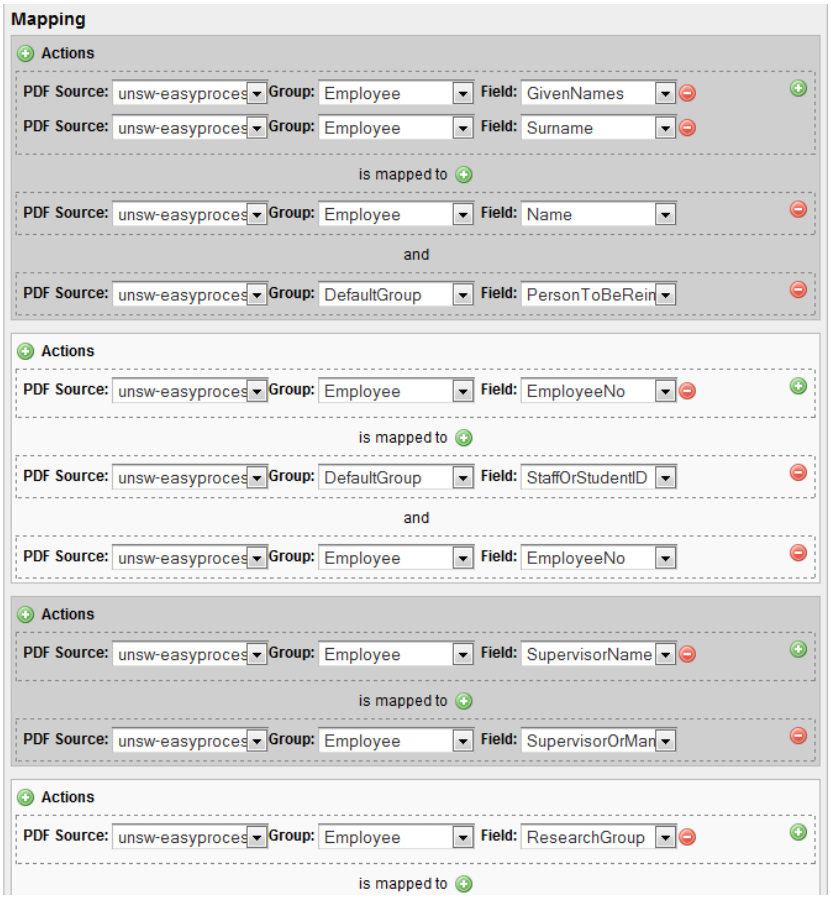

Figure 5.3: Textual data mapping for the UNSW travel process.

zation as well as outside in the process design. When the process design phase is completed, the processes can be executed. SAP's tool, *Gravity* [19], is a real-time collaborative process modelling environment and embedded in Google Wave and SAP 12sprint. There is a high-level modelling perspective (BPMN) and an executable level (proprietary notation, similar to Yahoo pipes). From the information available about the tool, the latter is primarily focused on widget-like user interfaces, but the tool also has a facility to send emails.

The company *Lombardi*[2] has a tool called Blueprint, in which users can collaboratively discover process models. The user, or group of users, starts by creating a "process map", which resembles Porter's value chains. The map consists of a sequence of high-level steps, where more detailed process steps can be added below each high-level step. The precise control flow can be designed in BPMN, and exported for further refinement to execution in Lombardi's other BPM tool, Teamworks 7. A strong feature of Blueprint is collaboration: users can share, comment, update, and be notified about changes in process models. The focus is, however, again on BPM professionals discovering or improving core processes.

[2] `http://www.lombardisoftware.com/`

A recent startup, *Signavio*[3], offers a browser-based BPMN and value chain editing tool. The basis of this was the tool Oryx [21]. Again, a focus is on collaboration, commenting, and sharing process models with others. Another feature is a user-defined dictionary, to encourage similar naming of activities, documents, and other labels. The open-source project Activiti[4] makes use of Signavio's modelling tool, so that BPMN models can be executed in the Activiti environment. In relation to our solution, the Signavio Process Editor is quite similar to Lombardi's Blueprint.

PICTURE is a domain-specific, building block-based modelling method and notation [14, 15] for public administration. Out of a research project, the company PICTURE GmbH[5] has been founded. The approach is based on a fixed, domain-specific set of modeling constructs: the building blocks. Example building blocks are "receiving a document / information", "coordinate / consult with other party", or "execute formal check". These can be named and further specified, e.g., by stating the channels through which information is retrieved, or which document is checked. Building blocks are arranged in sequences to form local subprocesses, and different subprocesses can be linked via so-called *anchors*. Conditional execution can be expressed as different possible subprocess variants. The argument for pure sequences in subprocesses is the same as in this work: a single knowledge worker is expected to perform only one task at any given time [15]. In a real-world process consulting project, the PICTURE method has been shown to offer significant higher efficiency in process discovery, e.g., a factor of five less time required in the two projects (one using eEPCs, the other PICTURE) compared in [15]. According to the same source, the reasons for the increasing efficiency are a fixed level of abstraction, a terminology with which the process participants are familiar, and the simplicity of the notation. PICTURE is a tool for discovering complete process landscapes in organizations, and not meant to design executable process models. However, we believe that we implemented the key features making process modelling simple and efficient in our approach as well, and thus hope to achieve the same advantages. The key difference to our work is that PICTURE targets capturing the processes, and does not have any features to create executable processes.

Finally, there is the tool *CoScripter* (formerly called Koala) [22, 23]. The primary focus is on personal processes in the scope of browsing and using Web applications. The user can record such browser processes, play them back, and save them on a public Wiki. Other users can then make use of saved processes. The processes are stored in a simple end-user understandable language, using natural language keywords such as " * go to <URL>" and " * click on <link>". Personal user information is stored in a *personal database* on the user's machine. In the scenarios covered by this approach, the usefulness has been demonstrated by real users in their day-to-day work lives [22]. In contrast to our approach aiming at forms, the scope of CoScripter is limited to the browser: all steps in a script need to be standard operations in a browser window. While closely related to our work in terms of the textual process representation and the end-user focus, it does not support Web service invocation or conditional execution. We plan on integrating CoScripter scripts as another type of activity in our personal process modelling environment in our future work.

[3] `http://www.signavio.com/`
[4] `http://www.activiti.org`
[5] `http://www.picture-gmbh.de`

7 Conclusion

In this report we presented the current status of our research and prototype development on personal process management. With the solution discussed, end-users can assemble executable counterparts of their form-based processes, which can significantly reduce the amount of redundant data entry required for those processes. The current status poses a first milestone, and has been briefly evaluated with two case studies. These case studies uncovered some weaknesses, which we will work on in the mid-term future, first and foremost support for semi-automatic data mapping.

More long-term future work will focus on the following: (i) including other types of services, including WSDL/SOAP, RESTful Web services, etc., at the cost of requiring a more complete solution for data flow; and (ii) more flexible execution, so that users can roll-back process instances, e.g., to change some input values, and re-execute selected steps.

Acknowledgements

This work has been supported by a grant from by the Smart Services CRC[1] under the Service Delivery Framework project.

Bibliography

[1] Davenport, T.H., Short, J.E.: The New Industrial Engineering: Information Technology and Business Process Redesign. MIT Sloan Management Review **31**(4) (1990) 11–27

[2] Richardson, C., Vollmer, K., Clair, C.L., Moore, C., Vitti, R.: Business Process Management Suites, Q3 2009 – The Need For Increased Business Agility Drives BPM Adoption. Forrester TechRadar For BP&A Pros (13 August 2009)

[3] Leymann, F., Roller, D.: Production Workflow - Concepts and Techniques. Prentice Hall (2000)

[4] Schurter, T.: BPM state of the nation 2009. bpm.com, `http://www.bpm.com/bpm-state-of-the-nation-2009.html` (2009) Accessed 25/11/2009.

[5] Oracle White Paper: State of the business process management market 2008. `http://www.oracle.com/technologies/bpm/docs/state-of-bpm-market-whitepaper.pdf`, accessed 20/11/2009 (August 2008)

[6] Wolf, C., Harmon, P.: The state of business process management. Technical report, BPTrends (June 2006) `http://www.bptrends.com/`.

[7] Ogrinz, M.: Mashup Patterns: Designs and Examples for the Modern Enterprise. Addison-Wesley Professional (March 2009)

[1] `http://www.smartservicescrc.com.au`

[8] Wong, J., Hong, J.: What do we "mashup" when we make mashups? In: WEUSE'08: 4th International Workshop on End-user Software Engineering at ICSE'08, Leipzig, Germany. (May 2008)

[9] Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding mashup development. IEEE Internet Computing **12**(5) (2008) 44–52

[10] Weber, I., Paik, H., Benatallah, B., Gong, Z., Zheng, L., Vorwerk, C.: FormSys: Form-processing web services. In: WWW'10: Proceedings of the 19th International World Wide Web Conference, Demo Track. (2010) `http://imweber.de/texte/FormSys-Form-processingWebServices--WWW2010--authors_copy.pdf`.

[11] Adobe Systems Incorporated: Acrobat Forms API Reference (2003) Technical Note No. 5181.

[12] van der Aalst, W., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases **14**(1) (2003) 5–51

[13] zur Muehlen, M., Recker, J.: How much language is enough? Theoretical and practical use of the business process modeling notation. In: CAiSE'08: 20th International Conference on Advanced Information Systems Engineering, Montpellier, France (June 2008)

[14] Becker, J., Pfeiffer, D., Räckers, M.: PICTURE - a new approach for domain-specific process modelling. In: CAiSE Forum. (2007)

[15] Becker, J., Algermissen, L., Pfeiffer, D., Räckers, M.: Bausteinbasierte Modellierung von Prozesslandschaften mit der PICTURE-Methode am Beispiel der Universitätsverwaltung Münster. Wirtschaftsinformatik **49** (2007) 267–279

[16] Do, H.H., Rahm, E.: COMA – a system for flexible combination of schema matching approaches. In: VLDB'02: 28th Intl. Conference on Very Large Databases. (2002)

[17] Drumm, C., Schmitt, M., Do, H.H., Rahm, E.: Quickmig - automatic schema matching for data migration projects. In: CIKM'07: 16th ACM Conference on Information and Knowledge Management. (2007)

[18] Weber, I., young Paik, H., Benatallah, B., Vorwerk, C., Gong, Z., Zheng, L., Kim, S.: Managing long-tail processes using FormSys. In: ICSOC'10: 8th International Conference on Service Oriented Computing, Demo Track, San Francisco, CA (December 2010)

[19] Balko, S., Dreiling, A., Fleischmann, K., Hettel, T.: Gravity – collaborative business process modelling and application development. SAP Community Network, `http://www.sdn.sap.com/irj/scn/weblogs?blog=/pub/wlg/17826` (23 February 2010)

[20] Sayer, P.: Software AG opens BPM social networking beta test. PCWorld Business Center, `http://www.pcworld.com/businesscenter/article/190592/software_ag_opens_bpm_social_networking_beta_test.html` (2 March 2010)

[21] Decker, G., Overdick, H., Weske, M.: Oryx – an open modeling platform for the BPM community. In: Demonstrations at BPM'08: 6th International Conference on Business Process Management. (2008)

[22] Leshed, G., Haber, E., Matthews, T., Lau, T.: CoScripter: Automating & sharing how-to knowledge in the enterprise. CHI Letters: Human Factors in Computing Systems **10**(1) (2008) 1719–1728

[23] Little, G., Lau, T., Cypher, A., Lin, J., Haber, E., Kandogan, E.: Koala: Capture, share, automate, personalize business processes on the web. CHI Letters: Human Factors in Computing Systems **9**(1) (2007) 943–946