Social-network-based Personal Processes

Seyed Alireza Hajimirsadeghi, Hye-Young Paik, and John Shepherd

University of New South Wales, Sydney, NSW 2052, Australia,
{seyedh,hpaik,jas}@cse.unsw.edu.au

Abstract. In this paper, we propose *Processbook*, a social-network-based management system for *personal processes* (ad hoc processes carried out to achieve a personal goal). A simple modelling interface is introduced based on ToDoLists to help users plan towards their goals. We describe how the system can capture a user's experience in managing their ToDoList and the associated personal process, how this information can be shared with other users, and how the system can use this information to recommend process strategies. We exemplify the approach by a sample administrative process inside University of New South Wales and report some preliminary evaluations of the proposed recommendation algorithms.

Keywords: Personal Process Management, Task Management Systems, Process Knowledge, Process Recommendation, Social Networks

1 Introduction

In modern society, we are frequently required to perform administrative or business processes in order to achieve our goals. Examples could be simple processes such as booking a theatre ticket, or more complicated and long lasting ones such as planning for immigration.

While the last decade has seen many of these individual processes codified via online services, there remain significant problems in discovering and integrating the very many tasks that have not yet been codified. An important aspect of the problem is that processes frequently span organisational boundaries and there are few mechanisms to carry information and outcomes from processes in one organisation to those in the next organisation. Another major factor is that it is sometimes difficult to identify precisely which organisations and which processes within those organisations are required to accomplish a stated goal.

A *personal process* is made up of a number of tasks which needs to be carried out in order to achieve a goal; a task may be as simple as one individual activity or may be as complex as a complete business process. Personal processes often require an integration of so-called "long tail" business processes from one or more organisations. The knowledge of such integration and consequently the knowledge of achieving a goal is referred to as *process knowledge* in this paper.

Our goal is to provide support for individuals to manage their personal processes. One possible approach for this would be to convert all personal processes into codified business processes, but this is neither plausible nor cost-effective

[7]. Instead, we aim to assist users in discovering the tasks they have to do to reach their goals, the order they need to do the tasks and the sets of constraints and rules they should follow in doing those tasks. This is accomplished in a context where other users have already successfully carried out the processes and have recorded the method by which they did so. Our system to support this (*Processbook*) adopts a social-based approach, aimed at non-expert users, who describe their personal processes via *ToDoList*s. *Processbook* collects, manages, merges and shares process descriptions and allows users to follow other users who are carrying out similar tasks and can recommend future steps in a given process based on how others users previously achieved the same goal.

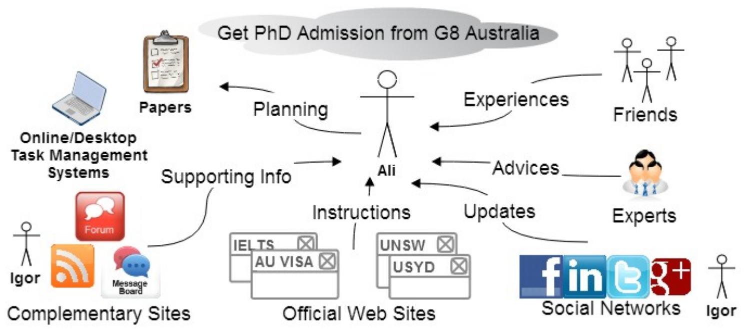
In section 2, we elaborate the problem area with examples. Section 3 describes related work in the space of personal processes and using social software in business process management. Section 4 presents the details of *Processbook* and explains how process knowledge is captured and then recommended to users. In Section 5 we investigate the implemented tool with an example in academic domain. Finally we conclude the paper in section 6 with future work.

2 Personal Process Management: A Motivating Scenario

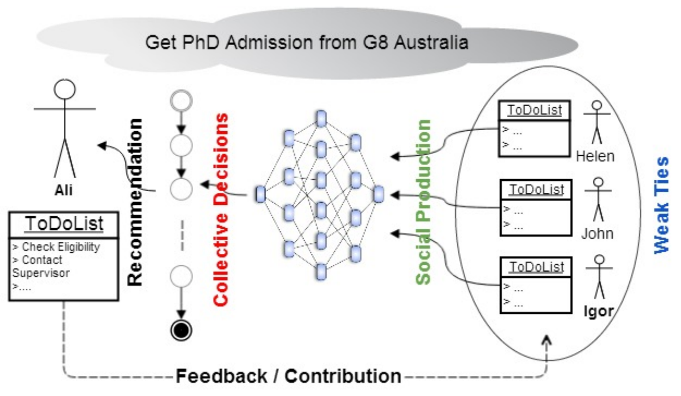
In this section, we consider the problem of carrying out personal processes via an example: Ali, a student from a non English-speaking country, wishes to study for a PhD in one of the top 8 Australian universities (known as Go8). Ali has two primary objectives: find a university that would accept him and maximise the amount of funding to assist his study. Additional constraints and preferences might include: a PhD topic in the service oriented computing area, a PhD program commencing after July 2013, etc.

Figure 1a illustrates how Ali plans to reach his goals and what sources he utilises for his purpose. He tries to identify universities that satisfy his constraints by asking friends or by searching on the Web, collects and collates information about the entry requirements and scholarship availability for each university from its official web site. He might also join relevant communities on popular social networks such as facebook and twitter to keep up with the latest news and updates about the institutions he is dealing with.

In carrying out the above, questions would arise at each stage for Ali. For example, the web site at some university might specify that a student needs to provide an undergraduate transcript and English proficiency test results, but might not mention the kind of visa that the student requires or how to obtain such a visa. Other typical questions that might arise are: what step should I take next, what do I do at each step, which organisation should I deal with, etc. To find the answers to such questions, Ali would seek answers from his friends, experts, social networks or other data available in online forums, blogs, etc. Ali uses a to-do list approach to organise his findings and to write down the tasks he has to do. He may do so by simply writing on paper, using computer-aided task management tools or registering in an online task management website where he is offered more gadgets to organise his tasks.



(a) Example of a Personal Process Management



(b) Personal Process Management in *Processbook*

Fig. 1: From Personal Involvement to Active Social Participation

Getting advice from someone experienced with the specific process would be extremely useful, but finding such an expert might be difficult. A more effective approach might be to have the process information available online, and have a system that understands both the process information and your personal situation (in terms of progress through the process), and can offer sufficient information to enable you to determine how to proceed. In practice, a number of difficult issues need to be dealt with before such a system can be realised:

Invalid, incomplete or inconsistent data: We may be faced with untrusted sources of information, or conflicting items of information, or may be given out-of-date information. Sometimes, we simply do not know certain parts of the process. In other cases, there may be hidden (or ignored) pieces of information. For example, Middle Eastern students may face a wait of up to three months in applying for a Australian student visa.

Individuals as process integrators: Individuals are responsible to collect all the relevant data and integrate it effectively while planning their goals. This is a challenging task as the process domain is usually new to individuals and they do not have an overview of the process when progressing step by step. Heterogeneous data sources and data formats, numerous data dependencies over multiple organisations and constantly changing policies and workflows makes it even more difficult to play the role of process integrator.

Inability to predict task effects: Sometimes it is difficult to know what kind of effect accomplishing a specific task will have on the process as a whole. For example, while either of the IELTS and TOEFL English competency tests are accepted world wide, it is better to have IELTS scores if applying for Australian universities because they are better regarded.

Isolated individuals: Although people join communities on social networks and discuss issues with peers in forums and e-how sites, they are ultimately progressing in an isolated manner In Figure 1a, Igor is a member of Go8 communities on facebook and is also contributing on an IELTS message board, while reflecting his experiences in his own blog. There is no established way for Ali to be aware of all Igor's contributions on the process, to contact him or to learn from his progress. People with similar goals and interests may not be able to find each other easily and each individual's progress is not necessarily recorded for future reuse.

In *Processbook*, we consider merging social networks principles with process management basics to overcome the above issues. As Figure 1b illustrates:

- A goal-based community is established to remove the isolation barrier and help individuals find peers with similar interests/goals more easily.
- Users' knowledge and experience is captured unobtrusively while they are planning for their goal via a simple modelling interface that requires no prior process modelling knowledge.
- Individuals' process knowledge is merged to produce a general view of the process.
- Users' votes and comments are added to the socially produced process knowledge to minimize the adverse effect of invalid, inconsistent and incomplete data.

3 Related Work

Personal process management has so far received limited attention from the academic research community. A vision statement can be found on the blog posted by Michael Rosemann [1]. Two possible implementations of personal process management are discussed in [7] and [1]. [7] mainly focuses on sequential and conditional constraints by introducing a formal personal process modelling language. The proposal in [1] is based on parallel executions, tries to simplify BPM techniques and pays attention to the role of social aspects of the process management such as sharing and assigning tasks. Both works remain at preliminary level and are yet to realise any significant improvement over personal process management.

[1] http:// www.michaelrosemann.com/uncategorized/113/

On the other side, a large number of commercial online tools exist for personal task management. These tools are end-user oriented and provide a plethora of features including task creation, sharing, social network integration, notification, etc. However, as [1] notes, none of them is concerned about the “process” concept; they do not embrace the practices of BPM, thus losing many beneficial aspects of structuring dependencies and constraints between tasks.

In terms of requiring flexibility and agility, personal process management is closely related to agile BPM. The most notable defect of classical BPM is the “model-reality divide”, the distance between abstract process models and the processes executed in practice. To overcome this defect, [2] states that agile BPM not only requires changes to the BPM life cycle, but also a paradigmatic change. This paradigmatic change can be obtained by applying social software features into business process management.

There have been some attempts in recent years to accommodate social features in the BPM environment. Most notably a recommendation-based process modelling support system with social features in [4], a modelling and execution tool for business processes with collaboration and wiki-like features embedded in [6], and an ad hoc workflow system focusing on non-intrusive capturing of human interactions in [5].

While most of the existing works in the social BPM area focus on adding social features to an existing BPM framework, our architectural framework [3], *Processbook*, gives principles and guidelines for managing personal processes *within* a social network. It embeds four key capabilities in its underlying social network: collaborative process modelling, knowledge capturing and sharing, social-network-based recommendation and notification-based management of the dynamic environment. This paper is an attempt to realise the conceptual framework introduced in [3] focusing on the process knowledge discovery and recommendation in personal process domain.

4 Process Aware Personal Process Management System

Processbook aims (i) to make personal process management as effortless as possible for individuals and (ii) to utilise user participation to produce meaningful social collective data. The first step is to engage people to manage their processes through *Processbook*. Given that *Processbook* users are ordinary people rather than trained BPM designers or knowledge workers, they posses little or no knowledge or prior experience in process modelling and management. Therefore one of the main issues *Processbook* deals with is to find a way to provide support for individuals in managing their tasks while simultaneously taking advantage of their social participation to enrich its support. In Section 4.1 we propose a simple modelling interface based on the idea of ToDoLists to facilitate the modelling experience for non-technical users.The second step is to expedite the transfer of knowledge about processes among users. For this purpose, in Section 4.2 we propose a method to capture users process knowledge. Then in Section 4.3 we show how to share the process knowledge in form of process recommendation.

4.1 Modelling Interface

Traditional business process modelling lacks the required flexibility and agility when it comes to unstructured or ad-hoc processes. To break the rigidity of traditional modelling methods and to simplify their syntax for novice users, we propose a simple modelling approach that resembles the natural planning model our brain follows. There are five steps that our minds go through to accomplish any task: defining purpose and principles, outcome visioning, brainstorming, organising, and identifying next actions [2]. Similarly our proposed planning approach consists of five steps:

- Defining a *goal*; A goal is any desired result that requires one or more action steps. It is described in natural language and is mandatory for each plan e.g "Gain admission to a PhD degree in computer science at UNSW"
- Defining a set of *constraints*; Constraints are sets of parameters and criteria that further elaborate goals. They can be *global* to describe the general parameters e.g "Application deadline for PhD admission is 1 Dec 2013" or *local* to reflect personal visions on the goal e.g "field of study: BPM".
- Gathering all required *tasks*; Determining the set of required tasks is the first step towards the desired outcome. A task is a single unit of work in the boundaries of a particular goal. A goal is achieved when enough of the right tasks have been performed successfully and some outcomes have been created that closely enough match the initial vision.
- Elaborating tasks; A short-list of tasks are specified and elaborated by linking them to local constraints or by adding annotations to them.
- Identifying next task to do; The order in which users want to carry out their listed tasks is the final planning step and is repeated until all tasks are carried out or the desired outcome has been achieved.

Formally we define a simple modelling interface called *ToDoList* to realise the idea of natural planning.

Definition 1. A *ToDoList* is a quadruple (I, G, T, C), where

- I is a unique identifier for the ToDoList
- G is a statement of the *goal* in natural language
- T is a set of tasks to be done to achieve the goal
- C is a set of constraints on the tasks and the overall planning

Each task gives a natural language statement of one activity to be completed in achieving the goal. A task is a pair (I, D), where I is a unique identifier for the task and D is a description which can be either a natural language description of an atomic task or a reference to another ToDoList.

Tasks are first class entities in our proposed ToDoList modelling approach. They are entered by individuals or given to them through a recommendation mechanism (Section 4.3). Figure 2 illustrates a *task state diagram* in *Processbook.* Each task, at any given time, could be in one of the "planning", "carrying out", "carried out" and "captured" states.

[2] Allen, D.: Getting things done. penguin books (2001)

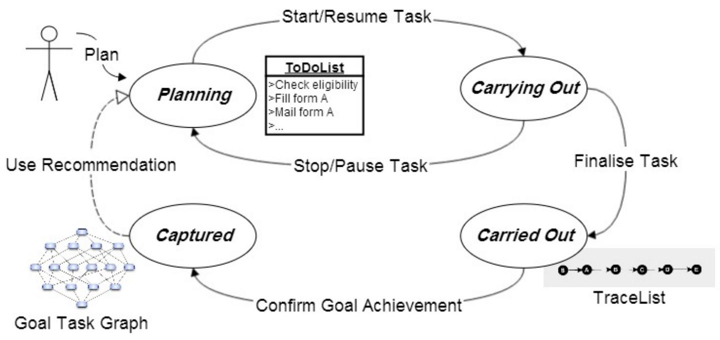


Fig. 2: Task State Diagram

- **Planning** Once a task is defined or recommended to a user it will be put in the planning state. The planning state is similar to drafting a ToDoList and resembles the brainstorming step.
- **Carrying out** Tasks are brought to "carrying out" mode on a user's decision to perform them. When a user identifies the next action she wants to take, she can simply pick the task from her ToDoList and bring it to the carrying out list. The task is then removed from the ToDoList, but can be reverted back when the user pauses or stops the execution of the task (to make changes to the task). When the user obtains the desired result from the running task, the task is considered finalised and will be moved to the "carried out" mode.
- **Carried out** The carried out state consists of a set of completed tasks, ordered by their completion timestamp. Hence it could be regarded as a trace log for each ToDoList. It is expected that by the end of the personal planning, no tasks remain in the ToDoList or in the carrying out list. Instead the required tasks for an achieved goal are found in the carried out mode or more specifically in *TraceLists* that we will later define in 4.2.
- **Captured** If the user cancels her plan or does not achieve her specified goal, her traces will be deleted. Otherwise the trace is aggregated with other users' traces for the same goal. The aggregation produces a graph called *GTG Goal-Task Graph*, where users' successful experiences for a particular goal are captured. The *GTG* is described in 4.2, and captures the social production for a community of users who share the same goal.

4.2 Capturing process knowledge

Associated with each ToDoList, there exists a *TraceList* that is opened after the first task in ToDoList has been carried out and is closed after the corresponding ToDoList terminates. A TraceList captures the execution of tasks in a ToDoList.

Definition 2. A TraceList is a quadruple (I, G, CG, H) where

- I is a unique identifier for the TraceList

- G is a statement of the goal in natural language
- CG is a set of constraints on the goal
- H is a set of history records of tasks and their properties

Each history record H consists of four elements:

- T is a unique identifier for the task
- CT is a set of constraints on the task
- ST is the time when task has been started
- ET is the time when task has been completed

A ToDoList terminates successfully if all its tasks have been carried out and the owner of ToDoList confirms that a desired outcome has been reached by performing those tasks. The TraceList of such a ToDoList execution is called a *complete* TraceList. Intuitively a set of complete TraceLists for a certain goal will give a useful insight on how to achieve that particular goal. We argue that each TraceList resembles a blog post describing a personal solution to reach the goal, while the aggregation of TraceLists builds a wiki that describes the general solution to reach that goal in different contexts and from different perspectives. *Processbook* realizes the concept of social production by merging all complete TraceLists of a goal into a graph structure called GTG(Goal-Task Graph).

Definition 3. A GTG is a weighted directed graph $GTG(I, G, V, E, W, A)$ where

- I is a unique identifier for the GTG
- G is the goal for which complete TraceLists are aggregated
- V is the union of tasks in the complete TraceLists with goal G; the result set forms the vertices of GTG
- E is the edges of GTG; each edge indicates the order of execution between two tasks
- W is a weight associated with the edges of GTG indicating the number of times a particular edge has been followed over all TraceLists of goal G.
- C is a set of constraints associated with the edges of GTG indicating the circumstances under which a flow of tasks has occurred over all TraceLists of goal G.

For example, $T_i \xrightarrow{3,\{c_1 \vee c_2\}} T_j$ means task T_j preceded task T_i, three times in all complete TraceLists with either c_1 or c_2 mentioned as constraints of T_j in those complete TraceLists. Procedure 1 demonstrates the procedure of merging TraceLists into a GTG for goal G. The input of the procedure is a set of complete TraceLists that have G as their goal. `PrepareTraceList` orders tasks in each TraceList by their ET attribute. More complicated metrics could also be applied to include the ST and C attributes of history records as well, though it is out of the scope of this paper.

Each precedence relationship between two tasks is then added to the graph as follows:`FindTask(`T_i, T_j, GTG`)` searches the graph for tasks T_i and T_j. If both tasks and their precedence relationship already exist in the GTG, we only need to increase the weight and update the constraint attributes of precedence. Otherwise if both tasks exist but they have not been connected, a new precedence should be added by (`AddPrecedence`) with its weight and constraints set. If only one of the tasks exists in GTG, we have to first add the missing task to the graph (`AddTask`), then add the corresponding precedence and finally set its weight and constraint attributes. This is similar to the case where both tasks are new to the GTG, with the minor difference that we have to add both tasks first.

4.3 Sharing process knowledge

Knowledge of achieving a goal, which we refer to as process knowledge, is captured when individuals carry out ToDoLists; this information is aggregated in the GTG. Process knowledge is the main artefact that is shared among users In *Processbook*. Process knowledge sharing happens via recommendations to targeted communities.

Figure 3 shows how such a sharing mechanism is realised in *Processbook*. For each goal, a community is created. Users are considered members of a community once they start towards a goal. These goal based communities, forming weak ties among users, are then used as a target group for recommendations. Users start planning their goals in the *ToDoList manager*. The executions of their plans are logged by *trace logger* in TraceList database. Secondary data from their involvements that includes votes and comments on recommended items are also recorded by the *event logger*. The *TraceList merger* performs aggregation of TraceLists (peer products) of a goal into the GTG to produce social production. Collective decisions are realised by applying users' votes on TraceLists and providing recommendations.

The *recommender* system consists of two different modules separated by the data sources they utilise: (i) *TraceList ranked retrieval module* that makes use of TraceList database and (ii) *process recommender* that utilises the GTG. Ranked retrieval of complete TraceLists could be based on several different metrics:

– Number of tasks in a TraceList; those with less tasks will be ranked higher implying they need less effort to achieve the goal
– Execution time of a TraceList calculated by $Max(ET) - Min(ST)$; those with shorter execution time will be ranked higher implying they reach the desired outcome sooner
– Popularity of a TraceList indicated by users' votes which reflects users' opinions on the usefulness and effectiveness of a TraceList

The Process Recommender uses two slightly different approaches: recommending the *next best task* or recommending a *process path*. Both approaches use the GTG as their data source. The next best task recommender gives users a task at a time while being dynamically modified as the GTG grows. The Process Path Recommender, on the contrary, provides the whole set of tasks needed to reach the goal in the form of a path. It does not reflect GTG changes unless the user explicitly asks for an updated recommendation. One advantage of the next

procedure 1 Merging *TraceLists* to *GTG*

```
Input: $TRL$: A set of complete TraceLists having goal $G$
Output: $GTG$ for goal $G$
  PrepareTraceList($TRL$)
  for all tracelists $trl$ in $TRL$ do
      for all $T_i \rightarrow T_j$ in $trl$ do
          Let $w_{ij} = trl\_weight_{T_i \rightarrow T_j}$
          Let $c_{ij} = trl\_constraints_{T_i \rightarrow T_j}$
          $found$ = FindTask($T_i, T_j, GTG$)
          switch $found$
              case both $T_i$ and $T_j$ were found in $GTG$
                  if $T_i \rightarrow T_j$ exist in $GTG$ then
                      $Weight_{T_i \rightarrow T_j} += w_{ij}$
                      $Constraints_{T_i \rightarrow T_j} = Constraints_{T_i \rightarrow T_j} \vee c_{ij}$
                  else
                      AddPrecedence($T_i \rightarrow T_j$)
                      $Weight_{T_i \rightarrow T_j} = w_{ij}$
                      $Constraints_{T_i \rightarrow T_j} = c_{ij}$
                  end if
              case $T_i$ was found in $GTG$
                  AddTask($T_j, G$)
                  AddPrecedence($T_i \rightarrow T_j$)
                  $Weight_{T_i \rightarrow T_j} = w_{ij}$
                  $Constraints_{T_i \rightarrow T_j} = c_{ij}$
              case $T_j$ was found in $GTG$
                  AddTask($T_i, G$)
                  AddPrecedence($T_i \rightarrow T_j$)
                  $Weight_{T_i \rightarrow T_j} = w_{ij}$
                  $Constraints_{T_i \rightarrow T_j} = c_{ij}$
              case neither $T_i$ nor $T_j$ were found in $GTG$
                  AddTask($T_i, G$)
                  AddTask($T_j, G$)
                  AddPrecedence($T_i \rightarrow T_j$)
                  $Weight_{T_i \rightarrow T_j} = w_{ij}$
                  $Constraints_{T_i \rightarrow T_j} = c_{ij}$
      end for
  end for
```

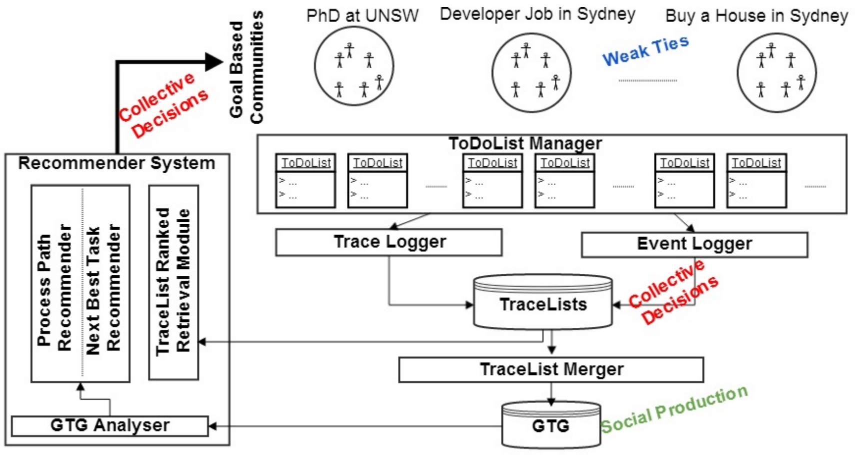


Fig. 3: Realisation of social software principles in *Processbook*

best task recommender appears when a user wants to select a task manually, which is not necessarily the next best one. In this case the recommender will consider calculating the best remaining path, taking into account what user has chosen so far. In both cases, the user may fully or partially accept the recommendation or may completely reject it and continue making her own plans.

The major advantage of the process recommender over the TraceLists ranked retrieval module is that it better reflects the knowledge of the crowd. It is because the next task or the process path given by the process recommender comes from merging different TraceLists and is considered to give the best possible solution from the crowd experience, while in the ranked retrieval system we are limited to the number of single TraceLists. The advantage of using a retrieval system based on single TraceLists is that what is finally recommended is already used by one or more users in real world, whereas in process recommender, the final recommended path might have not been carried out before by any user. In other words, it is a system generated path that is composed of different best practices found in the most successful attempts.

The basis of the process recommender is to find the minimum weighted path in the *GTG*. Weights in the *GTG* represent the popularity of a task flow but they have to be normalised so that Dijkstra's algorithm could be applied and the minimum weighted path from the starting task to the end task found. We also elaborate the weights of the *GTG* by taking into account users' votes in addition to the popularity measure. Therefore the final weight for an edge in *GTG* between tasks T_i and T_j is calculated as follows:

$$\alpha \times (10 - \frac{w_{ij}}{N} \times 10) + \beta \times (10 - vote)$$

where w_{ij} is the initial popularity weight of the edge between T_i and T_j, N is the number of complete TraceLists merged into the *GTG*, *vote* indicates the average votes for the task flow, ranging from 0 to 10 with 10 meaning the best and α and β are coefficients for popularity metrics and users' votes respectively.

It is probable that tasks that have been repeatedly used in complete TraceLists be excluded from the final recommendation due to existence of some unrealistic short paths in the graph. To avoid ignoring such tasks and thus improving the recommendation results, we define the concepts of *required tasks* and *required ratio*. The required ratio for task T_i is calculated by ${}^{w_{*i}}/_{N}$ where w_{*i} is the sum of the weights of ingoing edges to task T_i and N is the number of complete TraceLists merged into the *GTG*. A threshold for required ratio is set to force the inclusion of task T_i. Tasks whose required ratio is above the threshold are called required tasks, implying they are the necessary steps if the process is to achieve the goal. Considering required tasks, the recommendation mechanism should also be modified. To this purpose, the *GTG analyser* shown in Figure 3 is responsible for marking required tasks in each *GTG*. The final recommended paths will be filtered to avoid ignoring required tasks.

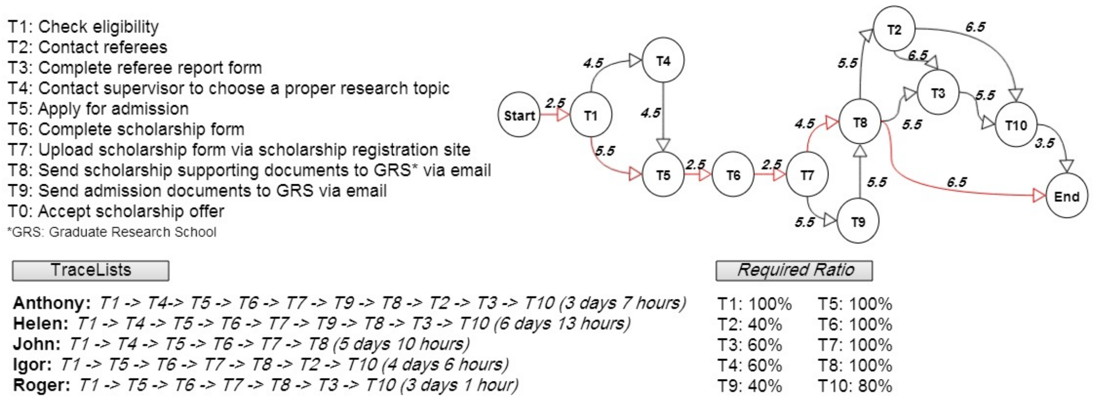
Towards Context-Aware Process Recommendation A further step in sharing the process knowledge is to enable context-aware process recommendation by taking into account user profiles and task constraints. The pre-requisite is to have a logical mapping between user profile attributes and task constraints. For instance, in a personal process which is about applying for a PhD scholarship, users should fill out the required fields about their educational background such as degrees they hold, universities and schools they have attended, etc.

Having a set of constraints associated with each edge in *GTG* in procedure 1, first challenge would be to find the most relevant constraint(s) for each edge. A simple approach is to rank those constraints based on the frequency of their appearance in the trace logs. This way, in its simplest form, we would have a *GTG* with each of its edges labelled with the most frequent constraint. Subsequently any of the recommendation algorithms introduced in 4.3 can be modified to include constraint labels in their internal logic.

An alternative approach is to first cluster trace logs based on the user profiles and then run procedure 1 to build *GTG*s for each cluster. An agglomerative hierarchical clustering algorithm can be used to find the proper number of clusters. Prior to using any recommendation algorithm, we should first assign each user a cluster of which its center is nearest to her user profile. The input of the recommendation algorithm would be the *GTG* which is built over that particular cluster.

5 Implementation and Pilot Study

A prototype of the proposed system is implemented as a Java-based Web application. In this section, we demonstrate our solution by presenting a simplified version of test cases we have run with *Processbook*. We have asked five students in our school to plan for the case "Applying for UNSW PhD scholarship" through ToDoList modelling interface of *Processbook*. Based on this input we show how the system generates recommendations for Ali, who also wants to apply for a PhD scholarship.


T1: Check eligibility
T2: Contact referees
T3: Complete referee report form
T4: Contact supervisor to choose a proper research topic
T5: Apply for admission
T6: Complete scholarship form
T7: Upload scholarship form via scholarship registration site
T8: Send scholarship supporting documents to GRS* via email
T9: Send admission documents to GRS via email
T0: Accept scholarship offer
*GRS: Graduate Research School

TraceLists

Anthony: *T1 -> T4-> T5 -> T6 -> T7 -> T9 -> T8 -> T2 -> T3 -> T10 (3 days 7 hours)*
Helen: *T1 -> T4 -> T5 -> T6 -> T7 -> T9 -> T8 -> T3 -> T10 (6 days 13 hours)*
John: *T1 -> T4 -> T5 -> T6 -> T7 -> T8 (5 days 10 hours)*
Igor: *T1 -> T5 -> T6 -> T7 -> T8 -> T2 -> T10 (4 days 6 hours)*
Roger: *T1 -> T5 -> T6 -> T7 -> T8 -> T3 -> T10 (3 days 1 hour)*

Required Ratio

T1: 100%	T5: 100%
T2: 40%	T6: 100%
T3: 60%	T7: 100%
T4: 60%	T8: 100%
T9: 40%	T10: 80%

Fig. 4: “Applying for UNSW PhD scholarship” TraceLists and *GTG*

Users’ knowledge and experience in planning the scholarship case has been captured in TraceList database. Task descriptions, complete TraceLists and votes statistics are provided in Figure 4. *Processbook* aggregates TraceLists and build the *GTG* shown in Figure 4. Without loss of generality, we assume that user votes for each edge is equal (e.g 5 out of 10). Moreover since the effect of constraints has not yet implemented in the recommender system of *Processbook*, we ignore constrains labels of *GTG* edges.

In terms of number of tasks, John’s TraceList stands higher while in terms of execution time, Anthony’s TraceList would be recommended to Ali. However, if Ali decides to use a process path recommender, he will be offered a new path which does not exist in any of those TraceLists. The recommended path excludes T_9 which as we detected is a common misunderstanding between applicants. However the path shown in Figure 4 also excludes T_3, T_4 and T_{10}, all seems mandatory tasks according to UNSW policies. To avoid this undesirable elimination, we have to tune the required threshold to 50%, to enforce inclusion of such required tasks.Therefore *Processbook* filters out the path illustrated in Figure 4 and returns the optimal path as can be seen in an screenshot of the system in Figure 5.

5.1 Preliminary Evaluation of Process Recommendation Algorithms

A main part of our proposed personal process management system is the recommendation module which plays a significant role in process knowledge sharing. A preliminary experiment was conducted to measure the effectiveness of the

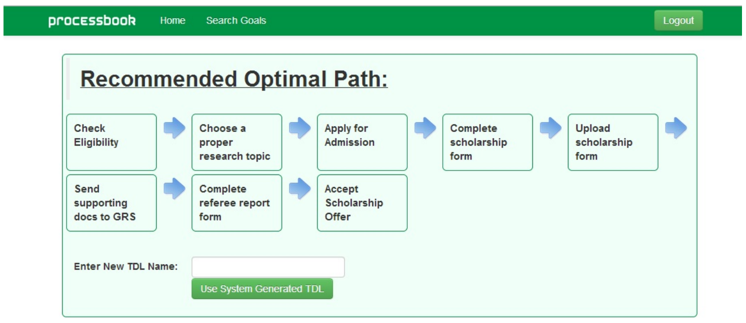


Fig. 5: Screenshot of optimal path generated by *Processbook*

core recommendation algorithms introduced in 4.3. For a proper evaluation we needed a fairly large amount of traces which was impossible to acquire from prototype logs in the limited time we had. Moreover, to the best of our knowledge there is no dataset available online to test process recommendation systems (or any path-based recommendation systems). As a result, we developed a TraceList simulator to produce TraceLists based on user profiles and expertise. Users' expertise parameter was added to enable generation of traces of different qualities: from nearly perfect traces to complete random task selection.

Two different algorithms were implemented for the purpose of this experiment: (i) *GTG*: created by procedure 1 labelled with the most frequent constraint, and (ii) clustered *GTG*: in which trace logs were clustered based on user profiles and procedure 1 was executed for each cluster afterwards. We used TraceList simulator to produces 50, 200, 500 and 1000 successful traces for Scholarship application process. Accuracy of each recommendation algorithm was measured by calculating the similarity between the recommended path and the ground truth (i.e the optimised path to apply for scholarships at UNSW). The similarity is formally calculated as follows:

$$1 - \frac{edit_distance(r,g)}{max(length(r),length(g))}$$

where r and g are recommended path and ground truth respectively and length indicates the number of tasks in a path. Table 1 summarizes the accuracy results obtained for both *GTG* and clustered *GTG* algorithms.

Table 1: Accuracy of recommendation algorithms

Size of log	*GTG*	clustered *GTG*
50	0.715	0.823
200	0.718	0.914
500	0.723	0.920
1000	0.731	0.917

The experiment shows that process paths recommended by clustered *GTG* is significantly more accurate than those recommended by *GTG*. However, we wish to emphasise that the experiment reported above is at an early stage and can be improved by (i) acquiring the real world traces and (ii) adding social network aspects such as users' votes and communities. It should also be extended to test variations of recommendation algorithms against various personal processes that differs in terms of size and complexity.

6 Conclusion and Future Work

Our proposed solution for personal process management is to create a flexible process management environment within a social network structure. We maintain the process awareness of traditional BPM to be able to handle dependencies and constraints between tasks, but at the same time we follow the principles of social software by establishing goal-based communities, enabling social production and utilising collective decisions. To realise all these, we have proposed a novel method for modelling personal processes based on the idea of ToDoLists and have implemented a mechanism to unobtrusively capture users' experience separately and then aggregate them in a graph structure that can be used as a source for process recommendation.

Our future work includes: (i) improving context-aware process recommendation, (ii) enforcing trustworthy recommendation by utilising more social software features and (iii) enriching our capturing mechanism by enabling parallelism in task flows. As well, we intend to evaluate *Processbook* in real world scenarios by conducting comprehensive user studies. We believe that *Processbook* can be employed in many knowledge intensive domains in addition to personal process management.

References

1. Brambilla, M.: Application and Simplification of BPM Techniques for Personal Process Management. In: Business Process Management Workshops, pp. 227–233 (2013)
2. Bruno, G., Dengler, F., Jennings, B., Khalaf, R., Nurcan, S., Prilla, M., Sarini, M., Schmidt, R., Silva, R.: Key Challenges for Enabling Agile BPM with Social Software. Journal of Software Maintenance pp. 297–326 (2011)
3. Hajimirsadeghi, S., Paik, H.Y., Shepherd, J.: Processbook: Towards social network-based personal process management. In: Business Process Management Workshops, pp. 268–279 (2013)
4. Koschmider, A., Song, M., Reijers, H.A.: Social Software for Modeling Business Processes. In: Business Process Management Workshops. pp. 666–677 (2008)
5. Martinho, D., Silva, A.R.: Non-intrusive Capture of Business Processes Using Social Software - Capturing the End Users' Tacit Knowledge. In: Business Process Management Workshops (1). pp. 207–218 (2011)
6. Silva, A.R., Meziani, R., Magalhães, R., Martinho, D., Aguiar, A., Flores, N.: AGILIPO: Embedding Social Software Features into Business Process Tools. In: Business Process Management Workshops. pp. 219–230 (2009)
7. Weber, I., Paik, H.Y., Benatallah, B., Vorwerk, C., Zheng, L., Kim, S.: Personal Process Management: Design and Execution for End-Users. Tech. rep., UNSW-CSE-TR-1018 (2010)