整合跨組織企業流程之個人化工作流程系統之研究

Personal Workflow Systems in Support of Inter-process Integration

研究生：涂俊凱撰
指導教授：黃三益 博士

中華民國九十二年七月
中文摘要

日常生活中，人們經常需要應付各種可能的狀況，對於陌生的狀況，使用者常常會多做或少做了某些事，或無法在預期的時間內完成。其中有一項很重要的原因，即事件之間經常存在著部份的相關性，可稱之為程序導向，例如：A 工作需要 B 工作的資料，或是 A 工作需要 B 做完後才能進行。在[Chen01]，作者 Chen 提出個人工作流程的概念和系統的設計，然而該研究並不支援企業組織所提供的服務。因此，在本論文中我們提出了一個模式，除了一般個人工作流程的功能，也允許組織以服務提供者的角色提供部分流程，並利用 Web services 與個人工作流程系統互動。在使用上，我們提出一個便於使用者應用個人工作流程的查詢模式，並提出一個架構來幫助使用者，在此架構中包含了服務提供者與樣版提供者，服務提供者提供實際的服務給使用者，樣版提供者是以使用者的目標為導向，提供流程樣版給使用者。對於我們所提出的模式與架構，我們實際建置了一個雛形系統 (PWFMS) 來驗證其實作性與可行性。
Abstract

In our daily lives, people constantly need to schedule their activities to meet their personal goals. Many of these activities involve the interaction with organization sectors, which must follow certain regulations in terms of input data, doable time and places. These regulations form personal processes. In previous work, [Chen01] proposed a personal process model that define a personal process as a set of tasks and a coordination on the tasks and a personal workflow system architecture. However, the proposed system is solely to facilitate a mobile user in deciding when, where, and how to process a task. We have noticed that many processes initiated by an individual often involve business processes coming from different organizations. Our objective in this thesis is to revise the personal process model proposed in [Chen01] by incorporating business processes existing in different sectors, developing a simpler query method, and extending personal workflow architecture in support of the new model. In our architecture, the management of personal processes involves three parties: the service provider, the template provider and the PWFMS. The service providers supply personal processes, each of which enable users to achieve a specific goal, and keep track of the status of executed tasks. The template providers incorporate several personal processes from different service providers and form a useful template for a user. The PWFMSs assist users in managing personal processes. In order to prove our idea, we implemented a prototype of the architecture.
List of figures

Figure 1-1: Vehicle registration process ............................................................... 3
Figure 1-2: Insurance reimbursement in Taiwan .................................................... 4
Figure 1-3: Execution of (a) an inter-organizational process (b) a personal process .... 5
Figure 2-1: Service point .................................................................................... 13
Figure 2-2: Model of the case serviceFlow .......................................................... 13
Figure 2-3: A Framework for systems supporting business process management
   (source: [DHL01]) ......................................................................................... 15
Figure 2-4: A scenario collaborative process management (source: [CH01]) ............ 17
Figure 3-1: Status transition diagram of a task .................................................... 25
Figure 3-2: Task threads .................................................................................... 26
Figure 3-3: Meta schema of the personal process ................................................. 27
Figure 5-1: Logical architecture of the entire system .......................................... 37
Figure 5-2: Global architecture ........................................................................... 38
Figure 5-3: Interaction method ............................................................................ 38
Figure 5-4: Activity diagram of a personal process .............................................. 44
Figure 6-1: Template provider-list processes ...................................................... 47
Figure 6-2: Template provider-choice providers ............................................... 47
Figure 6-3: Template provider-manage a personal process .................................. 48
Figure 6-4: Template provider-add task .............................................................. 49
Figure 6-5: Template provider-modify task ........................................................ 50
Figure 6-6: Template provider-correctness check ............................................... 51
Figure 6-7: Template provider-personal process diagram .................................. 51
Figure 6-8: Template provider-process transfer ............................................... 51
Figure 6-9: (a) PWFMS-menu (b) PWFMS-process list ...................................... 54
Figure 6-10: (a) PWFMS-error message of online check (b) PWFMS-task list ....... 55
Figure 6-11: (a) PWFMS-getTemplate (b) PWFMS-functions of task ................ 55
Figure 6-12: (a) PWFMS-list data (b) PWFMS-query date ................................. 56
Figure 6-13: PWFMS-functions of data list ....................................................... 56
Chapter 1 Introduction

1.1 Background

With the rapid development of communication and mobile device techniques in recent years, many hand-held devices equipped with communication capabilities, such as personal digital assistants (PDAs) and cellular phones, are extensively available on the market. People often use these devices to communicate with other people, to schedule their daily lives, to store their personal data, to access resources on the Internet, to play games, and so on. On one hand, mobile devices are widely used by individuals to handle personal things through, for example, calendars, memos, address books, to-do-list, and to access to proprietary databases on the Internet. On the other hand, hand-held devices are also used in enterprises. For example, salesmen of insurance companies use hand-held devices to assist in selling insurance products, and servicemen of hand-held devices use PDAs to record customers’ dish orders.

1.2 Motivation

We have observed that people’s daily activities are often not independent, and they are likely to be process-oriented. In the context of business activities, workflow management systems (or called business process management system [DHL01]) have been used in many enterprises to automate their crucial business processes. A workflow management system separates the specification of business processes, or so-called workflow types, from their execution and provides a convenient and powerful means of specifying a business process and controlling its executions. A workflow type must have well-formed structures and, once specified, are often instantiated many times. In contrast, a personal process is a coordination of a set of
personal tasks so as to achieve a personal goal [HC03]. Here we define a personal task as an endeavor provided by the owner of the personal process that usually involves the interaction with a third party in the outside world. Consider the following two examples. The first example is a vehicle registration process, which is a coordination of a set of personal activities needed for registering a vehicle which moves from one state to another in US. The second example is an insurance reimbursement process that is used by an individual who needs to apply the insurance reimbursement for National Health Insurance Program in Taiwan.

The vehicle registration personal process
A car owner David moves from Minnesota State to Pennsylvania State and wishes to register his car in the new state. When planning the relocation, David realizes that he can get a better insurance policy in Pennsylvania. Thus, the entire car registration process that suits David’s situation involves four tasks: obtain a new insurance policy, cancel the old policy, register car at the Pennsylvania’s vehicle registration agency, and send car plate back to the Minnesota’s vehicle registration agency (to cancel his car registration). The entire process is shown in Figure 1, where a task is represented as an edge (arc) that transform a set of input data to another set of output data, each represented as a circle. Obviously, each task requires the joint effort of David and an enacting organization, and the execution order of tasks is determined by the input and output data needed for executing tasks. As a matter of fact, each task may indeed be a business process within the enacting organization. However, from the viewpoint of David, if the execution of the business process does not need to involve him, the details of the business process are transparent, and only its input and output matter. On the other hand, if a business process needs an individual to accomplish a certain steps, more detailed steps have to be made explicit in the
personal process. For example, the task Get policy may not be a simple task and may involve several subtasks to be performed by David and therefore is not a simple task. Besides, different sets of insurance companies may require different input data and produces different output data. This template, which may be originally provided by the Pennsylvania vehicle registration agency for the convenience of its customers, is subject to change as needed by the enacting organizations and the user.

Figure 1-1: Vehicle registration process

**The insurance reimbursement personal process**

This example shows the reimbursement process for the National Health Insurance Program executed by the Bureau of National Health Insurance (BNHI) in Taiwan. The main task in this process is to execute “Reimburse”, which needs an individual to bring a set of documents, including Insurance Certificate, Diagnosis Certificate, and Severe Disease Certificate to a BNHI agency for application. However, each of the documents has to be given by different organizations, including the employer that insures the applicant and the hospital that provides medications to the applicant. In fact, to receive a Diagnosis Certificate, many hospitals require the applicant to first fill in an application form, have the doctor fill in the diagnosis details, and then pass to the authority for a final stamp (signature), all initiated by the applicant. This
process is shown in Figure 1-2. Note that obtaining a Severe Disease Certificate also requires another process, which is eliminated here for brevity.

![Diagram](image.png)

**Figure 1-2: Insurance reimbursement in Taiwan**

It is obvious that supporting the two scenarios described above demands new functionalities for system supports. In the context of inter-organizational workflows, enterprises that involve in a workflow must reach some sort of agreement in advance in order to achieve automatic interaction between two adjacent activities. The procedure can be shown in Figure 1-3(a). However, enabling interaction between activities from different organizations require substantial effort in both administration and computer systems. And it is unlikely for two unrelated organizations to communicate with each other systematically just to achieve a customer's personal goal. In contrast, we advocate the personal process model, as depicted in Figure 1-3(b), for supporting personal processes. In such a model, the only communications involved in executing a personal process are between the user and organizations. Organizations do not talk to each other. This model does not only saves the substantial cost for enterprise cooperation but also preserves user privacy.
Figure 1-3: Execution of (a) an inter-organizational process (b) a personal process

We observe that the management of personal processes possesses the following requirements:

1. For a given user, a personal process can be executed for only a limited number of times (and only once in most cases). However, many different users may share a common the same interest on a particular personal process. Therefore, it is imperative for an organization to provide the personal processes that are useful to its customers. In other words, there is a need to develop a personal process template provider.

2. Personal tasks are primarily related by their executable times, executable places, and input and output data. Unlike enterprises that impose many regulations and procedures that need to be followed strictly by their processes, human beings seldom impose rigid rules on the execution order of their personal tasks. As a result, the specifications of control flow constructs such as or-split, and-split, or-join, and and-join, are rarely needed. Thus, a personal process model that comprises executable times, executable places, and input and output data of tasks need to be developed.
A personal task, in most cases, is an interaction between a user and an enacting organization. While the operation of the organization is beyond the control of the user, it is important that the user is notified about the status of the task.

One important goal of managing personal processes is to remind or to provide suggestions to a mobile user, rather than to enforce task executions as does a commercial WFMS. Therefore, traditional workflow scheduling issues, which address how to determine the mapping between tasks and available resources, do not seem to exist in this context. Instead, query capabilities of a Personal Workflow Management System (PWFMS) that keep a mobile user updated about the current personal process status are essential.

The coordination between tasks of a personal process has to be flexible. It is quite often that a mobile user executes a task that produces unexpected results or even engages in a totally unexpected task. In this case, rather than rejecting this change, the personal workflow system examines the impact of this change and notifies the user.

In response to the above requirements, Chen proposed a personal process model and a query facility that address requirement 2 and 4 [Chen01]. She also proposed and implemented an architecture that allows users to define their personal processes and provided a set of operations for manipulating them. Moreover, In [Lin02], Lin addressed requirement 1 by extending the architecture to include a personal process template that provides users with personal process templates. She observed that personal process templates for customers with different background or needs may differ, and a decision tree was used to efficiently organize the set of personal process templates of a given personal process. A customer is given a personal process template by the provider after providing his background information and needs. In
this thesis, we mainly address requirement 3 and 5. Specifically, we define the correctness of a personal process and develop methods for verifying the correctness of a personal process instance in response to personal process design, completion of task executions, and change of personal process. We also propose to use Web services for communication between a user and an enacting organization toward the execution of task. In addition, we extend the personal process model proposed in [Chen01] by incorporating the concept of task threads, each of which represents a distinct task execution outcome.

1.3 Objective

Our objective in this thesis is to revise and extend the personal process model and architecture proposed in [Chen01] and [Lin02]. In summary, we will do the following tasks:

1. The previous architecture had provided modeling methods of personal process using metagraph. A personal process is either explicitly specified by a user or downloaded from a single organization. Here, we consider the case where a personal process involves tasks executed by multiple organizations. In this extended model, several new concepts, including task’s threads, and sets of target data are introduced.

2. Incorporate Web services into the system architecture. Web services have been recognized as an ideal communication vehicle for B2B e-commerce applications. More and more services are provided by organizations as Web services. We are using Web services as the vehicles for interactions between the PWFMS and the services provider and the template provider.

3. Propose new operations and frequently used queries. In [Chen01], a SQL-like query was proposed. However, the specification of a SQL-like query is difficult
and the execution in a hand-held device is inefficient. We propose some new operations and frequently used queries to facilitate users in managing personal processes more easily.

Evaluate constraints of personal process model: As a formal model, the correctness of the personal process model has to be defined. We develop a set of constraints for the personal process model, and a personal model is correct only if all the constraints are satisfied.

1.4 Thesis organization

In this thesis, we focus on the personal process model and system design/implementation. This thesis is structured as follows: In Chapter 2, we introduce some highly related work, which include Web services, and workflow related works. In Chapter 3, we describe the extended personal process model and define the correctness criterion. In Chapter 4, we develop some operations and frequently used queries for the personal process model. In Chapter 5, we propose an overall architecture. We have implemented three components discussed in this thesis, and the implementation status is reported in Chapter 6. The entire thesis is concluded in Chapter 7.
Chapter 2 Literature review

In our proposed architecture, we use Web services as a communication vehicle between various components of the architecture. Thus, we first introduce some fundamental concepts about Web services. Then, we review previous work in modeling and managing personal processes. Since a personal process in our model may come from several organizations, our work falls in the category of inter-organizational workflow management. We also review researches in the context of inter-organizational workflow management issues.

2.1 Web services

The Internet has allowed millions of users to share information. The promise of Web services is the widespread integration of applications on the Internet. Many organizations have been adopting Web services for building their systems. To locate a Web service, a developer queries a service repository, which lists a number of businesses that offer the type of service that the developer wants. From the service repository, the developer can obtain the WSDL (Web Service Description Language) [Chri02] that describes the interface of the desired Web service. Following the WSDL, the developer then constructs a known client interface that can communicate with the Web service.

The framework of Web services allows users to encapsulate and publish services, thus enabling seamless integration of services from different service providers. In spirit, a Web service is like a remote procedure call that allows the interchange of a set of function arguments and an execution result. From a developer perspective, we can
differentiate between two different kinds of Web services – the traditional data oriented Web service and presentation oriented interactive Web service [Scha02].

Data-oriented Web services are general Web services that receive requests and return data, both encoded in XML documents. The signatures of their operations as well as the structure and semantics of the returned data are typically service type-specific. It is the responsibility of the service consumer to process the received data in a service-specific manner and generate any required presentation.

Interactive Web services (IWSs) include presentation and optionally interaction as a part of the service itself. They receive raw data to be processed and produce markup fragments that can easily be aggregated by consumers. This topic has been attended by OASIS. The Web Services for Remote Portals (WSRP) standard [Scha02] aims to simplify integration of remote applications at the presentation level such that in many cases no programming effort is needed for integrating IWSs. WSRP standardizes Web services at the presentation layer on the top of the existing Web services.

Web services are based a common architecture that includes communication protocols, interaction models, unified representation methods for services, and service discovery. HTTP is used as the network communication protocol, XML for data representation, Web Services Description Language (WSDL) for service description of unified representation method, and Universal Discovery, Description and Integration (UDDI) for service registry/discovery. The interaction model generally follows Simple Object Access Protocol (SOAP). In recent years, many new issues, such as security, workflow integration and presentation layer Web services, emerge and demand new standards. In our architecture, we use only the basic functions of Web services.
2.2 Personal workflows

In [Chen01], Chen proposed a personal process model that composes a set of tasks and data as a personal process. Each task contains a set of input data, a set of output data, executable times, and available locations. In addition, the model is equipped with a set of operations for inquiring a personal process. The definition of operations is listed in the following: (\(T\) means a set of tasks and \(D\) a set of data)

**PLACE_OVERLAP**: Produces a subset of \(S_1\) whose executing place overlaps with the executing place of some task in \(S_2\).

**TIME_OVERLAP**: Produces a subset of \(S_1\) whose executing time overlaps with the executing time intervals of some task in \(S_2\).

**MAKE_EXECUTABLE**: Returns a subset of \(T\), each of which has the input data as a subset of \(D\).

**NEED_TASK**: Produces a set of tasks that can be collectively executed by taking \(D_1\) as the input and producing \(D_2\) and has the lowest cost.

**COMBINED_INPUT**: Returns a set of data, each of which is an element of input data of some element in \(T\) but not in the output data of any element in \(T\).

**COMBINED_OUTPUT**: Returns a set of data, each of which is an element of output data of some task in \(T\).

Along with the above operations, they also defined set operations, such as *intersection* (\(\cap\)), *union* (\(\cup\)), and *difference* (\(\sim\)), and a relational operator *selection* (\(\sigma\)) for users used in querying task. In addition, a declarative query language similar to SQL-statements was also proposed.

In [Lin02], Lin extended the above-mentioned system architecture by adding a personal process template provider, which allows the designer to define various templates for a personal process and enable users to choose a template that meets their
interests and background. This module aims to release the user from the tedious work of defining a personal process.

ServiceFlow introduced a general concept for supporting interrelated, personalized and situated services carried out across different organizational units or provider firms. It exploits both, workflow management and Internet technology, while adding further customer-related aspects [WK02]. In [KW01a], Klischewski and Wetzel developed a modeling methodology for service processes. Within Serviceflow modeling a series of service points (denoted as a list) serves as the process plan or schedule (looking ahead) as well as the process history (looking back). Each of those points includes a UML specification of actors carrying out certain tasks/activities as well as the pre- and post-conditions at each point for “contracting” within process execution (see Figure 2-1).

ServiceFlow was experimented on health provider networks of an inpatient surgical operation [KW01b] and the citizen postal vote through the Web portal of the city state of Hamburg (Germany) [KW02]. In the former, a patient usually moves back and forth between different physicians/specialists and a clinic to receive a profound diagnosis as well as appropriate medical and care treatment. In this process, as depicted in Figure 2-2, a patient typically starts with consulting a family doctor, followed by a series of activities, including being directed to a specialist, choosing a hospital, receiving consultation and registering at the hospital with a plan for further preparation, passing through all preparations, staying in the hospital where the operation is performed, and finally a aftercare treatment at specialists.

Although the ServiceFlow provides processes for personal use, these processes are actually an inter-organizational business process which, once defined, will proceed
automatically without the user’s intervention and cannot be modified by the user. In contrast, our personal process model, though also involves activities from various organizations, does not mandate the automatic interaction between organizations and thus allows the dynamic modification to a personal process.

![Figure 2-1: Service point](image1)

![Figure 2-2: Model of the case serviceFlow](image2)

### 2.3 Workflow collaboration

Competitive pressures are forcing organizations to increasingly integrate and automate their business operations such as order processing, procurement, claims processing, administrative procedures and the like. Such business processes are typically of long duration, involve coordination across many manual and automated tasks, require access to several different databases and the invocation of several application systems, and are governed by complex business rules. In [DHL01], Dayal et al. proposed a framework for classifying systems supporting business process management (Figure 2-3). It distinguishes business processes along two dimensions: task automation and process structure. Tasks in a process may be application centric or human centric. The process can be highly-structured, semi-structured, and
unstructured. The design center for process management systems lies in the upper right-hand shaded corner, i.e., they are intended for managing processes composed of more application centric tasks and are more structured, although the business processes they support often contain both manual and automated tasks, and they often accommodate certain degree of ad hoc scripting. In contrast, the lower left corner has been better supported by capabilities such as on-line shared space systems (e.g., Lotus Notes), or online meeting systems. One of the challenges in process management has been to provide end-to-end support for business processes that span both design centers: for example, a procurement process spans strategic sourcing, which tends to be less structured and involves human manipulation of documents, and order processing and payment processing, which tend to be more structured and involve mostly automated steps. Within such a framework, personal processes sit on the center of the horizontal axis (process structures) because personal processes are semi-structured in the sense that for achieving a common goal, different personal processes may undergo some differences in their task sets and relations, while sharing a majority of common task set. Regarding the vertical axis (task automation), most tasks in personal processes could be manual today as they do not provide programming interfaces for instantiation but a lot of them could become automatic in the near future, depending on how fast Web services are adopted by enacting organizations. Thus, the personal process is different from commercial business process management systems and on-line shared space systems.
In [CH01], a collaborative process framework was proposed to extend the centralized process management technology (Figure 2-4). A collaborative process involves multiple parties, each playing a role in the process. Two aspects distinguish the collaborative process model from the conventional centralized process model:

1. **The process definition** is based on a commonly agreed business interaction protocol, such as the protocol for on-line purchase or auction.

2. **The process execution** is not performed by a centralized workflow engine, but by multiple engines collaboratively.

**The Common Definition of Collaborative Process** is the starting point of the collaborative process framework. There are at least 3 parts for effective use of common business process definitions:

1. There must exist a common business process meta-model and its associated schema language, so that business processes can be codified in a standard way;

2. There must also exist a mechanism for common process descriptions, including business documents associated with these processes, to be easily re-used as building blocks for more complex processes, or customized for
the special needs of vertical industries or geographical segments;

Finally, there must exist a mechanism for enterprises to publish their ability to participate in specific roles of common business processes as process flow enabled Web services, so that potential partners can automatically discover each other and engage in the process execution.

In the collaborative process framework, the common business process definitions drive the definition and development of a process-compliant service at an enterprise. This is illustrated in the two Role Spec boxes in Figure 2-4. An enterprise determines the role(s) it wishes to play in a process, and develops the role process specifications and the corresponding internal execution control, including invocation or dispatching of local services (e.g. wrapped legacy applications). Once the process-compliant role specification is developed, it can be published as a Web service. Note that there is no requirement that local services be published as Web services, i.e., such services may not be directly accessible from trading partners. Local services are accessed indirectly through the local collaborative process manager.

A scenario of **The Process Execution** is shown in Figure 2-4. Each execution of a collaborative process, or a logical process instance, consists of a set of peer process instances run by the Collaborative Process Managers (CPMs) of participating parties. These peer instances conform in behavior to the specification of the role set forth in the common process definition, but may have private process data and sub-processes. The CPM of each party is used to schedule, dispatch and control the tasks that that party is responsible for, and the CPMs interoperate through an inter-CPM protocol to exchange the process data (or documents) and to inform each other on the progress of the process execution.

The framework requires that local CPMs interoperate with one another; however, there is no requirement that each local CPM be identical. To the extent that local
CPMs are capable of enforcing the common business process specifications, they can differ significantly in functionality, such as support for internal data flow, local service integration, and nested sub flows. Therefore it is possible that, for example, one CPM is based on Microsoft’s BizTalk Server, another is based on WSFL (Web Service Flow Language) server or IBM’s MQSeries.

![Diagram of collaborative process management](source: [CH01])

Figure 2-4: A scenario collaborative process management (source: [CH01])

To truly integrate business processes across enterprise boundaries it is not sufficient to merely support simple interaction using standard messages and protocols. Business interactions require long-running interactions that are driven by an explicit process model. In the past, there are some business process integration standards, such as RosettaNet[RosettaNet] and ebXML[ebXML]. The RosettaNet focused on defining standard interfaces between partners in IT industry. The ebXML attempts to leverage the pre-existing industry experience in Electronic Data Interchange (EDI) in designing new XML–based B2B framework. In recent years, to solve communication problems, several Web services composition languages such as BPEL4WS[Curb03], WSFL[Leym01], XLANG[That01], WSCI[Arki02b], and BPML[Arki02a] have been alerted. These languages are also known as Web services flow languages, Web
services execution languages, Web services orchestration languages, and web-enabled workflow languages [Aals03]. The WSFL (Web Services Flow Language) has formulated by IBM, which specifies two types of Web services composition 1) an executable business process known as a flowModel and 2) a business collaboration known as a globalModel. The XLANG was formulated by Microsoft, which is a business modeling language for BizTalk. The BPEL4WS (Business Process Execution Language for Web Services) is the cooperative merging of WSFL and XLANG. It extends the Web Services interaction model and enables itself to support business transactions. The WSCI has formulated by Sun/BEA/Intalio/SAP, which is an XML-based interface description language that describes the flow of messages exchanged by a Web Service participating in choreographed interactions with other services. In addition, IBM has also supplied available commercial product in this area, which can be seen in [LRS02]. Unfortunately, these so-called standards are proposed and supported by industry vendors, and as history shows, real standards do not originate from vendors pushing their own products. In [Aals03], Aalst has pointed out that these languages typically have no clearly defined semantic.

The Wf-XML1.1[Wfmc01] had released by WfMC, a language designed to standardize data transfer requirements between interoperable systems. It provides a XML message-based architecture for communicating between workflow sites. Based on this concept, it defines interaction model, message structure (DTD), operations, and status types.

**Interoperability**

Wf-XML can represent the data required to support chained, nested and parallel-synchronized workflows. When processes are chained, a process instance being enacted by one workflow site triggers the creation and enactment of a
sub-process on a second site. Once the sub-process initiates, the first site maintains no interest in the sub-process. When processes are nested, the process instance enacted on the first site causes the creation and enactment of a sub-process instance on a second site, and then waits for the sub-process to terminate before proceeding. When processes are parallel-synchronized, the process instance enacted on the first site cause the creation and enactment of one or more sub-processes instances on the other sites.

To enable interoperability, workflow sites need to expose an API sufficient to parse a Wf-XML message and act on its contents. WfMC expects that HTTP and SMTP will become the most widely used data transport mechanism. Wf-XML can be used as synchronous or asynchronous interaction mechanism between "generic services" that may consist of a number of different resources. The synchronous interaction mechanism likes traditional RPC calls. The asynchronous interaction mechanism likes the synchronous mechanism, but only an acknowledgement message is returned to the requester or the engine.

The batch mechanism comprised a set of synchronous or asynchronous messages, which reduces messages transfer. In addition, the batch message can hold request and response messages at the same time.

**Message structure and operations**

The Wf-XML specification defines individual and batch functions that can be exposed to other processes or services as an "operation". There are four special groups of operations identified in version 1.1 of the specification: Control, ProcessDefinition, ProcessInstance, and Observer. A resource must support operations defined for a given group, and may support more than one group. These operations are described bellow:
Control operations describe administrative interactions among interoperating services. These interactions are typically unrelated to specific processes, which interactions are managing the status of batch messages.

ProcessDefinition operations describe the service's basic functions, and provide a model for instances of the service. Defining a ProcessDefinition group for a generic service is the basic way to expose that service to other processes in the workflow application.

ProcessInstance operations represent the instantiation of a Process Definition group for an actual running process. Its operations can be used to acquire information about the instance and to control the instanced process.

Observer operations enable a process instance to advise other services (ex: the Observer of a process instance) of its events or status, such as its completion or termination.

**Status types**

The other important aspect of the Wf-XML is the status of the tasks in each process. The higher-level states definition has been defined, and an implementation can choose a port of status to implement. These statuses have different mean in individual or batch functions. In the following, we will use individual point of view to explain each status:

- **open.notrunning**: a task in this status means that some required data have not been acquired.
- **open.notrunning.suspended**: a task in this status means that required data have been acquired, but for some reasons made it is suspended (ex: the task execution unit too busy).
- **open.running**: a task in this status means that the task is running.
- **closed.completed**: a task in this status means that it was completed by
execution unit.

- closed.abnormalCompleted: a task in this status means that some problems were not able to solve in execution unit, but still return this status to the observer (requester).

- close.abnormalCompleted.terminated: a task is in this status when it has been terminated by the observer before it completed its work process. (optional)

- closed.abnormalCompleted.aborted: a task is in this state when then execution of its process has been abnormally ended before it completed its work process. (optional)

Note that the above-mentioned standards are used for coordinating processes executed within different organizations. In our work, the only type of interactions is between the PWFMS and organizations. Therefore, none of these standards will be adopted. Rather, we will define the Web services for the interactions between a PWFMS and an organization.
Chapter 3  Personal process model

In this section, we describe our personal process model, which is based on [Chen01]. This model allows users to specify and to query personal processes.

3.1 Personal processes model

In this section, we define the syntax of a personal process. A personal process is comprised of the following components:

- a set $T$ of tasks,
- a set $D$ of data,
- a set $R$ of threads,
- several functions that map a task to its name ($\Phi_n$), the input data set ($\Phi_i$), the executable time intervals and places ($\Phi_{tp}$), the set of involving threads ($\Phi_r$), the nested type ($\Phi_{nest}$), the automatic type ($\Phi_{type}$), and the task’s service provider ($\Phi_{url}$), as defined below:

$n\Phi_n: T \rightarrow \text{String}$, which indicates the name of a task.

$n\Phi_i: T \rightarrow 2^D$, which indicates the input data set of a task.

$n\Phi_r: T \rightarrow 2^R$, which indicates the set of threads pertaining to a task.

$n\Phi_{tp}: T \rightarrow 2^\text{Interval}\times\text{Region}$, which indicates the set of executable intervals and regions of a task.

$n\Phi_{nest}: T \rightarrow \text{Boolean}$, which indicates that the task is atomic or nested (i.e., a subprocess).

$n\Phi_{type}: T \rightarrow \text{Boolean}$, which indicates that the task is automatic or manual. A task is said to be automatic if the enacting organization provides a Web

\footnote{For formal definitions of Interval and Region, please refer to [Guet00]. However, in our current implementation, we represent an interval as a set of timestamps and a region as a symbol, as explained later.}
service for notifying the execution status of the task. A task is manual if it
is not automatic.

Φ_{url}: T→String, which indicates the URL of the Web service provided by
the organization. This is valid only for tasks of automatic type.

1 a function Φ_{o}: R→2^D that maps a thread to its output data set,
1 a function Δ_{s}: D→String that maps a data item to its associated name.

In addition, there are attributes that record the execution status of processes,
tasks, threads and data. We call these attributes control attributes. In this thesis, we
consider five control attributes, Φ_{ps}, Φ_{s}, Φ_{rs}, Φ_{c}, and Δ_{s} that are associated with
processes, tasks, threads and data respectively. Φ_{ps}: P → (INITIAL, EXECUTING,
success, or failed. Φ_{s}: T → (INITIAL, PREPARED, EXECUTING,
logical-completed, failed, or physical completed. Φ_{rs}: R→(UNDECIDED, FAILED, SUCCEEDED) describes the
status of a thread. For a given task, at most one of its threads could become
SUCCEEDED. Φ_{c} T→R maps a task to a completed thread. Δ_{s}: D→
(UNAVAILABLE, AVAILABLE) describes the availability of a data item. The status
about tasks and threads will be explained in more detail in the next subsection.

The seven functions Φ_{t}, Φ_{r}, Φ_{c}, Φ_{tp}, Φ_{nest}, Φ_{type}, Φ_{url} are attributes pertaining to
tasks, and Φ_{o} is an attribute pertaining to threads. A task may constitute several
threads (represented by Φ_{t}), each of which represents a different execution outcome
(represented by Φ_{o}). As a result, each thread has its output data set, but among the
threads of a task, only one (represented by Φ_{c}) can eventually succeed. Φ_{tp} are the
attribute that specifies when and where a task can be performed. An interval is
modeled as a 5-tuple <start, cycle, from, to, end>, where start and end indicate the
effective time and the expired time respectively, and *cycle*, *from*, and *to* dictate the effective period of a repetitive cycle. The time format of *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, to specify the interval of a task that can only be executed between 9am to 5pm everyday effective from 2003-7-1 to 2003-8-31, we can use the following format: <2003-7-1, ‘day’, 09:00:00, 17:00:00, 2003-8-31>.

We distinguish data items into two kinds: *primitive* and *processed*. A primitive data is not produced by any task modeled in the system, and a processed data must be generated by at least (a thread of) one task. A primitive data could be a data file, a blank form, a personal belonging (e.g., the ID card, credit card), or anything that can be prepared by the user. A processed data is available only when at least one task that is capable of producing it is completed. For example, the task *Reimburse* takes five data items as input: ‘Diagnosis Certificate’, ‘Insurance Certificate’, ‘Chronic Disease Proof’, ‘ID Card’, and ‘Application Form’. The ‘Diagnosis Certificate’ and ‘Insurance Certificate’ are processed data items, which can only be generated by *Stamp* and *RequestProof*, separately. The others are primitive data items.

We also define a source data set *P_s* and a target data set *P_t* for correctness verification as shown in the following. The source data set is the set of data items that are currently available, formally *P_s*={*d*∈*D*: Δ (*d*)=AVAILABLE} and a target data set *P_t*⊆*D* is a set of data items such that the availability of each data item in *P_t* marks the successful termination of the personal process. We will make use of the two data sets in deciding whether a personal process is correct, as explained in the following section.

Note that in the proposed model, there is no rigid order on task executions. Tasks
are associated by their respective attribute values, which may implicitly decide their execution orders. For example, if a task $T_2$ needs a data item that can only be produced by $T_1$, $T_2$ will not execute before $T_1$ terminates.

### 3.1.1 Task status

In the above section, we have defined the states about tasks and threads. The transitions between the states of a task can be expressed as a State Transition Diagram, which is depicted in Figure 3-1.

![Status transition diagram of a task](image)

Figure 3-1: Status transition diagram of a task

The “Prepare” event represents the fact that the input data set, time, and places for executing the task are all ready. The “Start” event, signified by an enacting organization, indicates that the corresponding business process in the organization is enacted. The “Complete” event, also signifies by the enacting organization, specifies that this task has been successfully completed (i.e., one of its pertaining threads becomes SUCCEEDED). However, some physical objects may not have reached the user. Once all output data items have arrived, which marks the event “Done,” the task is said to be in *Physically Completed* state. However, if, for some reason, the task is voluntarily or involuntarily terminated by the organization, an event “Not-Complete” will be issued, and the task is said to be in *Failed* state. At this time, if the user decides to re-instantiate the task, its state will return to *Initial*. Therefore, when we
say that a task $t$ is unexecuted, we actually mean that $t$ is in either Initial state or Failed state.

3.1.2 Task output thread

Each task can be expressed as a group of threads as shown in Figure 3-2. Initially, all threads are of the state UNDECIDED. When the task reaches the “Logically Completed” state, exactly one thread enters the state SUCCEEDED, while the other threads are of the state FAILED.

![Figure 3-2: Task threads](image)

3.1.3 Meta schema of personal processes

Based on the above-mentioned definitions, we use a class diagram to represent its meta-schema. As illustrated in Figure 3-3 a personal process is comprised by at least one task. Each task consists of a number of threads, a set of input data items, and a number of interval region pairs. A thread represents a possible execution outcome which is modeled as a set of output data items. The Interval-Region set contains task’s executable time intervals and locations. The attributes of each class are listed in Table 1:
Figure 3-3: Meta schema of the personal process

Table 1: Attributes of classes in the meta schema

<table>
<thead>
<tr>
<th>Class</th>
<th>Attribute</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal Process</td>
<td>Name</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Status</td>
<td>INITIAL, EXECUTING, SUCCESS, or FAILED</td>
</tr>
<tr>
<td></td>
<td>Provider</td>
<td>String (the name of template provider)</td>
</tr>
<tr>
<td>Task</td>
<td>Name</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Priority</td>
<td>(low)1, 2, 3, 4, or 5(high)</td>
</tr>
<tr>
<td></td>
<td>Status</td>
<td>INITIAL, PREPARED, EXECUTING, LOGICAL-COMPLETED, FAILED, or PHYSICAL-COMPLETED</td>
</tr>
<tr>
<td></td>
<td>Nest</td>
<td>Atomic or Nested</td>
</tr>
<tr>
<td></td>
<td>Type</td>
<td>Automatic or Manual</td>
</tr>
<tr>
<td></td>
<td>URL</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>String</td>
</tr>
<tr>
<td>Thread</td>
<td>Name</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Status</td>
<td>Success or Failed</td>
</tr>
</tbody>
</table>
### 3.2 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. Unless otherwise noticed, here we only refer to *unexecuted* tasks.

**Definition 1:** A task $t_1$ is said to temporally precede another task $t_2$, denoted $t_1 \prec t_2$ if $t_1$ can only be executed before $t_2$. Formally, $t_1 \prec t_2$ if

$$\max_{\text{time}}(\Phi_y(t_1)) \leq \min_{\text{time}}(\Phi_y(t_2)),$$

where $\max_{\text{time}}$ (min_time) returns the maximum (minimum) time of the intervals 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 one of $t_2$'s input data item can only be generated by $t_1$. Formally, $t_1 \prec_d t_2$ if

$$\{d : d \in \Phi_i(t_2), (\exists r \in \Phi_r(t_1), d \in \Phi_o(r)), (\forall t \in T, t \neq t_1, \forall r \in \Phi_r(t), d \notin \Phi_o(r))\} \neq \emptyset.$$
**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 $t_1 \prec t_2$ or $t_1 \prec_d t_2$.

**Acyclic Dependency Constraints**

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 tasks involved in the cycle of $(T, \prec_s)$ cannot possibly be executed and thus are redundant.

**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 $T_j$ such that

$$
\Phi_j(t_j) \subseteq \{d : d \in P \cup \Phi_o(r) : r \in t, t \prec t_j\}.
$$

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 the strict dependency $\prec_s$ as the strict dependency must be followed strictly. The second condition states that the output of all tasks that precedes a task $t_j$ as well as the currently available data items will make $t_j$ executable in terms of input data.

**Definition 5:** Given a set of tasks $D_1$ and another set of tasks $D_2$, 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 $T_i$. 
**Task aliveness constraints**

A personal process is said to preserve task aliveness if for each task \( t \), \( \Phi_i(t) \) is inducible from \( P_s \) and, for each data item \( d_i \) in \( P_t \), there exists a path from some data item produced by some thread in \( t \) to \( d_i \), where \( P_s \) is the source data set and \( P_t \) is the target data set.

A personal process that preserves the task aliveness constraint ensures that each task has the chance to be executed and that the execution of the task indeed contributes to the availability of at least one data item in the target data set, thereby leaving no redundant tasks in the specification of the personal process. As shown by the following lemma, the task aliveness constraint is indeed stronger than the acyclic dependency constraint.

**Lemma 1:** For each task \( t \) that is involved in a cycle formed by the task dependency \( \prec \), \( \Phi_i(t) \) is not inducible from \( P_s \).

From Lemma 1, we know that it suffices to check only task aliveness because task aliveness implies acyclic dependency.

**Process aliveness constraints**

A personal process is said to be alive (or preserve the process aliveness constraint) if \( P_t \) is inducible from \( P_s \), where \( P_s \) is the source data set and \( P_t \) is the target data set.

To check whether a personal process is alive, the user needs to specify a target data set, representing an acceptable outcome for the completion of the personal process. In other words, if all data items in the target data set become available, this personal process is pronounced as completed.

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 do not need to be executed.
Correct personal processes

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

If a personal process is incorrect, it represents the following possibilities:

1. Some tasks become obsolete and can 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 task.

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

Please note that a personal process may be correct at the design time but becomes incorrect when some task completes (as the result of successfully executing a task thread) or when the personal process undergoes some dynamic change. Therefore, correctness check of a personal process \( p \) should be conducted at the following time points:

- when \( p \) is initially designed,
- when a task in \( p \) is completed, and
- when the definition of \( p \) is dynamically changed (e.g., adding, deleting, or modifying a task)

From the above definition, one can imagine that checking the correctness of a personal process requires substantial computing power. Thus, a PWFMS that executes on a hand-held device cannot afford 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 the scratch.

**Lemma 2:** Assume a personal process \( p \) is correct before a task \( t \) is completed.

If \( t \) contains only a single thread, then \( p \) is still correct after \( t \) is completed.

**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.,
\[
U = \{ d : d \in D, \Delta_x(d) = UNAVAILABLE, (\exists r \in \Phi_r(t), \Phi_a(r) = FAILED, \Phi_o(r) = d) \}
\]
) 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 task in \( T \) (i.e., \( \forall d \in U, (\forall t \in T, d \notin \Phi_r(t)) \)), then \( p \) is still correct.

Therefore, each time when a task is completed, the PWFMS only needs to perform a simple check on the conditions stated in Lemma 2 and 3. If either condition is satisfied, we can ensure that the personal process is still correct. However, if neither condition holds, the PWFMS can either prompt the user for a human check or have a server on the wired network do the validation (when the wireless communication is available).
Chapter 4  Operation definition

In [Chen01], the author has proposed a set of operations for inquiring a personal process and each query are expressed as a SQL-Like query. However, it is not clear how powerful the proposed query syntax is. In this thesis, we adopt the first order predicate calculus [BM77] as the formal query language for querying personal processes. For example, to retrieve the names of the tasks in the personal process “reimbursement” process (shown in Chapter 1) with a given data item ‘Insurance Certificate’ as part of its input; we can use the following query:

Query scenario 1:
\[\{t.name | \text{TASK}(t) \land t.process\_name = \text{'reimbursement'} \land (\exists d \in t.input, d.name = \text{'Insurance Certificate'})\}\];

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**\((t_1, t_2)\)
  - It returns TRUE if the task \(t_1\) and \(t_2\) can be co-executed at a certain time and a certain place.

- **POSSIBLE_OUTPUT**\((t, d)\)
  - It returns TRUE if one of the thread in \(t\) produces the data item \(d\).

- **TASKSET_INPUT**\((d, t_1, t_2, \ldots, t_k)\)
  - It returns TRUE if \(d\) appears in the input of at least one task in \((t_1, t_2, \ldots, t_k)\)
    but not in the output of any thread in \((t_1, t_2, \ldots, t_k)\).
4.2 Frequently used queries

In this section, we propose a list of frequently used queries and the corresponding expressions using first order predicate calculus:

1 Query scenario 2: Find the names of tasks that can be executed after both ‘Request proof’ and ‘Stamp’ are completed in the “reimbursement” process.
   \[ \{ t.name | \text{TASK}(t) \text{ and } t.process.name = \text{‘reimbursement’ and } (\forall d \in t.input, (d.status = \text{‘available’ or } (\forall t_1, \text{TASK}(t_1) \text{ and } t_1.name = \text{‘Request proof’ and POSSIBLE_OUTPUT}(t_1, d)) \text{ or } (\forall t_2, \text{TASK}(t_2) \text{ and } t_2.name = \text{‘Stamp’ and POSSIBLE_OUTPUT}(t_2, d)) \} \]

1 Query scenario 3: Find a set of tasks whose input data are available and can be co-executed with ‘Diagnose’ in terms of time and place in the reimbursement process.
   - Calculus: \[ \{ t.name | \text{TASK}(t) \text{ and } (\exists t_1, \text{TASK}(t_1) \text{ and } t_1.name = \text{‘Diagnose’ and } t_1.process.name = \text{‘reimbursement’ and OVERLAP_TIME_PLACE}(t, t_1) \} \]

1 Query scenario 4: Find the set of tasks that can be executed immediately (Note that CURRENT is a system-defined dummy task that has the current time and the current place as the attribute values).
   - Calculus: \[ \{ t.name | \text{TASK}(t) \text{ and } (\forall d \in t.input, (d.status = \text{‘available’}) \text{ and } (\exists t_1, \text{CURRENT}(t_1) \text{ and OVERLAP_TIME_PLACE}(t, t_1)) \} \]

1 Query scenario 5: Find the set of data that is needed to complete tasks ‘Request proof’, ‘Diagnose’, and ‘Stamp’.
   - Calculus: \[ \{ d.name | \text{DATA}(d) \text{ and } (\exists t_1, \text{TASK}(t_1) \text{ and } t_1.name = \text{‘Request proof’ and } ((\exists t_2, \text{TASK}(t_1) \text{ and } t_1.name = \text{‘Diagnose’ and } (\exists t_3, \text{TASK}(t_1) \text{ and } t_1.name = \text{‘Stamp’ and TASKSET_INPUT}(d, t_1, t_2, t_3)))) \} \]
Note that some queries such as the following will not be solvable by first order predicate calculus.

*Find the set of tasks that need to be done in order to produce ‘receipt’*

As can be seen, the first order predicate calculus is capable of querying a personal process definition/execution quite well. Therefore, any query language that fully implements first order predicate calculus on objects with composite, multi-value attributes and allows user-defined functions meet our requirements perfectly. On such a language is SQL3 [EM99]. However, due to the limited computing power and memory inherent to hand-held devices, it is not possible and in practice is overkill to implement a full-fledged query language. Instead, we enumerate a number of frequently used queries and implement them as parameterized queries. To execute these queries, a user only needs to click some items or fill in forms that prompt the required parameters.
A PWFMS is designed to facilitate users in coordinating and tracking personal tasks with the assistance from various organizations. The overall system has to support the design of a personal process, the execution of a personal task (which interacts with a service provider), and queries to the personal process. To do so, the architecture involves three types of components: the template providers, the service providers, and the PWFMSs. The functions of each component are described below:

− **The Template Provider** provides a set of personal process templates; each suited to a user with distinct background and needs for achieving a personal goal. After selecting a particular template, a user can use his handhold device with a PWFMS installed to download the template.

− **The Service Provider** tracks the execution status of a task (possibly through an enterprise workflow management system) and publishes a set of Web services that reveal tasks’ execution status and available (partial) personal processes.

− **The PWFMS** is executed on handhold devices to manage personal processes. It is capable of downloading personal processes from template providers and interacts with service providers for keeping track the execution status of tasks. In addition, it also provides interfaces for a user to change task execution status and place queries.

The relationship between the three types of components is depicted in Figure 5-1. As will be described later, the interactions between different types of components are conducted via Web services. Note that we do not consider the user authentication and security issues in this thesis. While they are important components in a real system, they are ignored in this architecture for simplicity.
5.1 Components and interfaces

In this section, we describe the detailed functions of each component, as shown in Figure 5-2. Each component also provides a number of Web services for the other components, as depicted in Figure 5-3.
5.1.1 The service provider

To facilitate the deployment and execution of tasks in personal processes, an organization can install a service provider. The service provider has two objectives: one is to provide a collection of (partial) personal processes, each consisting of a set of tasks for realizing a service, and the other is to monitor the execution status of tasks. To fulfill these two objectives, the service provider has three interface modules: the
**Process Designer**, the **Maintenance** and the **Web Services**, as circled in dotted lines in Figure 12. The process designer module provides a user interface that assists the designer managing existing personal processes or creating new personal processes. The Maintenance module provides a user interface that assist service executor managing executing processes. The Web Service module handles a set of published Web services, including listProcess, getProcess, and getStatus (shown below). An invocation of a Web service is indeed an interchange of two XML documents (one for service request and the other for service response). In the following, we describe the kinds of data needed for executing these Web services.

- **listProcess**: return the names of all personal processes stored in the service provider

  Request: null.

  ```xml
  <request>
  <process>ALL</process>
  </request>
  ```

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

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

- **getProcess**: return the definition of a given personal process.

  Request: a personal process name

  ```xml
  <request>
  <process>processName1</process>
  </request>
  ```

  Response: definition of the process
− getStatus: return the execution status of a task.

yy Request: “process name” and “a set of task names”

<request>
  <process name="processName1"></process>
  <task>name1</task>
  <task>name2</task>
</request>

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

<response>
  <task name="name1" status="Logical-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 WFMS for fulfilling its services, each task in a personal process is treated as a workflow, and the Process Manager has to interact with the WFMS for the update of workflow 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 Chapter 3.
5.1.2 The template provider

The template provider allows designers to build personal process templates that comprise tasks published by various service providers (termed nested tasks) as well as atomic tasks. It also allows users to customize a given template and to download the customized template to its PWFMS. Moreover, it provides three interface modules (circled in dotted lines). The first is Template Designer module that assists designers to manage existing templates or creating new templates. Typically, a designer defines a personal process and creates several templates for it, each suited to a customer with a distinct background or need. Another is Personal Template Builder module that allows a user to select a template that meets his personal needs. Please refer to [HY03] for how a user interact with a prototyping system for selecting a template. Note that the selected personal process templates will be stored in the Repository for later access (e.g., downloading). The other is Web Service module that handles a set of published Web services, namely listTemplate and getTemplate. Their functions and invocation formats are described below.

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

  ü  Request: requires “user identifier”

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

  ü  Response: a set of names of customized personal processes

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

- getTemplate:

  ü  Request: requires “user identifier” and “personal template name”
Response: a personal process (customized template)

```xml
<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="2003-5-5" end="2003-5-10"></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 that described in Service Provider.

5.1.3 Personal workflow management system

The PWFMS enables users to create or fetch a new personal process (from the template provider), to manage existing personal processes, to interact with the service providers of organizations that are responsible for executing tasks. The WS 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 getTaskStatus Web service).

The Personal Process Manager is the core module of the PWFMS that performs main functionalities, 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 via the Web service getTaskStatus), the correctness of
the personal process has to be checked by the Validation Agent. The Query Processor supports all frequently used queries and other basic queries described in Chapter 4.

As mentioned in Chapter 1, an important goal of managing personal processes is to remind or provide suggestions to a mobile user, rather than to enforce task executions as does a commercial WFMS. Thus, while the PWFMS may automatically update the status of an executing task via Web services, the invocation of a (physical) task has to be performed by the user (e.g., hand in the required papers for applying a certain thing). 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, and Failed). The completion of a task may in turn make another task prepared (to execute). The user of the PWFMS may use the query interface to decide which task to engage next. The same procedure is iteratively performed.

5.2 Modeling workflows of a personal process

We use the UML activity diagram to model the workflow behavior of a personal process (see Figure 5-4). When a personal process is started, it will set the process status as “Executing”, the status of primitive input data items as “Available”. Then the user may place a query to choose the set of prepared tasks, among which one task is chosen for execution. Some time after a task is executed, a task status update event may arrive. A task status update event may be either initiated manually by the user (for manual tasks) or automatically by an organization via Web services (for automatic tasks). 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 re-evaluated. 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 pronounced as “Failed.” Conversely, the user may perform an update on the personal process definition and have the PWFMS to re-check its correctness. If a task status is changed to “Failed,” the user will examine the task and decide whether or not to re-set its status to “Prepared” for possible re-execution. When the process is correct and all the target data are available, we say the personal process is succeeded.

Figure 5-4: Activity diagram of a personal process
Chapter 6 Implementation

To prove our concept of the proposed personal process model and the architecture, we have implemented the three types of components, namely the service provider, the template provider, and the PWFMS. To ease the job of implementation, we have simplified a bit the proposed model and accordingly the architecture. First, in our implementation, time and location are separate attributes, while in our model time and location are integrated as a composite, multi-valued attribute. Second, due to the lack of positioning devices and services available to our environment, we ignore the position related query statements (e.g., such queries as finding me all tasks that can be executed at my current position will simply return all tasks). The active alert function is not implemented due to the lack of position acquisition capabilities and our desire to implement a cross-platform PWFMS (which can be executed in different mobile platforms, including Palm, WinCE, and mobile phone).

Both the service provider and the template provider run on servers located on the fixed network. Its Web services are implemented by using Perl and execute with Apache Web server. The template provider is a Web-based system using Apache as the Web server and MySQL as the underlying DBMS. We use PHP for implementing the Web interface and the Web services. The PWFMS executes on hand-hold devices and is implemented by using J2ME-MIDP (Java 2 Mobile Information Device Profile) as the development tool.

6.1 The template provider system

When a user logs on the template provider, s/he is prompted with a set of available templates and a set of customized templates s/he has built before. Figure 6-1 shows a screenshot of the template provider 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 built one template (Skin). When the user chooses to customize a template, he will be prompted to choose the service provider for each partial personal process in the template. Figure 6-2 shows the screenshot when the user chooses to customize the template reimbursement, in which there are two tasks, namely ‘Insurance Certificate’ and ‘Diagnosis Certificate’, each can be provided by two service providers. Then the user may further customize the template by following the guide of the system (see Figure 6-3). For example, the user can add new tasks (see Figure 6-4) or modify existed tasks (see Figure 6-5). After the customization, the user can define the target data set and verify the correctness of the definition (see Figure 6-6). The system also provides a facility for the user to view the data dependency between tasks, i.e., the relation $t_1 \prec_d t_2$ described in Section 3.2 (see Figure 6-7). Finally, the user can export the template (so that it can be downloaded to the PWFMS) (see Figure 6-8). An exported personal process template can be retrieved by a PWFMS via Web service interface (explained later).
### Available Templates

<table>
<thead>
<tr>
<th>Process Name</th>
<th>Provider</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reimbursement</td>
<td>NHIC</td>
<td>It assists users to execute the refund process.</td>
</tr>
<tr>
<td>vehicleReg</td>
<td>registration agency</td>
<td></td>
</tr>
</tbody>
</table>

### Personalized Templates of skin

<table>
<thead>
<tr>
<th>Process Name</th>
<th>Provider</th>
<th>Date</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skin</td>
<td>Skin(self)</td>
<td>2002-05-26</td>
<td>Delete</td>
</tr>
</tbody>
</table>

**Figure 6-1:** Template provider-list processes

**Figure 6-2:** Template provider-choice providers
Figure 6-3: Template provider-manage a personal process
Figure 6-4: Template provider-add task
**Figure 6-5: Template provider-modify task**

<table>
<thead>
<tr>
<th>Input Data</th>
<th>Type</th>
<th>Status</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>processed</td>
<td>unavailable</td>
<td>Delete</td>
</tr>
<tr>
<td>Insurance Certificate</td>
<td>primitive</td>
<td>unavailable</td>
<td>Delete</td>
</tr>
<tr>
<td>Source Disease Certificates</td>
<td>primitive</td>
<td>unavailable</td>
<td>Delete</td>
</tr>
<tr>
<td>ID Card</td>
<td>primitive</td>
<td>available</td>
<td>Delete</td>
</tr>
<tr>
<td>Application Form</td>
<td>primitive</td>
<td>available</td>
<td>Delete</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Data</th>
<th>Data</th>
<th>Thread</th>
<th>Status</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoice</td>
<td>success</td>
<td>unavailable</td>
<td>Delete</td>
<td></td>
</tr>
<tr>
<td>Check</td>
<td>success</td>
<td>unavailable</td>
<td>Delete</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time Data</th>
<th>Type</th>
<th>Start</th>
<th>End</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>0000-06-30</td>
<td>0000-07-30</td>
<td>Delete</td>
<td></td>
</tr>
<tr>
<td>Hour</td>
<td>08</td>
<td>18</td>
<td>Delete</td>
<td></td>
</tr>
<tr>
<td>Week</td>
<td>01</td>
<td>05</td>
<td>Delete</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Place Data</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NHIC</td>
</tr>
</tbody>
</table>

**Add New Information**

<table>
<thead>
<tr>
<th>Input Data Num</th>
<th>Output Data Num</th>
<th>Time Num</th>
<th>Place Num</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Input Data</th>
<th>Type</th>
<th>Status</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>primitive</td>
<td>unavailable</td>
<td>Delete</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Data</th>
<th>Data</th>
<th>Thread</th>
<th>Status</th>
<th>Action</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Place Data</th>
<th>Location</th>
</tr>
</thead>
</table>

50
Figure 6-6: Template provider-correctness check

Figure 6-7: Template provider-personal process diagram

Figure 6-8: Template provider-process transfer

Transfer OK, You can use

User Name: skin
Template Name: reimbursement
URL: http://140.117.74.228/cgi-bin/service
to retrieve this process
6.2 Personal workflow management system

The PWFMS runs on a hand-hold device. It assists users to manage personal processes by allowing them to download a personal process template (from the template provider), to browse the definition and status of tasks, to place queries, and to automatically track the status of an executing task (from a service provider). Figure 6-9a shows the main menu of the PWFMS. When the user choose ‘Select Process’, a list of personal processes existed in the PWFMS is displayed (see Figure 6-9b). When the user chooses one personal process from the list, the PWFMS will check the status of each executing task of the personal process by invoking Web services. If the status of any task is updated, the correctness of the personal process is automatically validated. In case the PWFMS finds the personal process incorrect, a warning message will be displayed. The user can browse a personal process by clicking on it (Figure 6-10b).

When the user wishes to download a new personal process from the template provider, s/he needs to supply the user id, template id, and URL (see Figure 6-11a). The PWFMS will automatically invoke the corresponding Web service to download the definition of the customized personal process to the PWFMS.

To manipulate tasks of a personal process, the PWFMS provides several parameterized functions (by clicking the left anchor on the ‘Task List’), as shown in Figure 6-11b. These functions include basic queries, frequently used queries as described in Chapter 4, and other maintenance functions. These functions are described below:

**List Available Tasks**: It shows the name of the tasks whose status are ‘Prepared’.

**List Unexecuted Tasks**: It shows the names of the tasks whose status are ‘Initial’.

52
**List Executing Tasks:** It shows the names of the tasks whose status are ‘Executing’.

**List All Tasks:** It shows the names of all tasks.

**Task Required Data:** It allows the user to select a set of tasks and list the aggregated input data items of these tasks.

**Task Related Tasks:** It shows the names of tasks that can be executed after a given set of tasks is completed (query scenario 2 of frequently used queries in chapter 4).

**Co-Executed:** It shows the names of the tasks whose input data are available and can be co-executed with a given task in terms of time and place (query scenario 3 of frequently used queries in chapter 4).

**Available tasks (time):** It shows the names of the tasks that can be executed immediately (query scenario 4 of frequently used queries in chapter 4).

**List Input Data:** It shows the names of the data items that are needed to complete a given set of tasks (query scenario 5 of frequently used queries in chapter 4). A sample output is shown in Figure 6-12a.

**Query Date:** It shows the names of tasks that can be executed in a specific date or date interval. The query screen is shown in Figure 6-12b.

**Task Status Change:** The function allows the user to explicitly specify their execution status. Manual tasks of course need this function. For automatic tasks, although their execution status will be automatically updated via Web services, the user may sometimes need this function to change their execution status (e.g., from ‘Logical Completed’ to ‘Physical Completed’ or from ‘Failed’ to ‘Initial’). When the task status is changed to ‘logical-completed’ and there are more then one thread, the PWFMS will guide the user to select a succeeded thread.

**Task Detail:** It shows the detailed information of a selected task.
Finally, the PWFMS provide functions for manipulating data items (by clicking the left anchor on the ‘Show Data List’) as shown in Figure 6-13. The detailed functions are described in the following:

**List available data:** It will list all available data items.

**List unavailable data:** It will list all unavailable data items.

**List all data:** It will list all data items.

**Data Related Tasks:** It shows the names of the tasks that need a given data item as its input (query scenario 1 of frequently used queries in chapter 4).

**Status change to available:** It changes the status of a given data item to ‘Available’.

**Status change to unavailable:** It changes the status of a given data item to ‘Unavailable’.

**Target Data Set:** It shows the names of the target data items and their status.

---

Figure 6-9: (a) PWFMS-menu (b) PWFMS-process list
Figure 6-10: (a) PWFMS-error message of online check (b) PWFMS-task list

Figure 6-11: (a) PWFMS-getTemplate (b) PWFMS-functions of task
Figure 6-12: (a) PWFMS-list data (b) PWFMS-query date

Figure 6-13: PWFMS-functions of data list
Chapter 7  Conclusions

In this thesis, we have proposed a formal personal process model and a system architecture that adopts Web services for communication between different components. The proposed system facilitates users building a personal process and provides personalized workflow templates of target-driven processes. The PWFMS interface was designed to provide easier browsing and efficient inquiry of a personal process. We have also implemented a prototyping system to demonstrate our concept.

Several components in the proposed PWFS architecture are left unimplemented, including the location aware functions and the active alert functions. This is part of our future work. In addition, how to derive an abstract personal process from a business process that can be used as (part of) a personal process by a customer is also our future work.
References


