Managing Personal Processes in the Support of Interorganizational Workflows

San-Yih Hwang[a] , Jiun-Kai Tu[a], Wan-Chien Lee[b]

[a]*Department of Information Management*
National Sun Yat-Sen University
Kaohsiung 80424, Taiwan

[b]*Department of Computer Science and Engineering*
The Pennsylvania State University
University Park, PA 16802-6106, USA

Sept. 2003

*Corresponding author:
Tel.: 886-7-5254700; e-mail: syhwang@mis.nsysu.edu.tw

Abstract

In this paper we exploit a user-centered approach to interorganizational workflows and propose the concept of *personal processes*. A personal process is a coordination of personal tasks, each requiring a joint effort between a user and an enacting organization to achieve a personal goal. The interoperation between organizations that engage in a common business often requires cumbersome administration procedures and costly computing infrastructure, and hence we propose that direct communications between two enacting organizations are not implemented when managing personal processes. A personal process is managed by a *personal workflow management system* (PWFMS) running on a handheld device. When a personal process is executed, communication occurs only between the PWFMS and the organizations, implemented via Web services. Moreover, since the implementation of many personal tasks often requires physical interactions with the associated organizations, times and places of execution need to be recorded. To satisfy these unique requirements, we formally define a personal process model and design a lightweight architecture for systematically supporting the management of personal processes. A prototype system has been implemented that includes a PWFMS running on a Palm PDA and two subsystems running on fixed networks. All communications between components of the prototype are enabled using Web services.

Keywords: Interorganizational workflows, Personal processes, Web services, Process integration

1 Introduction

The World Wide Web – running on the Internet – has revolutionized the way business is conducted. While long-lasting, stable business-partner relationships between enterprises remain important, organizations are increasingly seeking to engage parties with short-term partnerships (i.e., those with no prior trading relationships) in their common business processes in order to increase revenue and to meet customers' requirements in today's dynamically changing business environment. The term *business-to-business commerce*, abbreviated as B2B, refers to the workflows crossing organizational boundaries. Many standards (e.g., Wf-XML [WfXML01] and BPMI [BPMI03]), prototypes (e.g., Sagitta2000 [Aals99], CMI [Schu00], Mentor-Lite [Weis00], WISE [LASS00], WSFL [Leym01], XLANG [That01], SELF-SERV [SBDM02], and BPEL4WS [Curb03]), and products (e.g., RosettaNet [Roset03], Biztalk [Bizt03], and ebXML [ebXM03]) have been developed to aid the execution of business processes between companies.

The seamless integration of subprocesses executed in different companies requires many challenges and technical issues to be addressed, including heterogeneity, autonomy, external manageability, adaptability, security, and scalability [Medj03]. The many efforts addressing these issues are mostly focusing on factors related to enterprise business rather than to personal business. In fact, the concept of B2B and its implementation will not only affect business exercises but also our daily life. Indeed, taking care of daily personal business often requires interactions with several organizations. However, even though XML and Web services have emerged as the de facto standard vehicles for communicating across enterprises, the infrastructures for integrating business processes across organizations are often heavy duty and hence

inappropriate for executing personal processes. Thus, in this paper we present a new research direction, namely *personal process management*, and describe related research issues and our solutions to them. To the best of our knowledge, this is the first reported investigation on personal process management.

To illustrate the concepts and requirements of personal process management, we discuss two example processes encountered in daily life: (1) vehicle registration and (2) medical insurance reimbursement.

1.1 The vehicle-registration process

Consider a car owner David who moves from Minnesota to Pennsylvania and needs to change his vehicle registration. When planning the relocation, David realizes that he can get a better insurance policy in Pennsylvania. However, to cancel the current insurance policy in Minnesota, he must send back the old vehicle plate. Thus, in addition to registering his car at a Pennsylvania vehicle-registration office, David must perform three tasks: (1) obtain a new insurance policy, (2) cancel the old policy, and (3) send the car's old plate back to the Driver and Vehicle Services Division in Minnesota (to cancel his car registration). The entire process is shown in Figure 1, where a task is represented as an arrow that takes a set of data items (or documents) as the input and produces another set of output data items. Both sets of data items are represented as circles. Obviously, each task must be initiated by David and then processed by the applicable enacting organization. The execution order of tasks has to be determined by the input and output data items required for executing the tasks. It is quite likely for each task to be a business process executing within an enacting organization. However, from the viewpoint of David, if the execution of an internal business process does not involve him, the details of the business process can be

encapsulated and hence he need only be concerned with its input and output. On the other hand, if a business process requires an individual to accomplish certain tasks, more detailed steps have to be made explicit in the process. For example, the task “get policy” may involve several subtasks, each of which must be initiated by David. Moreover, different insurance companies may require different sets of input data and produce different sets of output data. The process template, as provided by the Pennsylvania vehicle-registration office for the convenience of vehicle owners, must be flexible enough to incorporate various (sub)processes coming from different organizations and meet the specific requirements of diverse users.

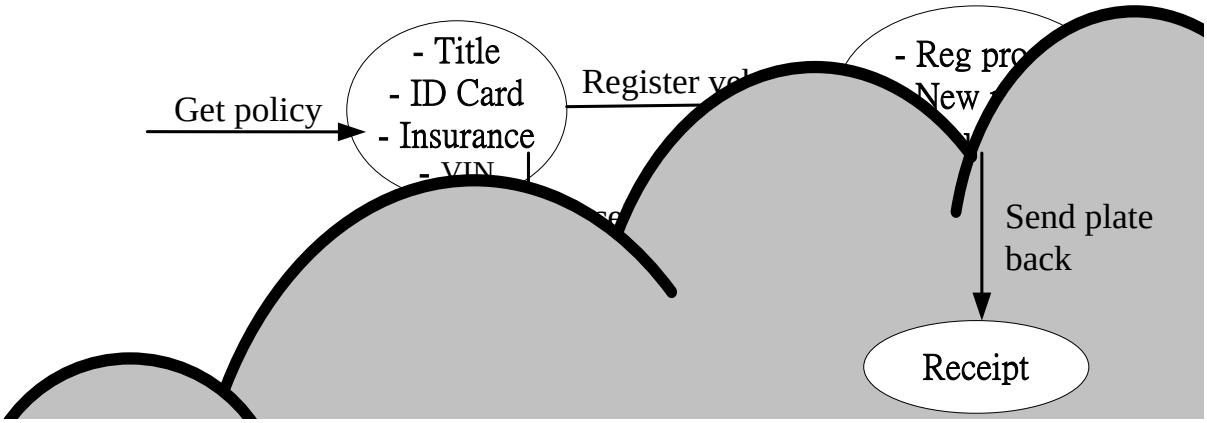


Figure 1: Vehicle-registration process

1.2 The medical-insurance-reimbursement process

This example relates to obtaining reimbursement of medical expenses from the Bureau of National Health Insurance (BNHI) in Taiwan. The main task in this process is to execute a ***Reimburse*** task (see Figure 2), which requires an individual to bring a set of documents, namely the insurance certificate, diagnosis certificate, and severe disease certificate, to a BNHI agency for a medical expense claim. However, these documents may be issued by different organizations, such as the employer that offers insurance to the applicant and the hospital that provides medication to the applicant.

Obtaining a document may in turn require the execution of other processes. For example, to receive a diagnosis certificate, many hospitals require the applicant to first fill out an application form, have the doctor detail the diagnosis results, and then pass the form on to the authority for final stamping (a signature). All these tasks must be initiated by the applicant (see Figure 2). Note that obtaining a severe disease certificate requires a different process, which is not shown here for brevity.

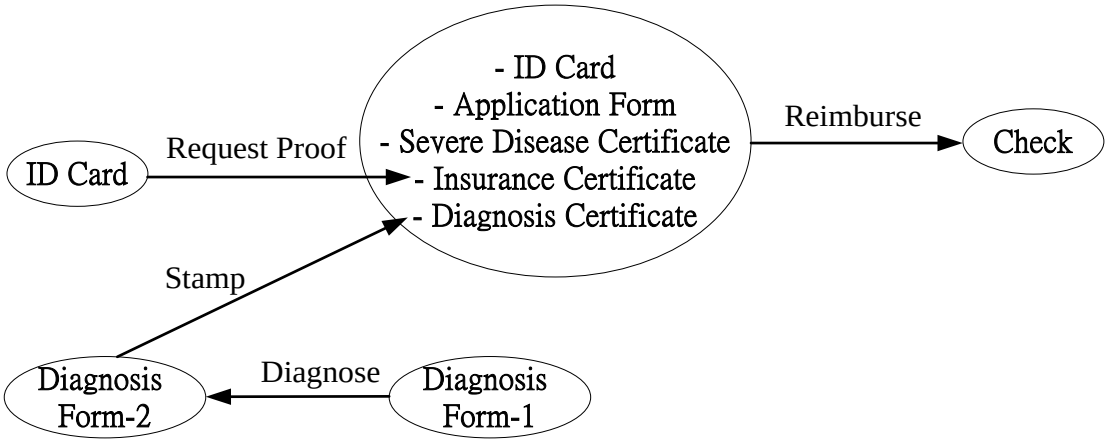


Figure 2: Insurance reimbursement in Taiwan

The user-coordinated processes described above are termed *personal processes* because they are for personal business. The examples illustrate the need for organizations to build systems and extend their IT infrastructure to the support of personal process management. In this paper, we employ a suite of system functions – collectively termed the *personal workflow management system* (PWFMS) – for the management of personal processes.

Enterprises involved in conventional interorganizational processes must reach some sort of agreement in advance in order to achieve automatic interaction between two related activities executed in different organizations. The procedure is shown in Figure 3(a). However, enabling the interactions for activities involving different

organizations often requires substantial administration effort and costly investment in computing infrastructure. It is therefore unlikely that two unrelated organizations will communicate with each other systematically just to meet a customer's dynamic requirement. In contrast, in our proposed personal process model (depicted in Figure 3(b)) the user serves as an agent for interaction between two tasks performed in different organizations. In such a model, the only type of communication involved in executing a personal process is between the user and organizations – organizations do not talk to each other directly. This model not only eliminates the substantial cost associated with enterprise cooperation but also preserves user autonomy.

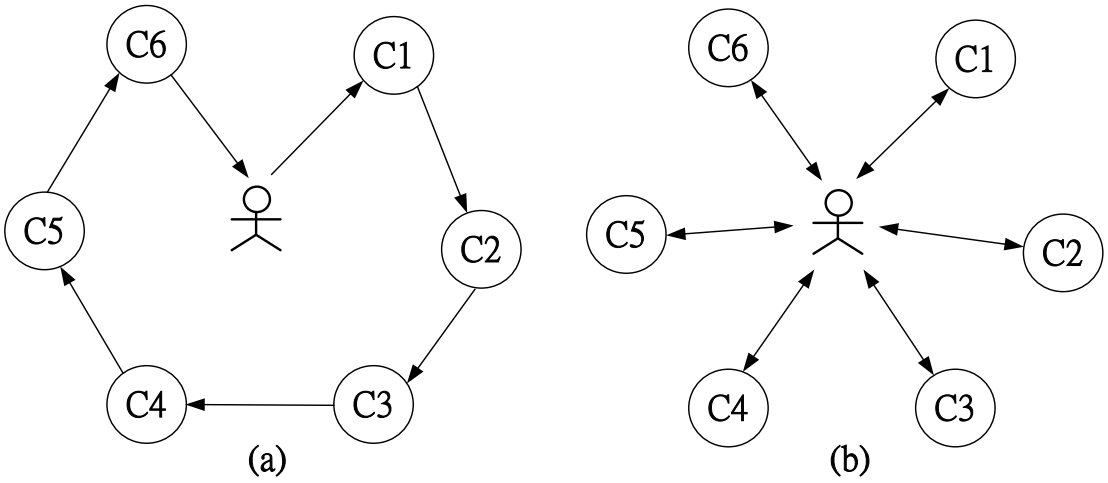


Figure 3: Execution of (a) a conventional business process and (b) a personal process

1.3 Observations

Some characteristics of personal processes and some requirements for managing them are as follows:

1. Tasks in a personal process require interactions between a user and enacting organizations. In fact, many tasks require physical interactions to be performed within certain time periods and at certain places. The order in which tasks are executed is determined mainly by the executable times, executable places, and

the data dependency of the tasks. Enterprises may impose many policies and procedures that must be followed strictly by their processes, whereas an individual seldom imposes rigid rules on the order of execution of his or her personal tasks. As a result, the control flow constructs required for enterprise processes, such as or-split, and-split, or-join, and and-join, are rarely required for personal processes.

2. Designing a valid personal process is not easy, and it is not surprising for a user to execute a task that produces unexpected results or even engage in a totally unexpected task. We call such situations *exceptions*. Rather than rejecting an exception, the user may wish to know its impact.
3. Personal process management can make a user's life more convenient. However, in contrast to commercial workflow management systems that *enforce* task executions, we envisage that a PWFMS would provide only suggestions or reminders to the user. Therefore, traditional workflow scheduling issues, which address how to determine the mapping between tasks and available resources, do not exist in this context. Instead, the query capabilities of a PWFMS that keep a user updated about the current personal process status are essential.
4. To exploit the full strength of personal process management, a PWFMS should be available on various platforms. Moreover, to facilitate ubiquitous access of personal process management information, it is desirable to implement a PWFMS on mobile devices (e.g., PDAs).
5. For a given user, a personal process may be executed only a small number of times (and probably only once in many cases). However, many different users may share a common interest in a particular personal process. Therefore, it is imperative for an organization to provide personal processes of high interest to its customers. In addition, the personal processes of different organizations

should be integrated in a coherent way. In other words, there is a need to provide tools for developing and managing predesigned personal process templates.

6. While the operation of the organization is beyond the control of the user, it is important to notify the user about the status of the process.

1.4 Contributions

In addressing the above observations and requirements, we develop an innovative (personal) process model that comprises executable times, executable places, and task input and output data. We also define the correctness of a personal process and develop methods for verifying the correctness of a personal process instance (corresponding to a personal process design), the completion of process executions, and changes to the process. To demonstrate the feasibility of our proposal, a PWFMS is designed and prototyped on a handheld device, thereby enabling personal processes to be queried at any time and at any place. To facilitate organizations to provide predesigned personal process templates to their customers, we develop a personal process template provider based on Web services technology. The status of a personal process can be tracked and reported to its user. All the components described in the architecture have been implemented on a Web-based system and a Palm PDA. The implementation status is also reported in this paper.

The rest of the paper is organized as follows. Section 2 formally defines the personal process model. Section 3 describes the associated consistency constraints used to verify the correctness of a personal process. Section 4 discusses the types of queries a PWFMS should provide and the syntax for expressing them. Section 5 describes the architecture for facilitating the management of personal processes. Section 6 presents our prototype system that is an implementation of the architecture described in

Section 5. Section 7 reviews the related work in the literature. We conclude the paper in Section 8 by summarizing the main results and identifying issues for further study.

2 The personal process model

In this section, we define the syntax and semantics of personal processes. A *personal process* is comprised of the following components:

1. A set T of tasks.
2. A set D of data items.
3. A set R of threads, each representing a distinct output as a result of executing a task.
4. Several functions that map a task to its name (Φ_n), the input data set (Φ_i), the time intervals when and places where execution is possible (Φ_{tp}), the set of participating threads (Φ_r), the type of nesting (Φ_{nest}), the tracking type (Φ_{type}) , and the service provider's URL (Φ_{url}), as defined below:
   A. Φ_n: $T \rightarrow$ String, which indicates the name of a task.
   B. Φ_i: $T \rightarrow 2^D$, which indicates the input data set of a task.
   C. Φ_r: $T \rightarrow 2^R$, which indicates the set of threads pertaining to a task.
   D. Φ_{tp}: $T \rightarrow 2^{Interval \times Region}$, which indicates the set of time intervals when and geographical regions where a task can be executed[1].
   E. Φ_{nest}: $T \rightarrow$ Boolean, which indicates whether the task is nested (i.e., a subprocess).
   F. Φ_{track}: $T \rightarrow$ Boolean, which indicates whether the task can be tracked automatically. A task is said to be automatically tracked if the enacting

[1] For formal definitions of "interval" and "region", readers are referred to [Guet00]. However, to ease representation, we simply denote a region by its name and use a five-tuple to represent periodically executable intervals.

organization provides a Web service for notifying the execution status of the task. A task is tracked manually if it is not automatically tracked.

G. Φ_{url}: $T \rightarrow$String, which indicates the URL of the Web service provided by the organization. This is valid only for tasks that can be tracked automatically.

5. A function Φ_o: $R \rightarrow 2^D$ that maps a thread to its output data set.
6. A function Δ_n: $D \rightarrow$String that maps a data item to its associated name.

In addition to the above items, there are attributes that record the execution status of processes, tasks, threads, and data; we call these *instance* attributes. We consider five instance attributes Φ_{ps}, Φ_s, Φ_{rs}, Φ_c, and Δ_s that are associated with processes, tasks, threads, or data[2],. Φ_{ps}: $P \rightarrow$ (INITIAL, EXECUTING, SUCCESFUL, FAILED) reveals the status of a process, which can be *initial, executing, successful, or failed*. Φ_s: $T \rightarrow$ (INITIAL, PREPARED, EXECUTING, LOGICALLY COMPLETED, FAILED, PHYSICALLY COMPLETED) reveals the status of a task, which can be *initial, prepared, executing, logically completed, failed, or physically completed*. Φ_{rs}: $R \rightarrow$ (UNDECIDED, FAILED, SUCCESFUL) describes the status of a thread. For a given task, at most one of its threads could become SUCCESSFUL. Φ_c: $T \rightarrow R$ maps a task to a completed thread. Δ_s: $D \rightarrow$ (UNAVAILABLE, AVAILABLE) describes the availability of a data item. The status of tasks and threads is explained in more detail in Section 2.1 and 2.2.

The seven functions Φ_i, Φ_r, Φ_c, Φ_{tp}, Φ_{nest}, Φ_{track}, and Φ_{url} are attributes pertaining to tasks, and Φ_o is an attribute pertaining to threads. A task may comprise several threads (represented by Φ_r), each of which represents a distinct execution outcome (represented by Φ_o). As a result, each thread has its output data set. Of the various

[2] Note that a personal process for a given user has at most one instance at any time in most cases. Thus, the term process (task, data, or thread) refers to either the definition or the instance – the actual meaning should be clear from the context.

threads of a task, only one (represented by Φ_c) can eventually succeed. Φ_{tp} is the attribute that specifies when and where a task can be performed. An interval is modeled as a five-tuple <*start, cycle, from, to, end*>, where *start* and *end* indicate the effective time and the expiry time, respectively, and *cycle*, *from,* and *to* dictate the effective period of a repetitive cycle. The format used for *start, end, from,* and *to* is "yyyy-mm-dd:hh:mm:ss" and the domain of *cycle* is {year, month, week, day, hour}. For example, the interval of a task that can only be executed between 9 a.m. and 5 p.m. everyday effective from July 1, 2003 to August 31, 2003 is specified as follows: <2003-07-01, "day", 09:00:00, 17:00:00, 2003-08-31>.

We categorize data items into two types: *primitive* and *processed*. Primitive data is not produced by any task modeled in the system, and processed data must be generated by at least (a thread of) one task. Primitive data could be a data file, a blank form, a personal belonging (e.g., ID card or credit card), or anything that can be prepared by the user. Processed data are available only when at least one task that is capable of producing it is completed. Consider the insurance-reimbursement process shown in Figure 2. The *Reimburse* task takes the following five data items as input: diagnosis certificate, insurance certificate, severe disease certificate, ID card, and application form; of these, only the diagnosis certificate and insurance certificate are processed data items, generated by *Stamp* and *RequestProof*, respectively; the other items are primitive data.

We also define a source data set P_s and a target data set P_t for correctness verification. The source data set is the set of data items that are currently available, formally P_s={d∈D: Δ_s (d)=AVAILABLE} and a target data set $P_t \subseteq D$ is such that the availability of each data item in P_t marks the successful termination of the personal

process. Consider the vehicle-registration process shown in Figure 1. One may set P_t={'Receipt1', 'Receipt2'} as the successful endpoint of the process when both receipts are received. We will make use of the two data sets in deciding whether a personal process is correct, as explained in Section 3.

Note that the order of task executions in the proposed model is not rigid: tasks are associated by their respective attribute values, which may implicitly determine their order of execution. For example, if a task T_2 requires a data item that can only be produced by T_1, T_2 will not begin executing before T_1 terminates.

2.1 Task status

The transitions between the states of a task can be expressed as a state-transition diagram, as depicted in Figure 4.

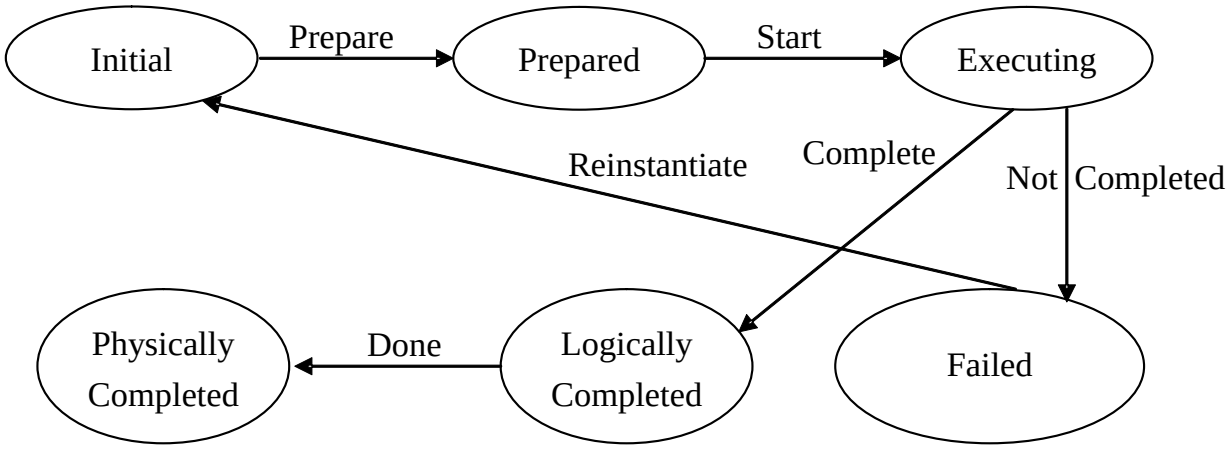


Figure 4: State-transition diagram of a task

The "prepare" event indicates that the user has engaged in executing a task. Consider the task *Register vehicle* in the vehicle-registration process as an example. When David has come to a vehicle-registration office and applied for the vehicle registration, the task is said to be *Prepared*. The "start" event, signified by an enacting

organization, indicates that the corresponding business process in the organization is enacted. Continuing with the specific example, when the internal vehicle-registration process is initiated to process Dave's vehicle-registration application, the task is said to be *Executing*. The "complete" event – also signified by the enacting organization – specifies that this task has been completed successfully (i.e., one of its relevant threads becomes SUCCEEDED). When the internal vehicle-registration process is completed, the task is said to be *Logically completed*. However, at this point some physical objects (e.g., the new vehicle plate) may not have reached the user. Once all output data items have arrived, which marks the event as "done", the task is said to be *Physically Completed*. However, if for some reason the task is voluntarily or involuntarily terminated by the user, an event "not completed" will be issued. In our vehicle-registration example, if the vehicle-registration station found that the ID card supplied by David has expired, the internal vehicle-registration process will terminate, and the task is said to have *Failed*. In this case the task behaves as if it was never executed. If the user decides to reinstantiate the task, it will become *Initial* and is ready to be executed. In the following, when we say that a task *t* is unexecuted, we actually mean that it is in the *Initial* state.

2.2 Task output thread

The output of each task can be expressed as a group of threads, as shown in Figure 5. Initially, all threads are in the UNDECIDED state. When the task is *Logically completed*, exactly one thread enters the SUCCEEDED state while all the other threads are in the FAILED state.

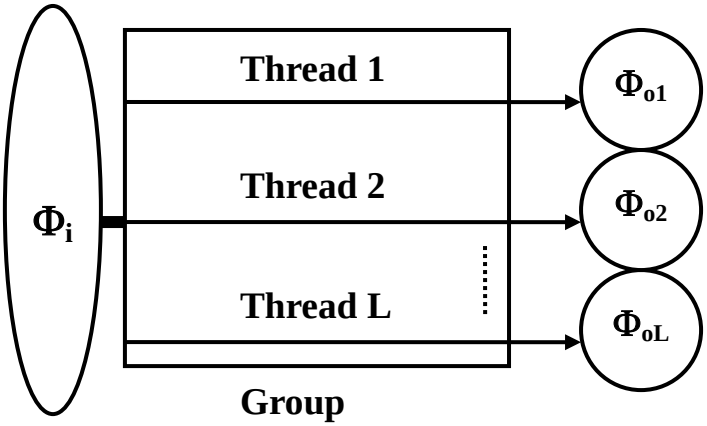


Figure 5: Task threads

2.3 Metaschema of personal processes

Based on the above definitions, we use a class diagram (shown in Figure 6) to represent its metaschema. A personal process comprises at least one task. Each task consists of at least one thread, a set of input data items, and at least one interval–region pair. A thread represents a possible execution outcome which is modeled as a set of output data items. The interval–region set contains the time intervals when and locations where the task can be executed. The attributes of each class are listed in Table 1.

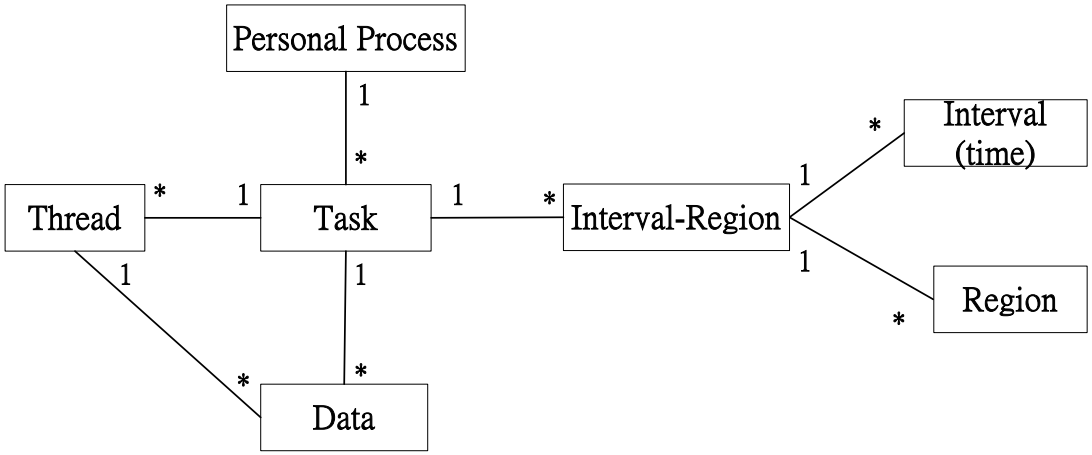


Figure 6: Metaschema of personal processes

Table 1: Attributes of classes in the metaschema

Class	**Attribute**	**Type**	
Personal process	Name	String	
	Status	INITIAL, EXECUTING, SUCCESS, or FAILED	
	Provider	String	
Task	Name	String	
	Priority	(low)1, 2, 3, 4, or 5(high)	
	Status	INITIAL, PREPARED, EXECUTING, LOGICALLY COMPLETED, FAILED, or PHYSICALLY COMPLETED	
	Nesting	Yes (nested) or No (atomic)	
	Tracking	Yes (tracked automatically) or No (tracked manually)	
	URL	String	
	Description	String	
Thread	Name	String	
	Status	UNDECIDED, SUCCEEDED, or FAILED	
Data	Name	String	
	Status	AVAILABLE or UNAVAILABLE	
	Type (for task)	PRIMITIVE or PROCESSED	
Interval–region	Interval (time)	Start	Date format “yyyy-mm-dd:hh:mm:ss”
		End	Date format “yyyy-mm-dd:hh:mm:ss”
		From	Date format “yyyy-mm-dd:hh:mm:ss”
		To	Date format “yyyy-mm-dd:hh:mm:ss”
		Cycle	year, month, week, day, or hour
	Region	String	

3 Constraints on personal processes

As a formal process model, the personal process model contains several constraints that limit the attribute values that can be specified on tasks, as defined below.

Definition 1: A task t_1 is said to temporally precede another task t_2 (denoted $t_1 \prec_t t_2$) if t_1 must be executed before t_2. Formally, $t_1 \prec_t t_2$ if

$\max\_time(\Phi_{tp}(t_1)) \le \min\_time(\Phi_{tp}(t_2))$, where max_*time* (min_*time*) is a function that returns the maximum (minimum) time of the intervals specified in the parameters.

Definition 2: A task t_2 is said to depend on another task t_1 in terms of data (denoted $t_1 \prec_d t_2$) if at least one of the input data items required by t_2 can only be generated by t_1. Formally, $t_1 \prec_d t_2$ if

$$\{d : d \in \Phi_i(t_2), (\exists r \in \Phi_r(t_1), d \in \Phi_o(r)), (\forall t \in T, t \neq t_1, \forall r \in \Phi_r(t), d \notin \Phi_o(r))\} \neq \phi .$$

Definition 3: A task t_2 is said to strictly depend on another task t_1 (denoted $t_1 \prec_s t_2$) if either $t_1 \prec_t t_2$ or $t_1 \prec_d t_2$.

Definition 4: Given a set of data items D_1 and another set of data items D_2, we say D_1 induces D_2 via a set of tasks T_i if there exists a partial order $\prec$ on T_i such that

1. $\prec_s \subseteq \prec$ and
2. there exists a thread r_j for each task t_j in T_i such that each input data item of t_j is either available or can be generated by some task preceding t_j in $\prec$. That is, $\Phi_i(t_j) \subseteq P_s \cup \{d : d \in D, (\exists r \in t, t \prec t_j, d \in \Phi_o(r))\}$.

In this case, we also say that T_i induces D_2 from D_1.

The first condition in Definition 4 confines the partial order to be the superset of the strict dependency $\prec_s$, since the strict dependency must be followed rigidly. The second condition states that the output of all tasks that precede a task t_j as well as the currently available data items will allow t_j to be executed (in terms of its required input data).

Definition 5: Given a set of data items D_1 and another set of data items D_2, we say D_2 is inducible from D_1 if there exists a set of tasks T_i such that D_1 induces D_2 via T_i.

3.1 Process-aliveness constraint

Definition 6: (Process aliveness) A personal process is said to be alive (or preserve the process-aliveness constraint) if the target data set P_t is inducible from the source data set P_s.

The process-aliveness constraint ensures that there is a chance that all the desired data items will become available. When the process-aliveness constraint is violated, it makes no sense to continue executing the process.

3.2 Task-aliveness constraint

Even if a process is alive, some of its unexecuted tasks could be redundant and will not or should not be executed. The task-aliveness constraint is proposed to ensure that task redundancy does not exist.

Definition 7: A task t is said to potentially contribute to a data item d if there exists a sequence $t_1, t_2, \ldots, t_n$ of unexecuted tasks such that the following conditions hold:

1. $t_1 = t$, i.e., the task sequence starts with t.
2. $\forall 1 < i < n, \forall 1 \le j < i, t_i \neg \prec_s t_j$, i.e., a task t_i cannot strictly follow any preceding task t_j.
3. $\forall 1 \le i < n, \exists r \in \Phi_r(t_i), \Phi_o(r) \cap \Phi_i(t_{i+1}) \neq \phi$, i.e., the output data item of some thread in a task t_i can be consumed by the next task t_{i+1}.
4. $\exists r \in \Phi_r(t_n), d \in \Phi_o(r)$, i.e., some thread in the last task t_n is capable of generating d.

Saying that a task t potentially contributes to a data item d means that the execution of t may lead to d becoming available in the future.

Definition 8: A task t is said to be alive if $\Phi_i(t)$ is inducible from P_s and t potentially contributes to at least one data item in P_t.

When a task is alive it has the chance to be executed, and its execution may contribute to the availability of at least one data item in the target data set. However, even if each task is alive, there is still a possibility that some redundancy exists in the specification of a personal process.

3.3 Acyclic-dependency constraints

Definition 9: (Acyclic dependency) A personal process is said to preserve the acyclic-dependency constraint if the graph (T, $\prec_s$) does not contain a cycle. In other words, the task dependency $\prec_s$ forms a partial order set on tasks.

The acyclic-dependency constraint is essential because not all the tasks involved in the cycle of (T, $\prec_s$) can be executed, and thus some tasks are redundant.

Definition 10: (Task aliveness) A personal process is said to satisfy task aliveness if every constituent task is alive and the acyclic-dependency constraint is preserved.

Note that it is sensible to continue executing a personal process only when the process is alive. Furthermore, if a personal process does not preserve task aliveness, some tasks become redundant and hence do not need to be executed.

3.4 Correct personal processes

Definition 11: (Correctness) A personal process (instance) is said to be correct if the task-aliveness constraint and the process-aliveness constraint are both satisfied.

A personal process can be incorrect for the following reasons:

1. Some tasks become obsolete (and should be removed).
2. The user may have made some mistakes in designing or changing the personal process.
3. The personal process was so badly executed that it makes no sense to continue executing the remaining unexecuted tasks.

In any of the above cases, some action may have to be taken to fix the personal process.

Please note that a personal process may be correct at design time but become incorrect when a task completes (as the result of executing an inappropriate task thread) or when the personal process undergoes some dynamic change. Therefore, the correctness of a personal process *p* should be checked at all of the following time points:

1. When *p* is initially designed.
2. When a task in *p* is completed.
3. When the definition of *p* is changed dynamically (e.g., by adding, deleting, or modifying a constituent task).

Task aliveness can be verified through *breadth first search* (BFS) for three times (one starts from P_s to verify it P_s reaches all unexecuted tasks, another starts from P_t by following the reverse edges to check if all unexecuted tasks can be passed, the other is performed on the data dependency graph to detect cycles). Process aliveness detection may be performed through BFS from P_s. However, since each task may comprise

multiple threads, the running time of BFS is exponentially proportional to the number of tasks. This suggests that checking the correctness of a personal process requires substantial computing power. As described in Section 4, when the PWFMS executes on a handheld device it is not possible to validate the entire personal process each time a task is completed. However, as shown by the following lemmas, in many cases there is no need to perform the personal process validation from scratch.

Lemma 1: Assume a personal process *p* satisfies the acyclic-dependency constraint before a task *t* is completed. *p* still satisfies the acyclic-dependency constraint after *t* is completed.

Proof: Suppose *p* violates the acyclic-dependency constraint after executing *t* and there exists a cycle in the graph (T, $\prec_s$). For each edge (t_i, t_j) in the cycle, either $t_i \prec_t t_j$ or $t_i \prec_d t_j$ must hold. Before *t* is executed, fewer data items are available and thus $t_i \prec_t t_j$ or $t_i \prec_d t_j$ must still hold. In other words, the cycle exists before *t* is executed. A contradiction!

From Lemma 1, we know that the acyclic-dependency constraint only needs to be verified once when a personal process is designed, provided the definition of personal process has not since been modified.

Lemma 2: Assume a personal process *p* is correct before a task *t* is completed. If *t* contains only one thread, then *p* is still correct after *t* is completed.

Proof: This is obvious.

Lemma 3: Assume a personal process *p* is correct before a task *t* is completed. Let *U*

be the set of unavailable data items that can be produced by some failed threads of *t* (i.e.,

$U = \{d : d \in D, \Delta_s(d) = UNAVAILABLE, (\exists r \in \Phi_r(t), \Phi_{rs}(r) = FAILED, d \in \Phi_o(r))\}$)

after *t* is completed. If there does not exist any data item *d* in *U* such that *d* is an input data item of some unexecuted task in *T* (i.e., $\forall d \in U, (\forall t \in T, \Phi_s(t) = INITIAL, d \notin \Phi_i(t))$), then *p* is still correct.

Proof: Since *p* is correct before *t* is executed, the task-aliveness and process-aliveness criteria must both hold. In other words, the following three conditions must be true:

1. There exists a set of unexecuted tasks T_1 that induces P_t from P_s.
2. For each unexecuted task *t*, there exists a set of unexecuted tasks T_2 that induces $\Phi_i(t)$ from P_s.
3. For each unexecuted task *t*, there exists an unexecuted task sequence T_3 that leads *t* to some data item in P_t.

Since all the failed threads in *t* do not contribute to the availability of any input data items of any unexecuted task, the following conditions will hold:

1. $T_1 - \{t\}$ induces P_t from P_s.
2. For each unexecuted task *t'*, $T_2 - \{t\}$ induces $\Phi_i(t')$ from P_s.
3. For each unexecuted task *t'*, $T_3 - \{t\}$ leads t' to some data item in P_t.

Thus, *p* is correct after *t* is executed.

4 Querying a personal process

We adopt first-order predicate calculus [BM77] as the formal query language for querying personal processes. For example, to retrieve the names of tasks in the personal process “medical insurance reimbursement” (shown in Figure 2) with a

given data item “insurance certificate” as part of their input, we can use the following query:

Query scenario 1

{*t.name* | TASK(*t*), (*t.process_name* = ‘medical insurance reimbursement’), (∃ *d* ∈ *t.input*, (*d.name* = ‘insurance certificate’))};

4.1 Predicates definition

To facilitate the users in expressing queries, we further define the following predicates that are frequently used in the context of personal processes:

- **OVERLAP_TIME_PLACE(t_1, t_2).** This returns TRUE if tasks t_1 and t_2 can be coexecuted at a certain time and at a certain place.
- **POSSIBLE_OUTPUT(*t*, *d*).** This returns TRUE if one of the threads in *t* produces the data item *d*.
- **TASKSET_INPUT(*d*, t_1, t_2, …, t_k).** This returns TRUE if *d* appears in the input data set of at least one task in {t_1, t_2, …, t_k} but not in the output of any thread of tasks in {t_1, t_2, …, t_k}.

4.2 Sample queries

Some queries and the corresponding expressions from first-order predicate calculus are as follows:

- **Query scenario 2.** Find the names of tasks that can be executed after both “request proof” and “stamp” are completed in the “medical insurance reimbursement” process. This type of queries is used when the user is planning to perform some tasks and would like to know what he or she can do next. The corresponding query expression is shown below:

  {*t.name* | TASK(*t*), (*t.process.name* = ‘medical insurance reimbursement’), (∀*d*

$\in$ *t.input*, ((*d.status* = ‘available’) or ($\forall t_1$, TASK(t_1), (t_1.*name* = ‘request proof’), POSSIBLE_OUTPUT(t_1 , *d*)) or ($\forall t_2$, TASK(t_2), (t_2.*name* = ‘stamp’) and POSSIBLE_OUTPUT(t_2 , *d*))))};

- **Query scenario 3.** Find a set of tasks whose input data are available and can be coexecuted with “diagnose” in terms of time and place in the reimbursement process. This type of query is used when the user has planned to perform some task and would like to know what tasks can be potentially coexecuted at the same time and at the same place. The corresponding query expression is shown below:

  {*t.name* | TASK(*t*) , (*t.process.name* = ‘medical insurance reimbursement’), ($\forall d$ $\in$ *t.input*, ($\exists$ t_1 , TASK(t_1), (t_1.*name* = ‘diagnose’), (*d.status* = ‘available’ or POSSIBLE_OUTPUT(t_1 , *d*))), OVERLAP_TIME_PLACE(*t* , t_1))};

- **Query scenario 4.** Find the set of tasks that are ready to be executed at the current place and at the current time. This query might be raised frequently by a mobile user who likes to do know what he or she can do at any particular moment. The corresponding query expression is shown below (note that CURRENT is a system-defined dummy task that has the current time and the current place as its attribute values):

  {*t.name* | TASK(*t*), (*t.process.name* = ‘medical insurance reimbursement’), ($\forall d$ $\in$ *t.input*, (*d.status* = ‘available’)), (OVERLAP_TIME_PLACE (*t* , CURRENT))};

- **Query scenario 5.** Find the set of data that are required to complete tasks “request proof”, “diagnose”, and “stamp”. This type of query is especially useful when a user has decided to performed several task and would like to know what data items are required:

  {*d.name* | DATA(*d*), ($\exists t_1$, TASK(t_1), (t_1.*name* = ‘request proof’), ($\exists t_2$, TASK(t_2), (t_2.*name* = ‘diagnose’), ($\exists t_3$, TASK(t_3), (t_3.*name* = ‘stamp’),

TASKSET_INPUT(d, t_1, t_2, t_3)))) };

As can be seen, first-order predicate calculus represents a powerful method for querying definition and execution of a personal process. Therefore, any query language that fully implements first-order predicate calculus on objects with composite, multivalue attributes and allows user-defined functions can be used for querying personal processes. One such a language is SQL3 [EM99]. However, this paper focuses on proposing the concepts of personal process management and demonstrating the feasibility, and hence a full-fledged query language has not been implemented. Instead, in our prototype we enumerate several frequently used queries and implement them as parameterized queries. To execute these queries, a user only needs to click on some items or fill in forms that prompt the required parameters. The prototype is described in Section 6.

5 The system architecture

The objective of the personal process management system is to facilitate users to coordinate and track personal tasks with assistance from various organizations. To meet this objective, various functions have to be provided to support the design, execution, and query formulation of a personal process. Our proposed architecture involves three types of component: the template providers, the service providers, and the PWFMSs. The functions of each component are described below:

1. **The template provider** provides predesigned personal process templates for users with various backgrounds and requirements. After composing a particular template, a user can download the template using his or her handheld device

running a PWFMS.

2. **The service provider** is provided by an organization that executes users' personal tasks. Some of the tasks can be organized as a subprocess (e.g., the "diagnose" subprocess described in the insurance-reimbursement process), which is used by a template provider to compose a larger personal process. It also allows the status of a task to be tracked. Both personal subprocess publishing and task status tracking are conducted via Web services.
3. **The PWFMS** executes on a handheld device to manage personal processes. It is capable of downloading personal processes from template providers and interacts with service providers to keep track of the execution status of tasks. In addition, it also provides interfaces for a user to change task execution status and to place queries.

The interactions between the three types of components are depicted in Figure 7. A user first makes a personal process from a template. To tailor-make a personal process, the user is free to choose the desired personal subprocesses available from different service providers. The user then downloads the completed personal process to the PWFMS and may begin executing the constituent tasks. Once an (automatically tracked) task is executed, the associated service provider will notify the PWFMS of its execution status. All the interactions between different components are conducted via Web services. Note that Figure 7 shows a logical architecture. In practice, the template provider and the service provider can be of the same organization and integrated into a single system. Figure 8 shows the Web services that are implemented between components. Note that we do not consider user-authentication and security issues in this paper: these are important components in a real system, but they are peripheral to our design and ignored in this architecture for simplicity.

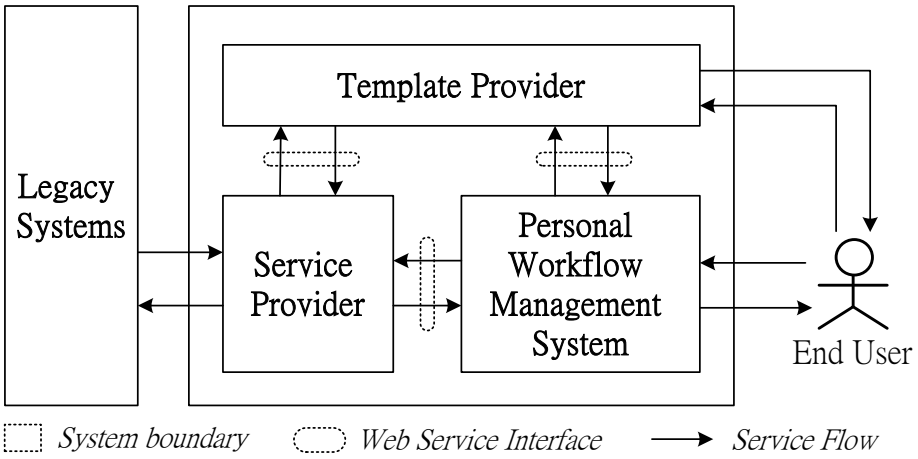


Figure 7: Logical architecture of the entire system

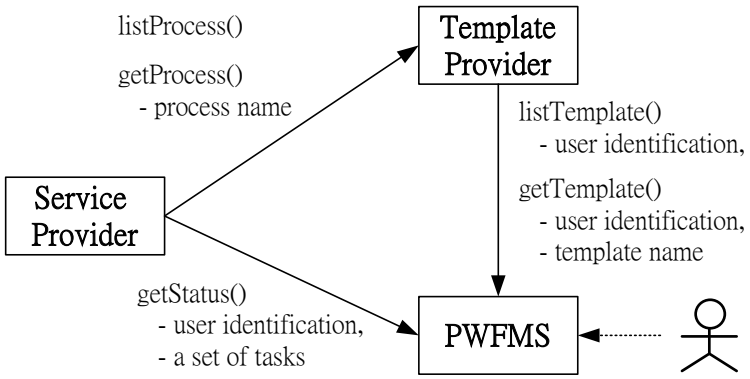


Figure 8: Interaction methods between components

5.1 Components and interfaces

In this section we provide a detailed description of the functions of each component (see Figure 9). Each component also provides several Web services for interacting with the other components depicted in Figure 8.

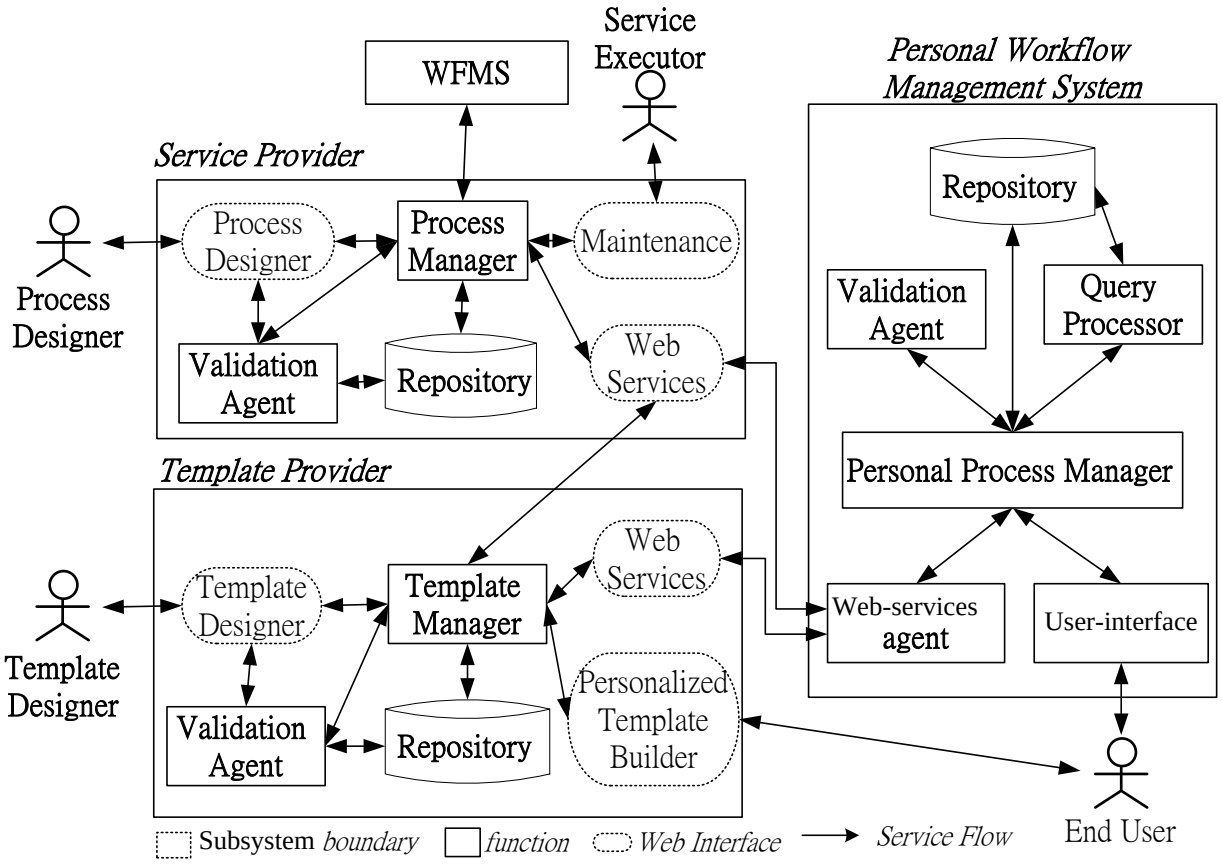


Figure 9: Subcomponents of the architecture

5.1.1 The service provider

An organization may install a service provider to facilitate the deployment and execution of tasks in personal processes. This service provider has two functions: (1) to provide a collection of (partial) personal processes, each consisting of a set of tasks for realizing a service, and (2) to monitor the execution status of tasks. To fulfill these two objectives, the service provider has three interface modules (enclosed within dotted circles in Figure 9): (1) the process designer, (2) the maintenance services, and (3) the Web services. The process-designer module provides a user interface that enables the user to create new personal processes; the maintenance module provides a user interface that assists a service executor in managing an executing task; and the Web-service module handles a set of published Web services, including listProcess,

getProcess, and getStatus (shown in Figure 8). An invocation of a Web service involves the interchange of two XML documents (one for service request and the other for service response). The content of the two XML documents required for executing each Web service is as follows:

1. listProcess: return the names of all personal processes stored in the service provider.

   Request: null

   <request>
   <process>ALL</process>
   </request>

   Response: names of available processes, whose XML format is shown below:

```
<response>
    <process>processName1</process>
    <process>processName2</process>
</response>
```

2. getProcess: return the definition of a given personal process.

   Request: a personal process name

   <request>
   <process> processName1</process>
   </request>

   Response: definition of the process

```
<process name=" processName1">
<task name="task1" priority="3" type="atomic" status="Initial"
description="test" provider="company A"
URL="http://cs.mis.nsysu.edu.tw/cgi-bin/service">
<place>school</place>
<time type="d" start="2003-05-05" end="2003-05-10"></time>
<inputdata type="primitive" status="unavailable">id1</inputdata>
<outputdata thread="success" status="unavailable">od1</outputdata>
</task>
<task…>…</task>
…
</process>
```

3. getStatus: return the execution status of a task.

   Request: process name and a set of task names

<request>

<process name=ProcessName1></process>

<task>name1</task>

<task>name2</task>

</request>

Response: the status (and the succeeded thread name, if any,) of each task

```
<response>
    <task name="name1" status="Logically-Completed"
thread=”success”></task>
    <task name="name2" status="Executing"></task>
</response>
```

The process manager is the core module of the service provider. It maintains the definition and monitors the executions of instances of each published personal process. In an organization that employs a workflow management system to provide its services, each task in a personal process may be regarded as a workflow, and the process manager has to interact with the workflow management system in order to update the task status. The definitions of personal processes as well as the execution status of each task instance are stored in the “repository”. The validation-agent module is responsible for checking the correctness of a personal process, as described in Section 3.

5.1.2 The template provider

The template provider allows designers to build personal process templates that comprise subprocesses (so-called nested tasks) and/or atomic tasks published by various service providers. It also allows users to customize a given template and to download the customized template to its PWFMS. Moreover, it provides three interface modules (enclosed within dotted circles in Figure 9). The template designer allows a designer to create new templates. Typically, a designer defines a personal process and creates several templates for it, each suited to a customer type with a

distinct background or requirement. The personalized-template-builder module allows a user to select a template that meets his personal requirements. Users interact with this module by providing their personal information as well as their requirements ([HC03] details these interactions). Note that the selected personal process templates will be stored in the repository for later access (e.g., downloading). The Web-service module handles a set of published Web services, namely listTemplate and getTemplate. Their functions and the invocation formats are as follows:

1. listTemplate: return a set of personal process templates selected by a given user.

   Request: user's identifier (name)

   <request>
   <user>name1</user>
   </request>

   Response: a set of names of customized personal process templates

```
<response>
    <template>template1</template>
    <template>template2</template>
</response>
```

2. getTemplate:

   Request: requires user identifier (name) and names of personal process templates

   <request>
   <user>name1</user>
   <template>template1</template>
   </request>

   Response: a personal process (customized template)

```
<template name="template1">
<task name="task1" priority="3" type="atomic" status="Initial"
description="test" provider="company A"
URL="http://cs.mis.nsysu.edu.tw/cgi-bin/service">
<place>school</place>
<time type="d" start="2003-05-05" end="2003-05-10"></time>
<inputdata type="primitive" status="unavailable">id1</inputdata>
    <outputdata thread="success" status="unavailable">od1</outputdata>
```

```
</task>
</template>
```

The validation agent is responsible for validating the correctness of a template constructed by the template designer, as described in the service provider.

5.1.3 Personal workflow management system

The PWFMS is the focal point of the entire architecture: it enables users to create and obtain a new personal process (from the template provider), to manage existing personal processes, and to interact with the service providers of organizations that are responsible for executing tasks. The Web-services agent module is responsible for interacting with the template provider (e.g., for retrieving and downloading personal processes by issuing listTemplate and getTemplate Web services, respectively) and with the service provider (e.g., for getting the execution status of a task by issuing the getTaskStatus Web service).

The personal process manager is the core module of the PWFMS that provides its main functionality, including process/task status control and task invocation control. When the definition of a personal process is modified or the status of a task is changed (either by the user or by the Web service getTaskStatus), the correctness of the resultant personal process has to be checked by the validation agent. The query-processor module supports basic queries as well as frequently used queries, as described in Section 4.

As mentioned in Section 1, an important goal of managing personal processes is to provide reminders or suggestions to a mobile user, rather than to enforce task executions as does a commercial workflow management system. Thus, while the

PWFMS may automatically update the status of an executing task via Web services, a physical task must be invoked by the user (e.g., hand in the required papers when applying for a certain document). Once this is done, the organization that executes the task will send the update of the task status to the PWFMS via Web services (e.g., executing, logically completed, or failed). The completion of a task may in turn ready another task for execution. The user of the PWFMS may use the query interface to decide which task to engage next. The above procedure is performed iteratively.

5.2 Workflow behavior of a personal process

Figure 10 shows a UML activity diagram that describes the workflow behavior of a personal process. When a personal process is started, this will set the process status as “executing” and the status of primitive input data items as “available”. The user may then place a query to choose the set of unexecuted tasks, among which one task is chosen for execution. Sometime after a task begins executing, a task status update event may arrive. This event may be either initiated manually by the user (for a manually tracked task) or automatically by an organization via Web services (for an automatically tracked task). The PWFMS then updates the corresponding task status accordingly. If a task status is changed to “logically completed”, the correctness of the resultant personal process has to be reevaluated. If it turns out that the personal process is incorrect, and the user does not want to update the definition of the personal process, the process is designated as “failed”. Conversely, the user may update the personal process definition and have the PWFMS recheck its correctness. If a task status is changed to “failed”, the user will examine the task and decide whether or not to reset its status to “initial” for possible reexecution. When the process is correct and all the target data items are available, the personal process is considered to have

succeeded.

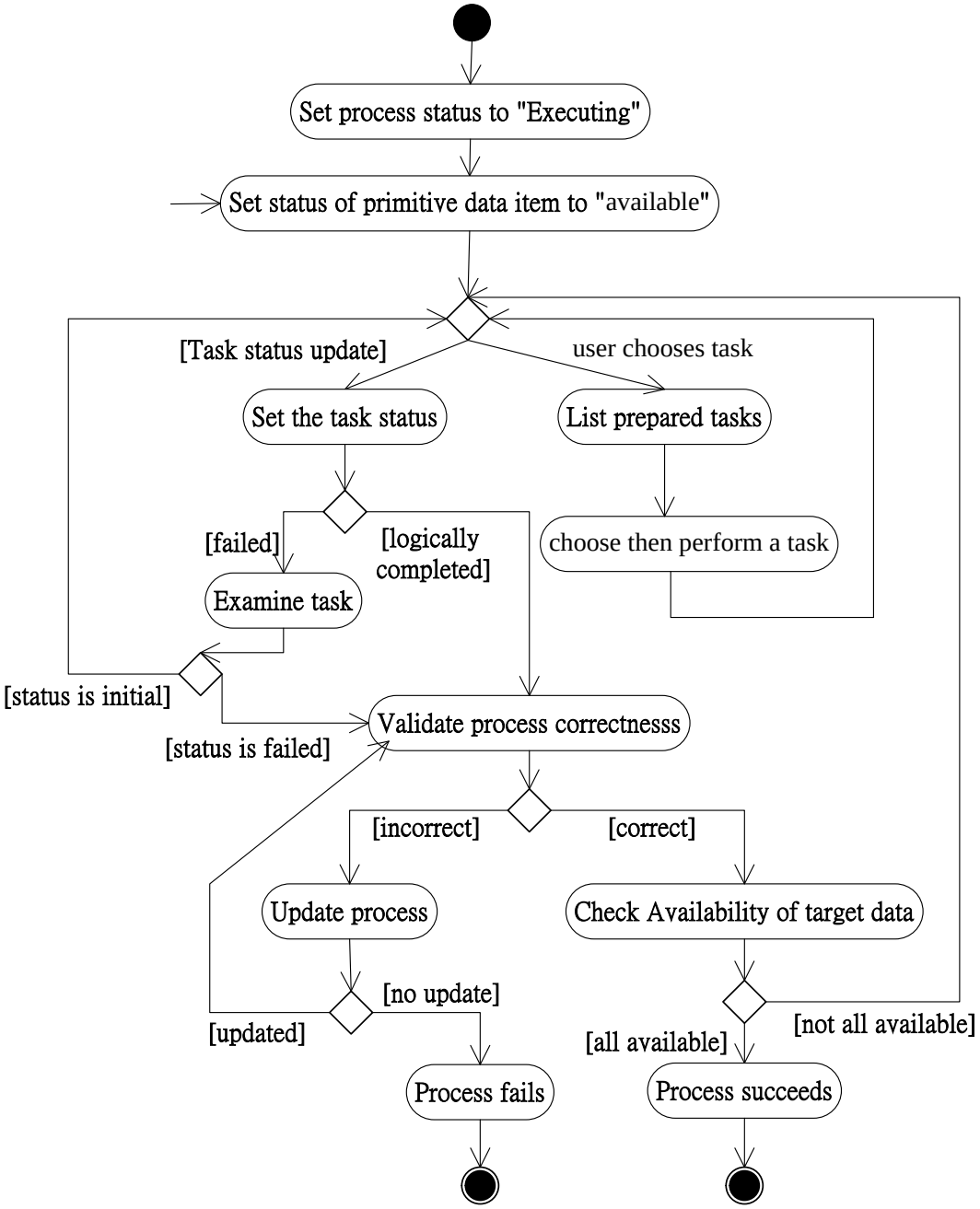


Figure 10: Activity diagram of a personal process

6 The prototype

We implemented the three required components to prove our concept of the proposed personal process model and the underlying architecture: the service provider, the

template provider, and the PWFMS. To facilitate the implementation, we have simplified the proposed model and accordingly also the architecture. First, time and location are separate attributes in our implementation, whereas in our model they are integrated as a composite, multivalued attribute. Second, due to the lack of positioning devices and services available to our environment, we ignore position-related query statements (therefore, querying which tasks can be executed at a user's current position will simply return all the tasks available).

Both the service provider and the template provider run on servers located on a fixed network. The service provider is implemented using Perl and executes on an Apache Web server. The template provider is a Web-based system using Apache as the Web server and MySQL as the underlying DBMS. PHP is used to implement the Web interface and the Web services. The PWFMS executes on handheld devices and is implemented using J2ME-MIDP (Java 2 Mobile Information Device Profile) as the development tool. The Appendix shows several screenshots of the prototype system, detailed descriptions of which are provided in Sections 6.1 and 6.2.

6.1 The template provider system

When a user logs onto the template provider, he or she is prompted with a set of available templates and a set of customized templates that he or she has built previously. Figure A-1 shows a screenshot of the template provider that appears when a user logs on. The upper screen shows that two personal process templates ("reimbursement" and "vehicleRegistration") are available to the user, and the lower screen shows that the user has constructed one template ("skin"). When the user chooses to customize a template, he or she will be prompted to choose the service

provider for each partial personal process in the template. Figure A-2 shows a screenshot for when the user chooses to customize the template “reimbursement”, in which there are two tasks, namely “insurance certificate” and “diagnosis certificate”, each of which can be provided by two service providers. The user may then further customize the template by following the system guide (see Figure A-3). For example, the user can add new tasks (see Figure A-4) or modify existing tasks (see Figure A-5). After the customization, the user can define the target data set and verify the correctness of the personal process (see Figure A-6). The system also provides a utility for the user to view the data dependency between tasks, i.e., the relation $\prec_d$ described in Section 3 (see Figure A-7). Finally, the user can export the template so that it can be downloaded to the PWFMS (see Figure A-8). An exported personal process template can be retrieved by a PWFMS via the Web service interface (explained in Section 5.1.2).

6.2 Personal workflow management system

The PWFMS can run on a handheld device. It assists users in the management of personal processes by allowing them to download a personal process template (from the template provider), to browse the definition, to place queries, and to automatically track the status of an executing task (from a service provider). Figure A-9(a) shows the main menu of the PWFMS. When the user chooses “select process”, a list of personal processes in the PWFMS is displayed (see Figure A-9(b)). When the user chooses one personal process from the list, the PWFMS will check the status of each executing task of that personal process by invoking the appropriate Web services. If the status of any task is updated (see Figure A-10(a)), the correctness of the personal process is validated automatically. A warning message is displayed if the PWFMS

finds that the personal process is incorrect. The user can browse and modify a personal task by clicking on it (see Figure A-10(b)).

When the user wishes to download a new personal process from the template provider, he or she must supply the user ID, template ID, and URL (see Figure A-11(a)). The PWFMS will automatically invoke the corresponding Web service to download the definition of the customized personal process from the template provider.

The PWFMS provides several parameterized functions for manipulating the tasks of a personal process (by clicking the left anchor on the “task list”), as shown in Figure A-11(b). These functions include the placement of basic queries and frequently used queries as described in Section 4, and other maintenance functions. These functions are described below:

1. **List available tasks.** This displays the names of the tasks whose input data items have the status “available”.
2. **List unexecuted tasks.** This displays the names of the tasks whose status is “initial”.
3. **List executing tasks.** This displays the names of the tasks whose status is “executing”.
4. **List all tasks.** This displays the names of all tasks.
5. **Task status change.** This function allows the user to explicitly specify the execution status of tasks. Users require this function for updating the status of manually tracked tasks. Although the execution status of automatically tracked tasks will be updated automatically via Web services, the user may sometimes need to use this function to change their execution status (e.g., from “logically

completed" to "physically completed", or from "failed" to "initial"). When the task status is changed to "logically completed" and there is more than one thread, the PWFMS will guide the user to select a succeeded thread.

6. **Task required data.** This allows the user to select a set of tasks and list the aggregated input data items of these tasks.
7. **Task related data.** This displays the names of tasks that can be executed after a given set of tasks is completed (query scenario 2 in Section 4.2).
8. **Co-executed tasks.** This displays the names of the tasks whose input data are available and can be coexecuted with a given task in terms of time and place (query scenario 3 in Section 4.2).
9. **Available tasks (time).** This displays the names of the tasks that can be executed immediately (query scenario 4 in Section 4.2).
10. **List input data.** This displays the names of the data items that are required to complete a given set of tasks (query scenario 5 in Section 4.2). An example of the output is shown in Figure A-12(a).
11. **Query date.** This displays the names of tasks that can be executed on a specific date or within a date interval (see Figure A-12(b)).
12. **Task detail.** This displays the detailed information of a selected task.

Finally, the PWFMS provides the following functions for manipulating data items (by clicking the left anchor on "show data list"), as shown in Figure A-13:

1. **List available data.** This lists all available data items.
2. **List unavailable data.** This lists all unavailable data items.
3. **List all data.** This lists all data items.
4. **Status change to available.** This changes the status of a given data item to

"available".

5. **Status change to unavailable.** This changes the status of a given data item to "unavailable".
6. **Target data set.** This allows the user to manipulate the target data items.
7. **Data related tasks.** This shows the names of the tasks that require a given data item as input (query scenario 1 in Section 4).

7 Related work

The development of workflow management systems aimed at automating business processes within an enterprise began in the early 1990s and has now reached a mature stage. Many proprietary workflow management systems (e.g., IBM's Lotus Workflow [Lotu03] and Ultimus's Workflow Suite [Ulti03]) and some ERP systems (e.g., SAP [SAP03] and PeopleSoft [Peop03]) capable of managing intraorganizational business processes are available on the market. The shift in the focus of research and development from intra- to interorganizational workflow began in the late 1990s and has resulted in the emergence of many research types (e.g., CMI [Schu00], Mentor-Lite [Weis00], Sagitta2000 [Aals99], SELF-SERV [SBDM02], and WISE [LASS00]) and products (e.g., RosettaNet [Roset03], Biztalk [Bizt03], and ebXML [ebXM03]). A long list of the related standards and products can be found in [OASI03]. The main goal of interorganizational workflow management systems is to support B2B commerce, and they must provide all of the following functions [Medj03, DHL01]:

1. A language model and the associated mechanism for specifying a common business process.

2. A mechanism for an organization to publish its abilities in participating in specific roles of a common business process.
3. A mechanism for an organization to interact with another for collaboratively executing a common business process.
4. A common format for documents to be exchanged between organizations.

Early efforts proposed diverse ways for addressing the above issues, whereas more recent work has tended to adopt XML and Web services in the implementation of these requirements. As such, documents to be exchanged between organizations must be specified in XML according to some common catalog, organizations must publish their services and interact with peer organizations via Web services, and the common business process is specified using some Web service composition language (such as those proposed by WSFL [Leym01], XLANG [That01], and BPEL4WS [Curb03]).

The present work differs significantly from the huge body of previous work on interorganizational workflows: others have aimed to automate an interorganizational business process, whereas our aim was to facilitate a mobile user in handling personal business across several organizations that require physical interactions from customers. Therefore, we do not require that a task (or a subprocess) is instantiated as a Web service from another organization. Instead, Web services are mainly used to publish a task, to track a task's execution status, and to download a process template. This approach dramatically reduces the administration effort, preserves autonomy, and still allows for interoperations between organizations.

One project that appears similar to ours is ServiceFlow, which proposes focusing on the customer-centered aspect on workflow management [KW01a, KW01b, KW02,

WK02]. ServiceFlow introduces a general concept for supporting interrelated, personalized, and localized services performed across different organizational units or provider firms. As in our work, a process is mainly designed to handle a personal business, such as a patient's medical treatment. A process within ServiceFlow is modeled as a series of service points (denoted as a list), each including a UML specification of participants carrying out certain tasks/activities as well as the pre- and postconditions required for process execution. However, although ServiceFlow proposes the consideration of customer-related factors, the procedures for publishing, instantiating, and tracking tasks are basically the same as those proposed by the conventional B2B workflows described earlier in this document.

8 Conclusions

In this paper we have exploited the user-centered aspect of interorganizational workflows and proposed the concept of *personal processes*. A personal process is defined as a coordination of personal tasks, each requiring a joint effort between a user and an enacting organization so as to achieve a personal goal. We have identified the unique requirements for managing personal processes. To satisfy these requirements, we formally defined a personal process model, a correctness criterion, and the query expressions. A lightweight architecture for systematically supporting the management of personal processes has also been presented. We have also detailed our implementation of a prototype system that includes a PWFMS running on a Palm PDA and two subsystems running on fixed networks.

One function that is particularly useful in managing personal processes is the *alert* (or *recommendation*) function. The alert function of a PWFMS advises the user of the

correct things to do at a particular time and place. This function was left out in our current prototype because location servers that are capable of providing accurate location information about mobile users in a wide range of regions (e.g., both indoors and outdoors) are not widely available. However, with the current rapid developments in wireless technologies, accurate and inexpensive location services will soon become available. Our future work includes the exploration of issues and solutions related to the design of the alert function.

References

[Aals99] W. M. P. Van Der Aalst, "Process-oriented Architectures for Electronic Commerce and Interorganizational Workflow," *Information Systems*, 24(8), 1999, pp. 639–671.

[BPMI03] Business Process Management Initiative, http://www.bpmi.org, September 2003.

[Bizt03] Biztalk, http://www.microsoft.com/biztalk/, September 2003.

[BM77] J. L. Bell and M. Machover, *A Course in Mathematical Logic*, North-Holland Publishing Company, 1977.

[Curb03] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana, "Business Process Execution Language for Web Services V1.1," http://dev2dev.bea.com/techtrack/BPEL4WS.jsp, September 2003.

[DHL01] U. Dayal, M. Hsu, and R. Ladin, "Business Process Coordination: State of the Art, Trends and Open Issues," *Proceedings of the 27th Very Large Databases Conference (VLDB 2001)*, Rome, Italy, 2001, pp. 3–13.

[ebXM03] ebXML, http://www.ebxml.org/, September 2003.

[EM99] A. Eisenberg, and J. Melton, “SQL:1999, Formally Known as SQL3,” *ACM SIGMOD Record*, 28(1), 1999, pp.131–138.

[Guet00] R. H. Gueting, M. H. Boehlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. S., and M. Vazirgiannis, “A foundation for representing and querying moving objects,” *ACM Transactions on Database Systems*, 25(1), 2000, pp.1-42.

[HC03] S.-Y. Hwang, Y.-F. Chen, “Personal Processes: Modeling and Management,” *4’th Int’l. Conf. on Mobile Data Management (MDM03), Melbourne, Australia*, Jan. 2003. LNCS2574, Springer Verlag.

[KW01a] R. Klischewski, I. Wetzel, “Modeling Serviceflow,” *In: Godlevsky, M., Mayr, H. (ed.): Information Systems Technology and its Applications. Proceedings ISTA 2001. Bonn: German Informatics Society*, 2001, pp.261-272.

[KW01b] R. Klischewski, I. Wetzel, “Serviceflow Management for Health Provider Networks,” *in: Information Age Economy. Proceedings 5th International Conference Wirtschaftsinformatik (Business Information Systems)*, Heidelberg, 2001, pp.161-174.

[KW02] R. Klischewski, I. Wetzel, “Serviceflow Management: Caring for the Citizen’s Concern in Designing E-Government Transaction Processes,” *Proceedings Hawaii International Conference on System Sciences (HICSS-35), IEEE*, 2002.

[LASS00] A. Lazcano, G. Alonso, H. Schuldt, and C. Schuler, “The WISE Approach to Electronic Commerce,” *Journal of Computer System Science and Engineering*, 15(5), 2000, pp. 343–355.

[Leym01] F. Leymann, "Web Services Flow Language (WSFL 1.0)," http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, September 2003.

[Lotu03] IBM Lotus Workflow, http://www.lotus.com/products/domworkflow.nsf, September 2003.

[Medj03] B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. H. Ngu, and A. K. Elmagarmid, "Business-to-business Interactions: Issues and Enabling Technologies," *The VLDB Journal*, 12, 2003, pp. 59–85.

[OASI03] OASIS, "Business Process Management and Choreography," http://xml.coverpages.org/bpm.html, September 2003.

[Peop03] PeopleSoft Inc., http://www.peoplesoft.com, September 2003.

[Roset03] RosettaNet, http://www.rosettanet.org, September 2003.

[SAP03] SAP Inc., http://www.sap.com, September 2003.

[SBDM02] M. Shen, B. Benatallah, M. Dumas, and E. Mak, "SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-peer Environment," *Proceedings of the International Conference on Very Large Databases*, Hong Kong, China, 2002, pp. 1051–1054.

[Schu00] C. Schuster, D. Baker, A. Cichocki, D. Georgakopoulos, and M. Rusinkiewicz, "The Collaboration Management Infrastructure," *Proceedings of the IEEE International Conference on Data Engineering*, San Diego, Calif., USA, 2000, pp. 677–678.

[That01] S. Thatte, “XLANG: Web Services for Business Process Design,” http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm, September 2003.

[Ulti03] Ultimus Workflow Suite, http://www.ultimus.com/ultintro.htm, September 2003.

[Weis00] J. Weissenfels, M. Gillmann, O. Roth, G. Shegalov, and W. Wonner, “The Mentor-lite Prototype: a Lightweight Workflow Management System,” *Proceedings of the IEEE International Conference on Data Engineering*, San Diego, Calif., USA, 2000, pp. 685–686.

[WfXM01] Workflow Management Coalition (WfMC), “Wf-XML Binding,” http://www.wfmc.org/standards/docs/Wf-XML-11.pdf, September 2003.

[WK02] I. Wetzel, R. Klischewski, “Serviceflow beyond Workflow? Concepts and Architectures for Supporting Inter-Organizational Service Processes,” In *Proc. of 14th CAiSE (International Conference on Advanced Information Systems Engineering)*, Springer, Berlin, 2002, pp.500-515.

Appendix Screenshots of the prototype system

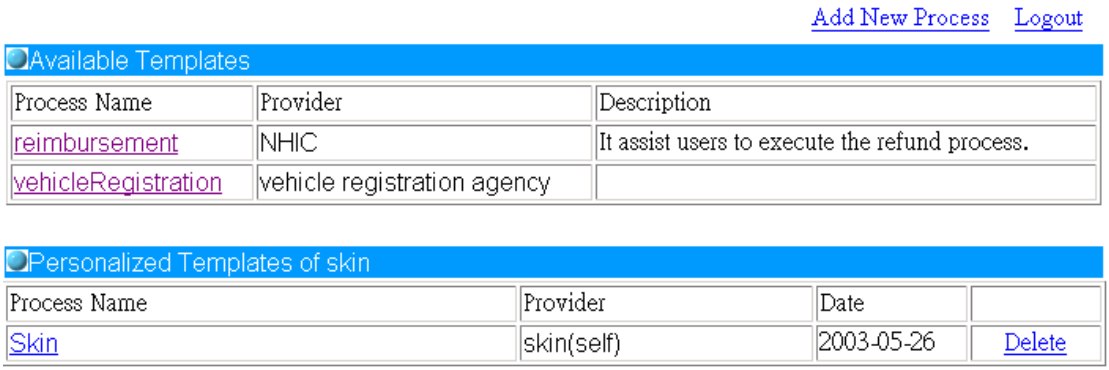


Figure A-1: Template provider – list processes

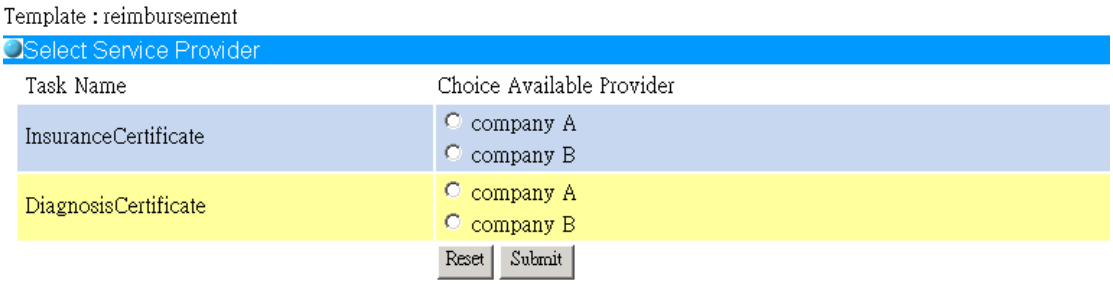


Figure A-2: Template provider – choose providers

Figure A-3: Template provider – manage a personal process

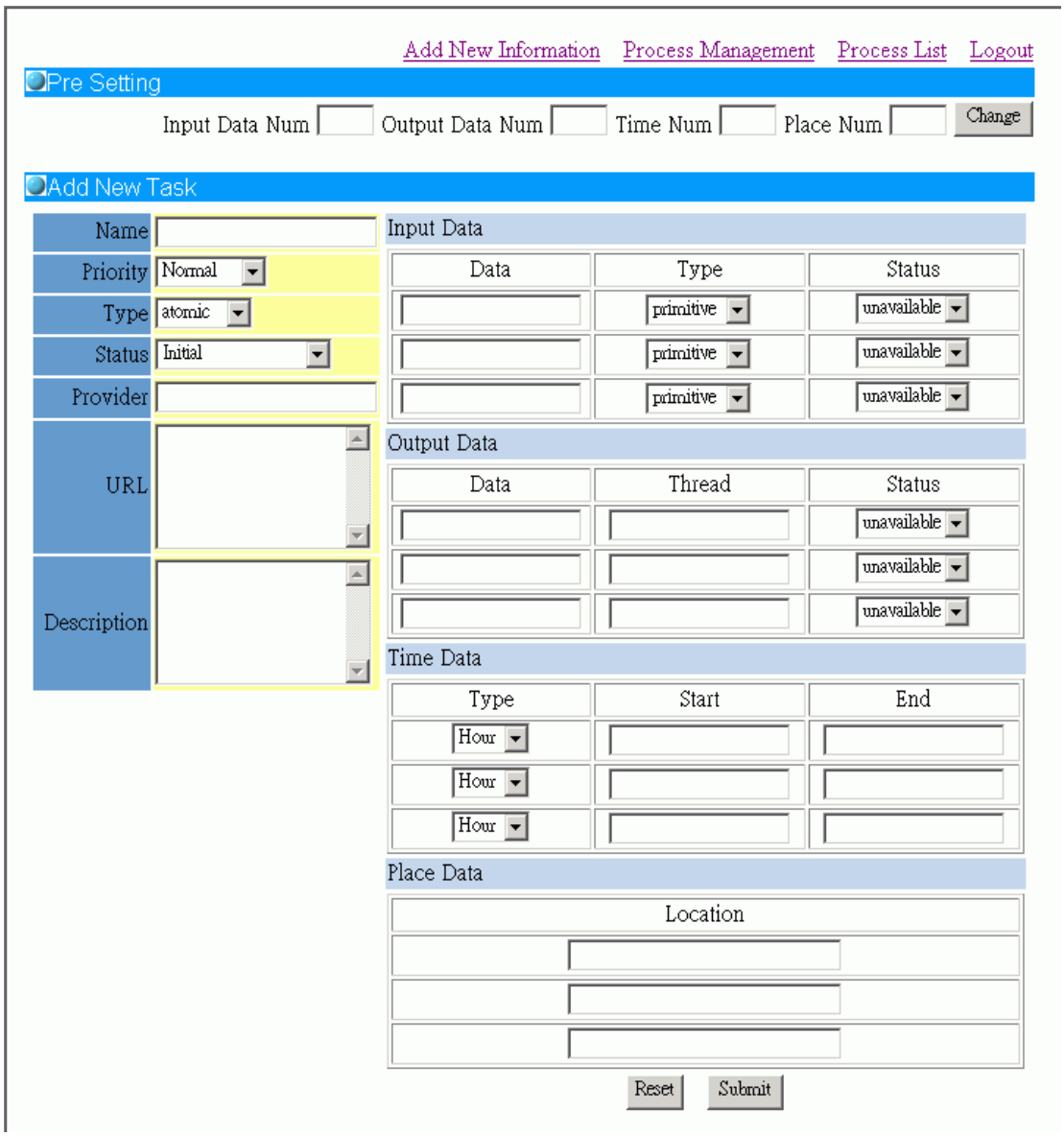


Figure A-4: Template provider – add a task

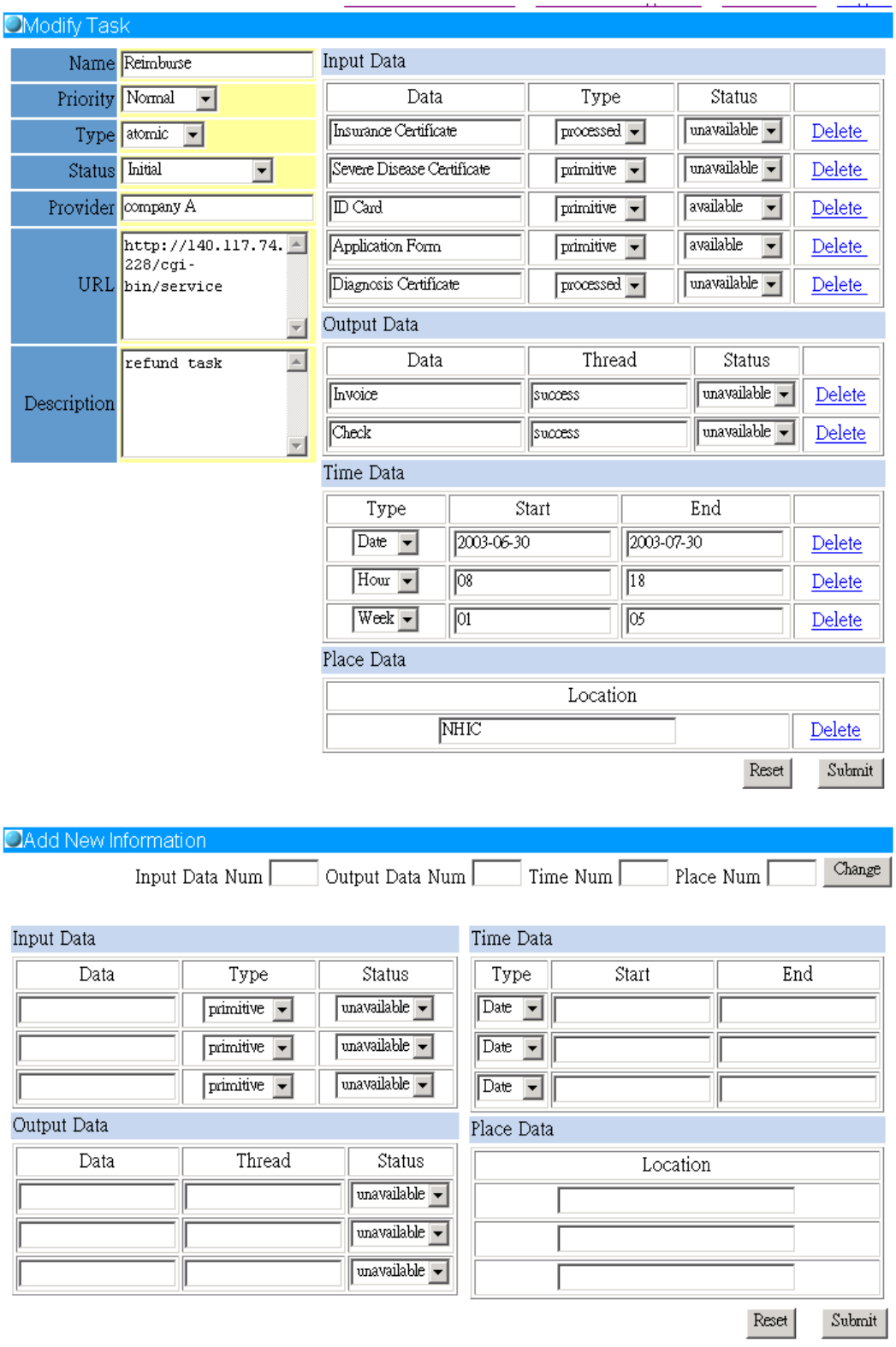


Figure A-5: Template provider – modify a task

Figure A-6: Template provider – check correctness

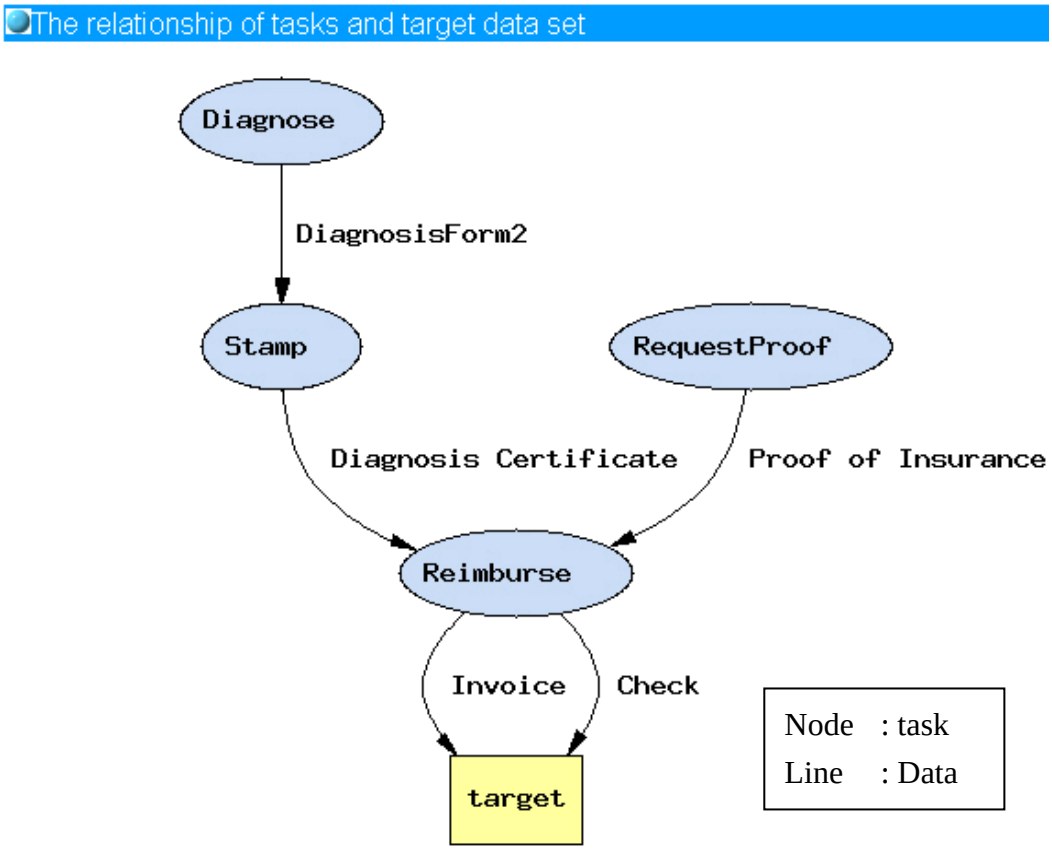


Figure A-7: Template provider – personal process diagram

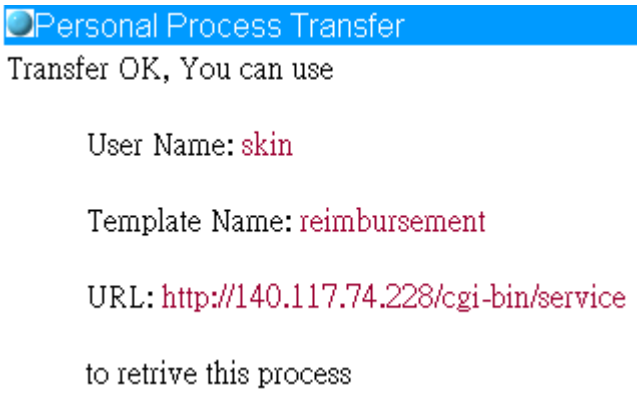
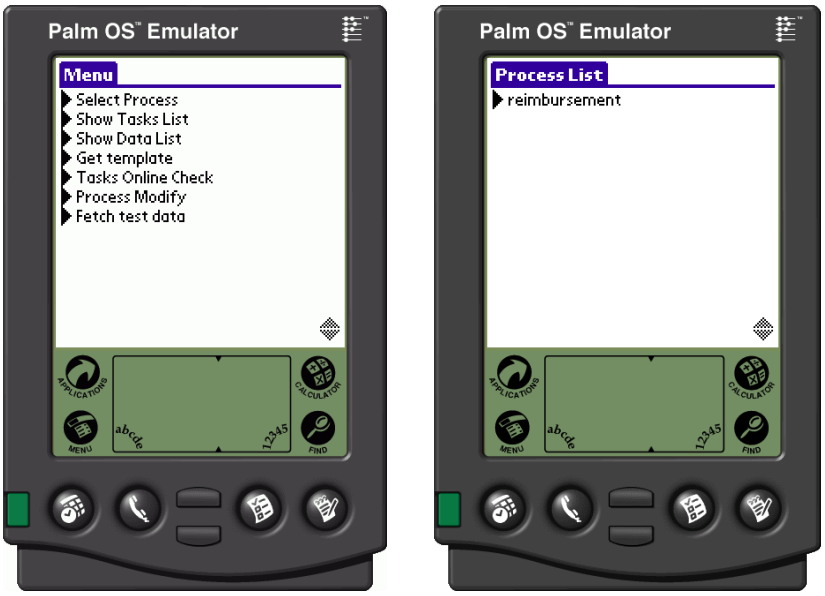
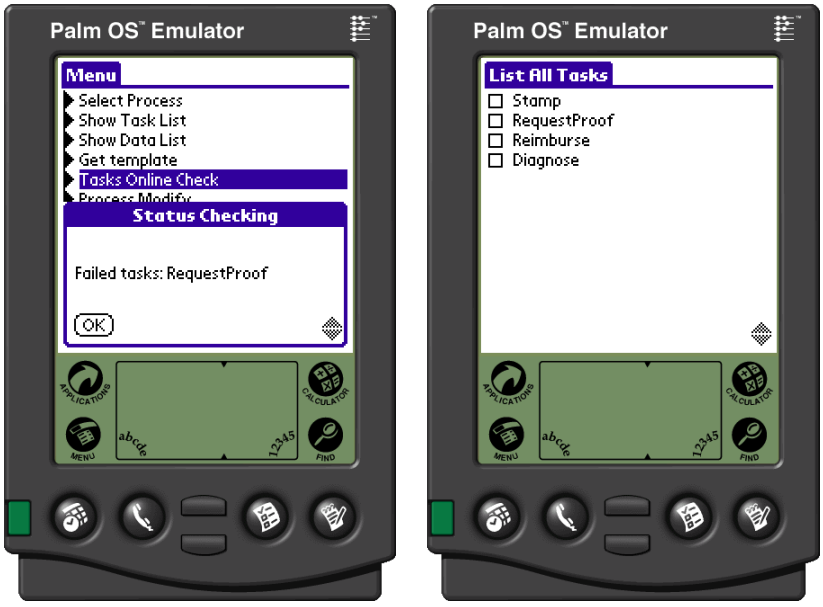


Figure A-8: Template provider – process a transfer



(a) (b)

Figure A-9: (a) PWFMS menu; (b) PWFMS process list

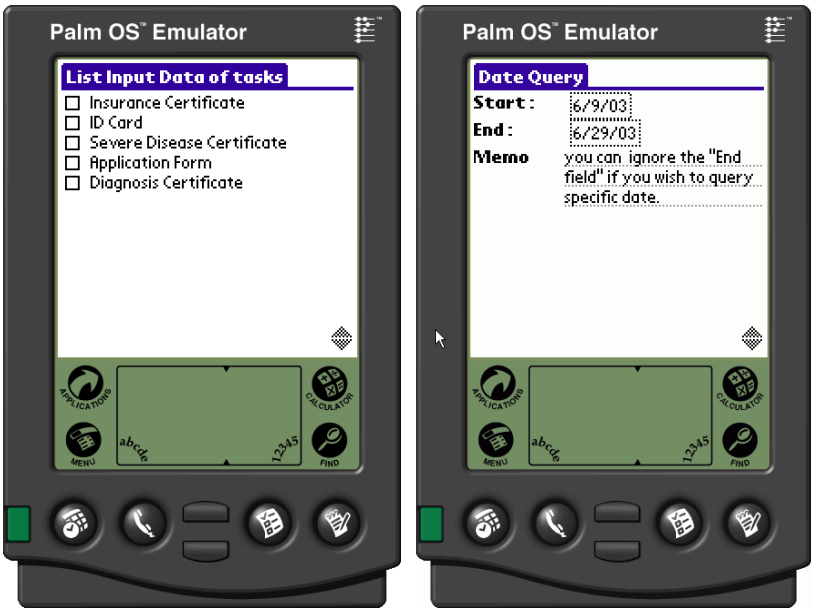


(a) (b)

Figure A-10: (a) PWFMS response message to an online task status check; (b) PWFMS task list

(a) (b)

Figure A-11: (a) PWFMS getTemplate function; (b) PWFMS task functions



(a) (b)

Figure A-12: (a) PWFMS list data; (b) PWFMS query date

Figure A-13: PWFMS functions for manipulating data