無線環境下個人化工作流程系統之研究
The Research on Personal Workflow Systems in Support of Pervasive Computing

研究生：陳雅芳 撰
中華民國九十年七月
# TABLE OF CONTENTS

Chapter 1. Introduction ............................................................................................ 1  
Chapter 2. Personal Process Model ......................................................................... 5  
Chapter 3. Literature Review ................................................................................ 10  
  3.1. Metagraph .................................................................................................... 10  
  3.2. Spatial and temporal data model .............................................................. 13  
  3.3. Related work .............................................................................................. 19  
Chapter 4. Operation Definition ............................................................................ 27  
  4.1. Binary operations ...................................................................................... 27  
    4.1.1. UNION Operation .............................................................................. 27  
    4.1.2. INTERSECTION Operation ............................................................... 28  
    4.1.3. DIFFERENCE Operation ................................................................. 28  
    4.1.4. PLACE_OVERLAP Operation ........................................................... 28  
    4.1.5. TIME_OVERLAP Operation .............................................................. 30  
    4.1.6. MAKE_EXECUTABLE Operation ..................................................... 31  
    4.1.7. NEED_TASK Operation ................................................................... 32  
  4.2. Unary operation .......................................................................................... 34  
    4.2.1. SELECT Operation .......................................................................... 34  
    4.2.2. COMBINED_INPUT Operation ....................................................... 35  
    4.2.3. COMBINED_OUTPUT Operation ..................................................... 36  
Chapter 5. Transformation and Optimization of Query Expression ...................... 37  
  5.1. Transformation of Query Expression ....................................................... 37  
    5.1.1. UNION, INTERSECTION, DIFFERENCE Operation ......................... 37  
    5.1.2. PLACE_OVERLAP Operation ........................................................... 39  
    5.1.3. TIME_OVERLAP Operation .............................................................. 40  
    5.1.4. MAKE_EXECUTABLE Operation ..................................................... 40  
    5.1.5. NEED_TASK Operation ................................................................... 41  
    5.1.6. SELECT Operation .......................................................................... 41  
    5.1.7. COMBINED_INPUT Operation: ....................................................... 42  
    5.1.8. COMBINED_OUTPUT Operation: .................................................... 43  
  5.2. Query Optimization ..................................................................................... 43  
Chapter 6. SQL-Like query language and Rule model ......................................... 48  
  6.1. SQL-Like query language ......................................................................... 48  
  6.2. Rule model ............................................................................................... 50  
Chapter 7. Implementation Status ......................................................................... 55
LIST OF FIGURES

Figure 1. Personal Workflow System Architecture .............................................. 3
Figure 2. The metagraph of a party planning process ........................................... 7
Figure 3. Metagraph ........................................................................................... 11
Figure 4. Overlap Detection ............................................................................. 18
Figure 5. A Workflow Management Model Based on Mobile Agent ................. 25
Figure 6. Performance of query optimization ..................................................... 47
Figure 7. Task Definition ................................................................................... 56
Figure 8: Input Data Definition .......................................................................... 56
Figure 9. Specifying <operations> in SELECT clause ....................................... 57
Figure 10. Specifying (<task attribute> | <data attribute>) in SELECT clause .... 57
Figure 11. Specifying <selection condition> in GIVEN clause ......................... 57
Figure 12. Specifying (<task attribute>| <data attribute>) in GIVEN clause .......... 57
Figure 13. Specifying <condition> in GIVEN clause ......................................... 58
Figure 14. Specifying <value> in GIVEN clause ................................................. 58
Figure 15. Specifying function ......................................................................... 59
Figure 16. Specifying first operand ................................................................. 59
Figure 17. Specifying second operand ............................................................. 59
Figure 18. Finally Result ................................................................................... 59
LIST OF TABLES

Table 1. Relevant Questions About Process Components During Process Analysis..12
Table 2. Operators Defined on The Class Geometry by The OpenGIS .................15
Table 3. Cost and Selectivity of Each Operation..............................................44
Table 4. The Declarative Language ..................................................................48
Chapter 1. Introduction

As wireless technologies are getting their maturity, mobile devices such as PDAs and mobile phones are becoming popular recently. The services provided by these mobile devices enable users to work without space or time limitation. Already many applications have been developed for mobile devices. Typical applications include calendar, memo, address book, to do list, games, email sending/receiving, and web access via WAP. With the increasing number of applications, mobile devices are reaching into virtually every corner of people’s life. However, while these applications allow users to record and retrieve information about tasks and data, the relationships between tasks and data are left out. For pervasive computing core and percepts, an application on mobile device should not only store information but also perform a task [Bana00]. In fact, many of people’s daily activities are not independent, and they are likely to be process-oriented.

Traditional workflow management systems (WFMSs) are mainly used to coordinate business processes in enterprises. These processes must be repetitive and have well-formed structures. Personal processes differ from business processes in the following perspectives:

1. Each personal process instance has a unique structure. In other words, after a process is specified, only one instance is to be executed.

2. From the viewpoint of a personal mobile computer (PMC), activities of a given process are executed by a single execution entity, namely the personal mobile computer’s owner. Traditional workflow scheduling issues, which address how to determine the mapping between activities and available resources, do not seem to exist in this context.
3. Tasks of a process are related mainly due to their data dependency. Unlike enterprises that comprise many regulations and procedures that need to be followed strictly by their processes, human beings seldom impose rigid rules on their personal activities. As a consequence, the execution order of tasks is implied primarily by their input/output data requirements.

4. The coordination between steps of each 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 (PWFS) can do nothing but accept the consequence. In return, the personal workflow system examines the impact of this change and adjusts the unexecuted part of the process according to the owner’s desire.

5. The unique features of PMC, e.g., unexpected disconnection, limited storage, and restricted power energy, play an important role in deciding when and where to obtain resources required for conducting a given process.

These unique features call for a novel design of a personal workflow system. Figure 1 shows the architecture of a personal workflow system. In view of the limited intrinsic nature of PMC, we adopt a client-server paradigm to facilitate mobile users’ inquiries about their personal processes, where the server side is responsible for jobs that require massive storage and/or heavy computation. In particular, the server encompasses three components:

1. Location Server: Location information is essential for answering location dependent queries. In the context of personal workflows, the location information can be used to, for example, remind the mobile user of the next tasks.
to perform when s/he is approaching a specific location. Keeping track of the precise locations of involved users without incurring too much overhead in communication is a non-trivial task and is being intensively researched by many previous works. In cellular systems, the commercial standard such as IS-41 has adopted HLV/VLR scheme for locating mobile callers [EIA91]. Global position system (GPS) based location tracking is still in its infancy and several proposals have recently been proposed [Djuk01].

2. PWF Proxy: Due to computation power and memory limitation, complicated queries cannot be handled solely by PMCs. In this case, PWF Proxy acts as proxy to facilitate the computation of these queries.

3. Process Recommendation System: This component intends to aid mobile users in specifying their personal processes. It does so by searching the personal processes of other mobile users with similar interests and then recommending process templates that mobile users can tailor to meet their specific needs.

![Diagram of Personal Workflow System Architecture](image)

Figure 1. Personal Workflow System Architecture
On the other hand, the client side allows the mobile user to specify and inquire about its personal processes. These queries need to be processed efficiently, possibly with the help from the PWF server.

In this thesis, we focus on the process storage and query processing. In particular, a personal process model that specifies a personal process will be proposed, accompanied by a set of primitive operations that can be used to express queries. Issues on the implementation of these operations will also be discussed. This thesis is structured as follows: In Chapter 2, we will describe the personal process model. Related literature reviews are provided in Chapter 3. Operations on our personal workflow system are formally defined in Chapter 4. Algorithms for implementing these operations are also presented. The algebraic properties about these operations and how to optimize queries comprised of these operations are studied in Chapter 5. In Chapter 6, we design a SQL-like language that allows the declarative query specification and a rule model that enables active reminding. Implementation issues are described in Chapter 7. Finally, Chapter 8 concludes this thesis.
Chapter 2. Personal Process Model

The aim of workflow management systems (WFMSs) is to support the specification and enforcement of business processes [Geor95]. In the context of personal workflows, enforcement actually means reminding users of the next tasks to do or answering users inquiries. In this chapter, we define the semantics of personal processes. A personal process is comprised of the following components:

- a set $T$ of tasks,
- a set $D$ of data,
- several functions that map a task to its name ($\Phi_n$), its input data set($\Phi_i$), its output data set($\Phi_o$), its executable time intervals($\Phi_t$), and its executable places($\Phi_p$):
  - $\Phi_n: T \rightarrow \text{String}$
  - $\Phi_i: T \rightarrow 2^D$
  - $\Phi_o: T \rightarrow 2^D$
  - $\Phi_t: T \rightarrow 2^{\text{Time} \times \text{Time}}$, where Time is the set of time.
  - $\Phi_p: T \rightarrow 2^{\text{Point} \times \text{Point}}$, where Point is the set of geographical points, each of which is expressed as (latitude, longitude),
- a function $\Delta_s: D \rightarrow \text{String}$ that maps a data to its associated name.

In addition, there are attributes that record the execution status of tasks and data. We call these attributes control attributes. In this thesis, we consider two control attributes, $\Phi_s$ and $\Delta_s$, that are associated with tasks and data respectively. $\Phi_s: T \rightarrow$ (UNEXECUTED, EXECUTING, COMPLETED) maps a task to its execution status, namely unexecuted, executing, or completed. $\Delta_d: D \rightarrow$ (UNAVAILABLE, }
AVAILABLE) maps a data item to its availability.

The four functions, $\Phi_i, \Phi_o, \Phi_t, \Phi_p,$ are described as follows: $\Phi_t$ and $\Phi_p$ are *time* and *place* attributes that specify respectively when and where the pertaining task can be performed. $\Phi_i$ and $\Phi_o$ are input and output attributes that specify the sets of data items that this task takes as input and output respectively. Note that input and output attributes together embody the data flow between tasks. As mentioned above, it is the data flow that relates tasks of the same process. For representation purpose, we adopt metagraph for visualizing data dependencies. A metagraph is a graph-theoretic construct that captures relationships between pairs of sets of elements [Basu94]. In its pictorial representation, a set of elements is surrounded by a small cycle, and the edges are arrows connecting the cycles. That is, an edge represents the direction of the input-to-output relationship between two element sets produced by a task. Viewing the input data set as well as the output data set of a task as an element set and a task as an edge, a process that comprises a set of tasks can be represented as a metagraph [Basu00]. Metagraphs are designed to formally express data dependencies of processes. As an example, Figure 2 shows the metagraph of an example party planning process.

We distinguish data in the input and output attributes into two kinds: *primitive* and *processed*. A primitive data is not produced by any task modeled in the system. On the other hand, a processed data must be generated by at least one task. A primitive data could be a data file, a blank form, a personal belonging (e.g., the ID card or a credit card), or anything that is physically available somewhere. A processed data is available only when at least one task that is capable of producing it is completed. For example, the task *sending invitation letters* takes two data items as
input: ‘Invitation Letters’ and ‘Credit Card’. The former is a processed data item, which can only be generated by writing invitation letters, while the latter is a primitive data item.

Figure 2. The metagraph of a party planning process

Though most personal processes require only data dependencies, control dependencies may exist in some exceptional cases. In this case, a control dependency \( T1 \rightarrow T2 \) can be emulated by creating a dummy data item that is produced by \( T1 \) but need by \( T2 \) as input. Therefore, in our model we consider only data dependencies.

Since we view a personal process as a set of tasks and a set of data, traditional set operations like intersection \( (\cap) \), union \( (\cup) \), and difference \( (\neg) \) are applicable in our model. Besides, as tasks and data are associated with a set of predefined attributes, relational operation selection \( (\sigma) \) [Elma00] is also included. Other than these obvious ones, we need further operations that describe process execution.

We propose several operations that intend to answer the users queries. Based on
the types of data they needs, they can be classified into four categories: binary operations with operands $T \times T$, binary operations with operands $D \times D$, binary operations with operands $D \times T$, and unary operations with operand $T$.

1. $T \times T \rightarrow T$: These are binary operations that take operands of type $T$. These operations include UNION, INTERSECTION, DIFFERENCE, TIME_OVERLAP, and PLACE_OVERLAP. UNION, INTERSECTION and DIFFERENCE operations are basic set operations. TIME_OVERLAP (PLACE_OVERLAP) is used for retrieving a subset of tasks in the first operand whose execution times (places) overlap with some task in the second operand.

2. $D \times D \rightarrow D(T)$: UNION, INTERSECTION, and DIFFERENCE operations are also applied to operands of type $D$. In addition, we propose a new operation NEED_TASK that returns a set of tasks which take data items in the first operand as the input and produce the data set in the second operand.

3. $D \times T \rightarrow T$: One operation MAKE_EXECUTABLE is proposed. It identifies a subset of executable tasks in the second operand while given the data items in the first operand as the input.

4. $T \rightarrow D$: Two operations, namely COMBINED_INPUT and COMBINED_OUTPUT, are proposed that return the aggregate input and output respectively for executing the tasks in the first operand.

In the following, we illustrate the power of the operators proposed above by showing several concrete query examples:

1. Find the set of tasks that produces ‘receipt’.
   $$\sigma_{\text{available}(D)} \text{ NEED_TASK } \sigma_{n=\text{receipt}}(D)$$

2. Find a set of tasks that can be co-executed with ‘buying invitation cards’, when
'planning' and 'finding a place' are both completed.

\( \sigma_{\text{available}}(D) \ \text{UNION} \ \text{COMBINED\_OUTPUT} \ \sigma_{\text{n='planning'}}(T) \ \text{UNION} \ \sigma_{\text{n='finding a place'}}(T) \ ) \ \text{MAKE\_EXECUTEABLE} \ \sigma_{\text{n='unexecuted'}}(T) \ \text{TIME\_OVERLAP} \ \sigma_{\text{n='buying invitation cards'}}(T) \ \text{PLACE\_OVERLAP} \ \sigma_{\text{n='buying decoration stuff'}}(T) \)

Find the set of data that are needed to complete tasks ‘finding a place’ , ‘buying invitation cards’, and ’buying decoration stuff’.

COMBINED\_INPUT \ (\sigma_{\text{n='finding a place'}}(T) \ \text{UNION} \ \sigma_{\text{n='buying invitation cards'}}(T) \ \text{UNION} \ \sigma_{\text{n='buying decoration stuff'}}(T) )

4. Retrieve a set of tasks with a specified data item (‘data’) as part of its input.
\( \sigma_{\text{i='data'}}(T) \)

5. Retrieve a set of tasks whose executions results in the generation of a specified data item (data).
\( \sigma_{\text{o='data'}}(T) \)

6. To find the set of tasks that can be executed immediately.
\( \sigma_{\text{available}}(D) \ \text{MAKE\_EXECUTEABLE} \ \sigma_{\text{n='unexecuted'}}(T) \ \text{TIME\_OVERLAP} \ \text{CURRENT\_TIME} \ \text{PLACE\_OVERLAP} \ \text{CURRENT\_LOCATION} \)

From the above examples, we can see that there are infinite ways to combine these operations so as to express various queries. Besides, using a high-level query language as a glue, we are capable of answering users’ inquiries on process information with constraints on time, place, and data dependencies. Description of the high-level query language will be provided in Chapter 6.
Chapter 3. Literature Review

As mentioned in Chapter 2, we use a metagraph to represent the data dependencies of a personal process. We will review the concept of metagraph in this chapter. Besides, as users’ inquiries may involve temporal and spatial constraints, we will briefly survey the spatial and temporal data models proposed in the literature. There have been several proposals for designing workflow systems for mobile environments. We will also survey these proposals.

3.1. Metagraph

Metagraph is a graphical structure that represents relationship between multiple inputs and outputs. Metagraph is the method of representing data dependencies of a personal process. Metagraph is defined as follows [Basu94]:

**DEFINITION:** Consider a finite set \( X=\{x_i, \ i=1 \square I\} \), called a generating set. A metagraph on \( X \), denoted \( S \), is an ordered pair \(<X,E>\), in which \( E=\{E_k, k=1 \square K\} \) is a set of edges. Each \( E_k \) is an ordered pair \(<V_k,W_k>\), in which \( V_k \subseteq X \) is the in-vertex of \( E_k \) and \( W_k \subseteq X \) is the out-vertex and \( V_k \cup W_k \neq \Phi \) for \( k = 1 \cdots K \).

For example, consider a metagraph in which \( X=\{x_1 \square x_6\} \) and \( E=\{E_1,E_2,E_3\} \), \( E_1=<\{x_1\},\{x_1, x_4\}> \), \( E_2=<\{x_2, x_3\},\{x_5, x_6\}> \), \( E_3=<\{x_5, x_6\},\{x_6\}> \). Its diagrammatical representation is shown in Figure 3.
The Metagraph model can be used to specify the data flow between tasks. This information is either missing or not stressed in traditional workflow model, as most workflow models focus on specifying control flow rather than data flow.

**DEFINITION:** Given a generating set \( X = \{x_i, i = 1 \cdots I\} \) and a metagraph \( S = (X, E) \) with \( E = \{E_k, k = 1 \cdots K\} \), the adjacency matrix, \( A \), of \( S \) is an \( I \times I \) matrix with each \( a_{ij} \) for \( i, j \in \{1 \cdots I\} \) defined as follows:

\[
 a_{ij} = \bigcup_{k=1}^{K} (\alpha_{ij})_k \text{ where }
\]

\[
 (\alpha_{ij})_k = \begin{cases} 
 <V_k \setminus \{x_i\}, W_k \setminus \{x_j\}, <E_k>> & \text{if both} \ x_i \in V_k \ \text{and} \ x_j \in W_k \\
 \Phi & \text{otherwise.}
\end{cases}
\]

The adjacency matrix \( A \) of a metagraph is a square matrix with one row and one column for each element in the generating set. Each element \( A_{ij} \) comprises a set of 3-tuples, each of which identifies a task that leads input data item \( x_i \) to output data item \( x_j \). The first component of a 3-tuple is the set of input data items other than \( x_i \), the second component is the set of output data items other than \( x_j \), and the third component is the edge \( E_k \) connecting \( x_i \) to \( x_j \). Adjacency matrix is a base structure for...
computing possibility sequence of tasks.

Metagraph allows us to solve questions during process analysis, these question are illustrated in Table 1[Baus00].

Table 1. Relevant Questions About Process Components During Process Analysis

[Basu00, pp. 20]

<table>
<thead>
<tr>
<th>Process Component</th>
<th>Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information Elements</td>
<td>1. Given two information elements, is one of them needed to determine the value of the other? Is it needed only under certain conditions and if so, what are the conditions?</td>
</tr>
<tr>
<td></td>
<td>2. Given two sets of information elements, is it possible to determine the value of the second set from the elements in the first set? If not, are there any additional information elements that would make it possible to do so?</td>
</tr>
<tr>
<td></td>
<td>3. Given a complex process, are there any ways to focus on only important information elements, hiding intermediate elements that are needed only to calculate the important ones?</td>
</tr>
<tr>
<td>Tasks</td>
<td>4. Given a task that we wish to execute, what other tasks must be executed to provide the information needed to do so?</td>
</tr>
<tr>
<td></td>
<td>5. If a task is disabled, what other tasks will be affected—that is, what other tasks cannot be executed?</td>
</tr>
<tr>
<td>Resources</td>
<td>6. Given a set of resources, what information passes among them as the tasks that utilize them are executed?</td>
</tr>
<tr>
<td></td>
<td>7. If a resources is unavailable, what other resources and affected—that is, what other resources will be idle because their tasks cannot be executed?</td>
</tr>
<tr>
<td>Interactions among Components</td>
<td>8. If a resource used in a process is unavailable, which workflows within the process can still be completed?</td>
</tr>
<tr>
<td></td>
<td>9. If an information element is found to be inaccurate, which resources were used, directly or indirectly, in the calculation of that element?</td>
</tr>
</tbody>
</table>

[Baus00] have proposed the operations associated to adjacency matrix. These
operations allow us to answer a great number of queries about processes such as those listed in Table 1. Therefore, it is possible to make use of these operations to implement our proposed operations, namely NEED_TASK(), COMBINED_INPUT(), and COMBINED_OUTPUT(). However, this straightforward approach may not be efficient. We will describe more efficient algorithms in Chapter 4.

3.2. Spatial and temporal data model

In our process model, two functions, namely $\Phi_t$ and $\Phi_p$, map a task to its executable time intervals and executable places respectively. For defining domain of time intervals and places, we should probe into the data model about spatial and temporal type. [Forl00] described the abstract modeling of spatial-temporal data types as follows:

\[
\begin{align*}
\rightarrow BASE & \quad \text{int, real} \\
& \quad \text{string, bool} \\
\rightarrow SPATIAL & \quad \text{point, points,} \\
& \quad \text{line, region} \\
\rightarrow Time & \quad \text{instant} \\
BASE \cup TIME & \quad \rightarrow RANGE \quad \text{range} \\
BASE \cup SPATIAL & \quad \rightarrow TEMPORAL \quad \text{intime, moving}
\end{align*}
\]

Except for extending a special value “undefined”, types of int, real, string, and bool, are defined as usual. A value of point is given as a pair (x,y) of coordinates, which represents a point in the real (2D) plane, and points type is composed of a finite set of points. A set of continuous curves or any collection of line segments in the plane is the value of line type. A region value is composed of a finite set of disjoint faces where each face can be viewed as a polygon with polygon holes in the plane. Type instant is represented as real numbers, and intime type yields a time instant with a value of argument domain. The value of moving type is a function from time to the
domain of given $\alpha$. All types of abstract model will correspond to discrete model except for moving type. The abstract model also defines a set of operations such as predicates (e.g. inside or $\leq$), set operations (e.g. union), aggregate operations, operations with numeric result (e.g. size of a region) and a distance and direction operations.

Spatial operators are used to capture all the relevant geometric properties of objects embedded in the physical space and to perform spatial analysis. In OpenGIS described by Clement et al, several geometric data types, including point, linestring, polygon, multipoint, multilinestring, multipolygon, etc., were designed, and several operators, as listed in Table 2, were proposed to manipulate geometric objects [Clem00]. Spatial operators were classified into topological, projective, and metric. Topological operators represent the most primitive kind of spatial information. Topological properties are invariant with respective to transformations like rotation and translations. Projective properties are related to straight or curvilinear, the number of vertices of a polygon, the number of concavities of a contour, which cannot be expressed in topological properties. A basic projective operator is the convex hull of region. For example, “using a convex hull projective operator to represent houses whose geometry is inside that of a given forest”, a SQL-like query can be specified as follows [Clem00]:

```sql
SELECT House.id
FROM Forest, House
WHERE insideConvexHull(House.geo, Forest.geo)
```

Metric operators represent more special description than topological and projective. For a single object, the description such as the area, the relative size of object’s compactness, symmetry, and so on. Between different objects, metric
operators measure the relative position in terms of distance and direction. For example, the query “show me all the cities at most 100 Km away from Rome” can be expressed by the following the SQL-like statement [Clem00]:

```sql
SELECT c2.name
FROM City as c1, City as c2
WHERE Distance (c1.location, c2.location) <= 100km
AND c1.name = "Rome"
```

<table>
<thead>
<tr>
<th>Basic Operator</th>
<th>SpatialReference</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpatialReference</td>
<td>Returns the Reference systems of the geometry</td>
</tr>
<tr>
<td>Envelope</td>
<td>The minimum bounding rectangle of the geometry</td>
</tr>
<tr>
<td>Export</td>
<td>Convert the geometry into a different representation</td>
</tr>
<tr>
<td>IsEmpty</td>
<td>Tests if the geometry is the empty set or not</td>
</tr>
<tr>
<td>IsSimple</td>
<td>Returns True if the geometry is simple</td>
</tr>
<tr>
<td>Boundary</td>
<td>Returns the boundary of the geometry</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Topological Operator</th>
<th>Geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal</td>
<td>Tests if the geometries are spatially equal</td>
</tr>
<tr>
<td>Disjoint</td>
<td>Tests if the geometries are disjoint</td>
</tr>
<tr>
<td>Intersect</td>
<td>Tests if the geometries intersect</td>
</tr>
<tr>
<td>Touch</td>
<td>Tests if the geometries touch each other</td>
</tr>
<tr>
<td>Cross</td>
<td>Tests if the geometries cross each other</td>
</tr>
<tr>
<td>Within</td>
<td>Tests if the given geometry is within another given geometry</td>
</tr>
<tr>
<td>Contains</td>
<td>Tests if the given geometry contains another given geometry</td>
</tr>
<tr>
<td>Overlap</td>
<td>Tests if the given geometry overlaps another given geometry</td>
</tr>
<tr>
<td>Relate</td>
<td>Returns True if the spatial relationship specified by the 9-Intersection matrix holds</td>
</tr>
<tr>
<td>Spatial Analysis Operator</td>
<td>Distance</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td></td>
<td>Buffer</td>
</tr>
<tr>
<td></td>
<td>ConvexHull</td>
</tr>
<tr>
<td></td>
<td>Intersection</td>
</tr>
<tr>
<td></td>
<td>Union</td>
</tr>
<tr>
<td></td>
<td>Difference</td>
</tr>
<tr>
<td></td>
<td>SymDifference</td>
</tr>
</tbody>
</table>

Similar to relational DBMSs, a spatial relation is a collection of spatial objects defined on the same attributes. An efficient implementation of spatial queries is important for retrieving suitable spatial objects of relations. In a spatial DBS, spatial queries include point queries, window queries, nearest neighbor queries, and spatial joins [Brin94]. One of the most important spatial queries is spatial join, which matches two or more relations. Spatial join operates on a given predicate and retrieves satisfied objects from relations. Predicate includes overlap, distance, intersect, and so on. An example of spatial joins is 'find all forests which are in a city'. Efficient query processing algorithms of spatial joins have been proposed in many researches. One of the well-known approaches is Rote algorithm [Rote91], it uses a multidimensional index structure called grid file to accommodate point data, and generates a join index from two given grid files. In the two dimensional case, each entry includes a region-id and two pairs of coordinate giving the bottom left and the top right corners of each region. The algorithm assumes that each file $G_i$ has an associate region directory $RD_i$. The region directory is a structure, which contains $n_i$ entries, and is used in overlap detection. Assuming an imaginary line called plane.
sweeping, it parallels to the y axis, and scans region from left to right. At each point in
time, all regions cut by the line are called candidate regions. The detection of an
overlap means two regions are candidate at the same time. The detection overlap
algorithm is described in detail as bellow [Rote91]:

1. Three auxiliary structures are constructed for algorithm: A set $X$ of vectors, and a
pair of arrays $Y_1$ and $Y_2$.

A vector with the structure: $<\text{value}, \text{bt}, \text{region-id}, \text{file-id}>$ is constructed. In the
structure, value is the value of an x coordinate. If this is a left coordinate, bt is 0,
otherwise bt is 1. The region-id is the region number for this coordinate. The file-id is
either 1 or 2 depending on the region directory $RD_i$. These vectors are elements of a
set $X$. The set $X$ has a cardinality of $2t$, where $t=n_1+n_2$ since each region contributes
two coordinates. A pair of arrays $Y_1$ and $Y_2$ are represented in $Y_1[n_1][2]$ for y
coordinate of $RD_1$, and $Y_2[n_2][2]$ for y coordinate of $RD_2$. For example, an entry with
two coordinates (3,6), (12,15), respectively bottom left and top right, and the region id
in $RD_1$ is 8. The vectors in $X$ are $<3,0,8,1>$ and $<12,1,8,1>$, and a pair of entries
$Y_1[8][0]=6$ and $Y_1[8][1]=15$ in $Y_1$. In addition to set $X$, the set $\vec{X} = (x_1, x_2, \ldots, x_n)$ is
generated for scanning detection points. It sorts the elements in $X$ by first component
as major and second component as minor.

2. Three case for overlap detection:

Consider two regions $i$ and $l$ where region $i$ from $RD_1$ and current candidate
region called region $l$ from $RD_2$. Two regions will overlap if the condition (1)
$Y_2[l][0] \leq Y_1[i][0] \leq Y_2[l][1]$ or (2) $Y_1[i][0] \leq Y_2[l][0] \leq Y_1[i][1]$ holds. Constructing a
binary search tree $B_1$ structure based on bottom coordinates as key for detection. Each
node $s$ in $B_1$ (see Figure 4(d)) contains three elements $<BOT_s;TOP_s;REG_s>$. The
elements $BOT_i$, $TOP_i$ are described bottom and top coordinates of $REG_i$. Candidate regions of $RD_i$ are also constructed as similar binary tree called $B_2$. Then, three cases for overlap detection are below:

**Case I: There is no overlap between regions (see Figure 4(a)).**

$Y1[i][1]$ is smaller than any key in $B_2$ or $Y1[i][0]$ is bigger than any key in $B_2$.

**Case II: There is an overlap between regions (see Figure 4(b)).**

If a node $w$ exists in $B_2$ such that $BOT_w ≤ Y1[i][0] ≤ TOP_w$, searching forward from that node until reaching the end of the tree or a node $y$ such that $BOT_y ≤ Y1[i][1] ≤ TOP_y$.

**Case III: There is an overlap between regions (see Figure 4(c)).**

Searching from the leftmost node in $B_2$ until some node $a$ is found where $BOT_a ≤ Y1[i][1] ≤ TOP_a$.

![Figure 4. Overlap Detection](Rote91 pp.504)
3.3. Related work

The applications and services with integrated personal information on pervasive computing called Personal Information Services and Application (PISA) were identified by [Elma95] as future challenging area. They proposed a wireless client/server computing architecture for delivery of PISA, and PISA for wireless client/server computing were characterized as follows:

1. Asymmetrical communication between clients and servers. Receiving messages is considered less costly than sending message for mobile hosts.

2. Long disconnections of clients. Mobile clients will connect only for getting services. However, they are likely to stay in a disconnected or weakly-connected mode most of the time.

3. Virtual mobility of servers. Mobile hosts often connect to different access points during their move. In generally, servers near by mobile host will provide services more efficiently. Therefore, virtual mobility of servers should be introduced into wireless computing architecture.

For delivery of PISA under such an architecture, data management issues such as transaction services and cache consistency were examined in detail. In one way of processing transactions, a mobile host may submit a transaction in a single request message, deliver execution control to a single coordinator, and then wait the return of execution results. In another way, a mobile host may submit a transaction as multiple request massages, each of which is delivered to a distinct coordinator because the mobile host keeps changing current location (eg. moving into a new cell). Because of high latency of wireless communications, long disconnections, and mobility of clients,
a mobile transaction trends to be long-lived, error-prone and access distributed and heterogeneous databases. Therefore, modeling mobile transactions as ACID transactions may be very restrictive. Data consistency and services handoffs were discussed in transaction services. Some possible techniques to maintain consistency of data under disconnection were proposed:

1. Read-any/write-any weakly consistent replications: Replicated data is provided while completely disconnected from the rest of computing environment.

2. Deferred convergence control for eventual consistency of replicated data: Replicated data propagate if there are no new updates.

3. Dependency checks on each write to support detection of update conflicts.

By the character of mobility, services handoffs mechanism can usefully reduce transmission delays. Managing transaction logs efficiently from service handoffs is also an important issue. In generally, solutions for management of distributed transaction logs included:

1. A global commitment protocol is used to restore log information back into database at a commit time.

2. The log transfers during service handoffs and local commitment of transaction at the commit time.

3. Log information related to commutative operations can be left in different servers until non-commutative operations on the same data from other transactions are executed. It can reduce message cost for mobile transactions.

Another data management issue is cache consistency. Caching of frequently accessed data will improve query response time and reduce delivering on narrow bandwidth. Traditional caching approaches can be divided into callback and detection
categories. Callback approach is that servers send invalidation messages directly to clients. Detection approach is clients send queries to servers to validate cached data. Each of these approaches is not applicable to mobile computing, since mobile hosts often stay in disconnected mode or move frequently. A new approach that servers broadcast an invalidation report with changed data periodically was proposed to adapt to the mobility and weak connected mode of a mobile host.

In [Huan99], an architecture namely Rome, was proposed to manage triggers at a centralized infrastructure. The Rome architecture is composed of four parts. The Fronted part provides users an interface to enter task information, and forward to Semantic Translator part for translating high-level tasks into a set of triggers. Then, Trigger Manager couples with Unit Manager part to deliver triggers to appropriate end-devices (e.g. routing trigger with place condition to device equipped with GPS, routing trigger with time condition to a PDA). A trigger is defined as a self-describing chunk of information bundled with the spatial and/or temporal constraints that define the context in which the information should be delivered. By this means, mobile users will be alerted to do something when the trigger conditions have been satisfied. The main contribution of the architecture is an infrastructure-centric approach to the trigger management problem. The described work is similar to the triggering system proposed in our PWFS architecture, as described in Chapter 6.

In summary, [Elma95] and [Huan99] works intend to provide more services to personal activities rather than personal processes.

For integrating workflow management and pervasive computing, two adaptive strategies are proposed in [Jing99] to deal with dynamics of mobile resources in workflow resource management and the constraints of bandwidth in electronic
Mobile resource management is concerned with assigning resource (person or mobile users) to perform work activities. One way of assignment called optimistic policy was proposed that assign resources according to their types (for example, personal position, professional skill). However, it might not be suitable when the resources are dynamically changed (such as a mobile user change his location and far away from the executing location) or disconnected. Another way of assignment called pessimistic policy delays assignment until the resource is connected and ready to select work items for execution. It also has a problem when the number of work items and resources are very large. An adaptive strategy called hybrid assignment policy was proposed to combine optimistic and pessimistic policies to support mobile resource management. A resource can be treated as a controlled resource if its status has been within the deviation bound for the last \( m \) (a threshold value) connected, then applying optimistic policy. Otherwise, this resource is seen as uncontrolled and applying pessimistic policy.

In face of the constraints of bandwidth, three strategies for delivering electronic documents were proposed:

1. Background pre-fetching: the idea is to pre-fetch documents for each assigned work item into the user’s mobile device and rebind the fetched data to the work item before it is processed. The fetching procedure is based on the user’s speed of document processing, the available network bandwidth, and the size of a document data.

2. On-line fetching strategy: users can be given options to decide at what quality level these data can be fetched at the on-line time when the document for the work item is opened and being transmitted for processing.
3. Complementary and collaborative capabilities: it incorporates the above two strategies to maximize the benefits. For each type of background pre-fetching and on-line fetching, this strategy decides how the available bandwidth is allocated, what type of data and what quality level in each document should be fetched. For example, when the communication bandwidth is low and background pre-fetching cannot match the user’s speed of document processing, on-line fetching strategy is adopted instead.

A workflow enhancements for mobility prototype system called WHAM was proposed in [Jing00]. For long disconnection and mobility features of mobile hosts, a two-level resource management approach was used. Global resource management (GRM) is responsible for (1) resource specification and synchronization, and (2) adaptive work activity assignment. Each mobile resource in resource database is modeled as an object which consists of name, type, capabilities, and status attributes. A status attribute includes location (where the resource is located), state (whether the resource is available), and load (the waiting time needed to perform the assigned activities). Value of resource status attribute can dynamically be changed, e.g., location is changed when a mobile resource is moving. Load attribute could be a function of locations. For example, driving time should be included into actual waiting time (load) if a mobile resource is away from work activities executing site. A work activity assignment approach combined optimistic and pessimistic policies, as described above, for assigning mobile resources. The second level management is local resource management (LRM), which maintains a local resource profile. The goal of LRM is to prioritize and filter work activities assigned. A resource profile consists of two resources attributes information as follows:
1. Work preference attribute of human resource. The additional work preference such as the route of a work that a mobile resource will take. LRM can prioritize assigned work activities according to work preference profile for getting optimized load.

2. Application-specific resource type and status. An example is the equipment parts carried by a mobile resource and the number of available parts. LRM uses filtering function to filter out the activities that can be done with current resource status.

A novel workflow management model based on mobile agent (see Figure 5) was proposed by [Gang00]. A mobile agent is the core of their model. It comprises code, data, execution state parts, and is an active autonomous computation entity to take charge of communicating with environment and other entities (e.g., humans, machines). This architecture includes two parts: workflow agent developing environment and workflow execution environment. The first part is workflow agent developing environment (WADE). WADE is composed of process definition tools and agent developing module. Base on process definition tools, the workflow agent is designed and programmed by developing module. Workflow execution environment (WEE) is the other part in the architecture. It can be divided into a workflow management environment (WME) and a mobile agent runtime environment (WARE). WME includes workflow management model (WMM) and an agent base. Workflow management model (WMM) takes charge of the following tasks: receiving workflow agent from developing environment, storing them to agent base, interacting with users to fetch fit agent, and then sending to runtime environment for execution. Mobile agent runtime environment (MARE) is supporting execution, mobility, life-cycle
management and other function of the agents.

![Workflow Agent Developing Environment](image)

Figure 5. A Workflow Management Model Based on Mobile Agent [Gang00 pp.185]

The advantages of this novel model are as follows:

1. Loose-cooperating architecture. Mobile agent mechanism can support to adjust the workflow process for the changes of loose-cooperating relationship. The difference between tradition and novel workflow is matching operation delayed to the workflow runtime.

2. Heterogeneous distributing platform. Each participant needs only a mobile agent, which is independent from the system platform.

3. Autonomy. The workflow agent can decide the next destination or operate according to the current status of the environment.

4. Concurrent computing. An agent might concurrently spawn several sub-agents to reduce the overall process time and merge when sub-agents finishing their tasks.
5. Workflow recovery and fault tolerance. A mobile agent lifecycle contains initializing, moving, completing and persistence storing. Persistence storing and encapsulation state enhance recovery process and fault tolerance.

Aspects of next generation enterprises (NGE) in [Karu00] includes integrating vendor and suppliers through internet, allowing users access services from diversely devices by both wire-line and wireless, facilitating dynamic service creation and disassembly of services when not needed. Mobile application, which has features as mobility and limited devices, is one of the important classes of NGE applications. Mobile applications such as information services providers, which allow user to register interesting events, and receive information they are interested in actively/passively through mobile devices. A generic mobile application framework is introduced in integrating wireless networks, mobile agents, wireless middleware, workflow, data replication, adapters, etc.

Although these works ([Jing99],[Jing00],[Gang00],[Karu00]) were concerned on processes, their focus is on extending workflow technology to support mobile clients or mobile workforces for business processes, rather than processing and coordinating activities for personal processes.
Chapter 4. Operation Definition

Based on the personal process model described in Chapter 2, we propose several operations that intend to answer the users queries. These operations have been briefly described in Chapter 2. In this chapter, we will give a formal definition for each operation and describe its algorithm. We further distinguish between unary and binary operations to assess properties of single objects or relations between objects.

4.1. Binary operations

Binary operations include UNION, INTERSECTION, DIFFERENCE, PLACE_OVERLAP, TIME_OVERLAP, NEED_TASK, and MAKE_EXECUTABLE.

4.1.1. UNION Operation

UNION operation is used to combine two sets of the same type, which could be tasks or data. UNION operation is denoted as $\bigcup$.

**DEFINITION:** Given two sets $S_1$ and $S_2$ of items with the same type (data or task), UNION operation produces a set that includes all items in $S_1$ or $S_2$, or both $S_1$ and $S_2$. $S_1 \cup S_2 = \{ x \mid x \in S_1 \text{ or } x \in S_2 \}$.

Suppose there are $N_{T1}$ elements in $S_1$ and $N_{T2}$ elements in $S_2$. The running time of the operations $\cup$ is $(N_{T1}+N_{T2})$ without duplication elimination and may be up to $(N_{T1} \times N_{T2})$ with duplication elimination by using nested loop approach [Elma00]. Since the volume of personal process is usually small, this straightforward implementation is sufficient most of the time.
4.1.2. INTERSECTION Operation

INTERSECTION operation is used to identify the duplications between tasks sets or data sets. INTERSECTION operation is denoted as $\cap$.

**DEFINITION**: Given two sets $S_1$ and $S_2$ of items with the same type (data or task), INTERSECTION operation produces a set that includes all items both in $S_1$ and $S_2$. $S_1 \cap S_2 = \{ x \mid x \in S_1 \text{ and } x \in S_2 \}$.

Suppose there are $N_{T1}$ elements in $S_1$, $N_{T2}$ elements in $S_2$ The running time of the operation $\cap$ is $(N_{T1} \ast N_{T2})$ with the simple nested loop implementation.

4.1.3. DIFFERENCE Operation

DIFFERENCE operation is used to locate the difference between two task sets or data sets. DIFFERENCE operation is denoted as $-$.

**DEFINITION**: Given two sets $S_1$ and $S_2$ of items with the same type (data or task), DIFFERENCE operation produces a set that includes all items in $S_1$ that are not in $S_2$. $S_1 - S_2 = \{ x \mid x \in S_1 \text{ and } x \notin S_2 \}$.

Suppose there are $N_{T1}$ elements in $S_1$ and $N_{T2}$ elements in $S_2$. The running time of the operation $-$ is $(N_{T1} \ast N_{T2})$ with the simple nested loop implementation.

4.1.4. PLACE_OVERLAP Operation

PLACE_OVERLAP is used for retrieving tasks in the first operand whose executable regions overlap that of some task in the second one. PLACE_OVERLAP operation is denoted as $\circ_p$.

**DEFINITION**: Given two sets $S_1$ and $S_2$ of tasks. PLACE_OVERLAP produces
a subset of $S_1$ whose executing place overlaps with the executing place of some task in $S_2$. Formally,

$$S_1 \circ_p S_2 = \{ t1 | t1 \in S_1, \exists t2 \in S_2, \exists p1 \in t1.p, \exists p2 \in t2.p, \text{RECTANGLE}_\text{OVERLAP}(p1, p2) \}$$

Suppose there are $N_{T1}$ elements in $S_1$, $N_{T2}$ elements in $S_2$, and each of $N_{T1}$ and $N_{T2}$ has $N_p$ regions in place attribute. The running time of the operation $\circ_p$ is $(N_{T1} \cdot N_{T2} \cdot N_p^2)$ with the following nested loop implementation.

**Algorithm:**

```plaintext
PLACE_\text{OVERLAP}(S_1, S_2){
    S <- \Phi;
    for_S1: For each task $t1$ in $S_1$
        For each place $p1$ in $t1$
            For each task $t2$ in $S_2$
                For each place $p2$ in $t2$
                    If $p1$ rectangle_overlap with $p2$
                        $S = S \cup \{t1\};$ continue for_S1;
                    }
    }
    */ for_S1 */
    Return $S$
}
```

As many researches have proposed algorithms for detecting overlapping, we will just adopt one such algorithm proposed in the literature. Specifically, we use the algorithm proposed in [Rote91] (which has been described in Chapter 3) to meet our purpose. This algorithm is well-known and has been used in a broad range of application domains.
4.1.5. TIME_OVERLAP Operation

Similar to its counter part PLACE_OVERLAP, TIME_OVERLAP is used for retrieving tasks in the first operand whose executable time intervals overlap that of some tasks in the second one. TIME_OVERLAP operation is denoted as $o_1$.

**DEFINITION:** Given two sets $S_1$ and $S_2$ of tasks, TIME_OVERLAP produces a subset of $S_1$ whose executing time overlaps with the executing time intervals of some task in $S_2$. Formally,

\[
S_1 o_1 S_2 = \{ t_1 | t_1 \in S_1, \exists t_2 \in S_2, \exists i_1 \in t_1.t, \exists i_2 \in t_2.t, \text{ INTERVAL OVERLAP}(i_1,i_2) \}
\]

**Algorithm:**

```plaintext
TIME_OVERLAP (S1,S2){
    S←∅;
    for_S1:  For each task t1 in S1{
        For each time interval i1 in t1 {
            For each task t2 in S2{
                For each time interval i2 in t2 {
                    If i1 interval_overlap with i2
                        S=S∪{t1}; continue for_S1;
                    }
                }
            }
        }
    }
    */ for_S1 */
    Return S
}
```

Suppose there are $N_{T1}$ elements in $S_1$, $N_{T2}$ elements in $S_2$, and each task in $S_1$ and $S_2$ has $N_i$ intervals in time attribute. The running time of the above algorithm for computing $o_1$ is $(N_{T1}*N_{T2}*N_i^2)$. 

~30~
4.1.6. MAKE_EXECUTABLE Operation

MAKE_EXECUTABLE is used for retrieving a subset of executable tasks in the second operand while given the data items in the first operand as the input. MAKE_EXECUTABLE operation is denoted as \( \rightarrow \).

**DEFINITION:** Given a set of tasks \( T \) and a set of data \( D \), MAKE_EXECUTABLE returns a subset of \( T \), each of which requires a subset of \( D \) as the input. Formally, \( D \rightarrow T = \{ t \mid t \in T, t.i \subseteq D \} \)

**Algorithm:**

MAKE_EXECUABLE\((D,T)\) {
    \( S \leftarrow \Phi \);
    For each task \( t \) in \( T \) {
        For each data \( d \) in \( t.i \) {
            If \( d \notin D \) break;
        }
        \( S = S \cup \{ t \} \);
    }
    Return \( S \)
}

Suppose there are \( N_T \) elements in \( T \), \( N_D \) elements in \( D \), and each task in \( T \) has \( N_{in} \) input data items. In the algorithm, each input data elements of a task takes time \( O(N_D) \) in If clause. As there are \( N_{in} \) elements in inner For clause, and \( N_T \) elements in outer For clause, The running time of the above algorithm for computing \( \rightarrow \) is \( (N_T * N_{in} * N_D) \).
4.1.7. NEED_TASK Operation

NEED_TASK operation is used for retrieving a set of tasks, whose execution requires the data set in the first operand and produces the data set in the second operand. More specifically, NEED_TASK returns a meta path between an input data set and an output data set. As mentioned in [Basu00], given a pair of data sets as input and output respectively, there may exist more than one meta path. NEED_TASK returns the meta path that has the lowest cost, where the cost of a meta path is defined as the sum of the priorities of its constituent tasks. We assume each task is assigned a priority value by the mobile user. Smaller priority value means higher priority. NEED_TASK operation is denoted as \( \rightarrow \).

**DEFINITION:** Given two data sets \( D_1 \) and \( D_2 \), NEED_TASK operation produces a set of tasks \( T \), \( T \) constitutes a meta-path with \( D_1 \) and \( D_2 \) being the input and output data sets respectively. Besides, \( T \) has the lowest cost among all meta-paths that connect \( D_1 \) and \( D_2 \). Let \( \text{MinMetaPath}(D_1, D_2) \) be a function that returns the minimum cost of the meta path that connects \( D_1 \) and \( D_2 \). \( \text{MinMetaPath}(D_1, D_2) \) can be defined recursively as follows:

\[
\text{MinMetaPath}(D_1, D_2) = 0 \quad \text{if } D_2 \subseteq D_1
\]

\[
\text{MinMetaPath}(D_1, D_2) = \min_{(T' \subseteq D_2)} \left( \text{MinMetaPath}(D_1, \text{Input}(T')) + \text{Cost}(T') \right);
\]

where \( \text{Input}(T') \) and \( \text{Output}(T') \) denote the input data set and output data set of a task set \( T' \) respectively. Besides, \( \text{Cost}(T') = \sum_{t \in T'} \text{Priority}(t) \).

For example, consider the following ‘A party planning process’ depicted in Figure 2. Suppose the data items ‘Invitation-List’ and ‘Time’ \( (D_1) \) are already available. The user may be interested in knowing what tasks need to be done in order
to get ‘Receipt’ ($D_2$). The function $MinMetaPath(D_1, D_2)$ can be recursively computed as below:

$$MinMetaPath(\{ \textit{Invitation List}, \textit{Time} \}, \{ \textit{Receipt} \})$$

$$=MinMetaPath(\{ \textit{Invitation List}, \textit{Time} \}, \{ \textit{Invitation Letters}, \textit{Credit Card} \}) + \text{Priority}(\text{sending invitation letters});$$

$$=MinMetaPath(\{ \textit{Invitation List}, \textit{Time} \}, \{ \textit{Invitation Card} \}) + \text{Priority}(\text{sending invitation letters}) + \text{Priority}(\text{writing invitation letters});$$

$$=MinMetaPath(\{ \textit{Invitation List}, \textit{Time} \}, \{ \textit{Invitation List}, \textit{Time}, \textit{Place} \}) + \text{Priority}(\text{sending invitation letters}) + \text{Priority}(\text{writing invitation letters}) + \text{Priority}(\text{buying invitation cards});$$

$$=MinMetaPath(\{ \textit{Invitation List}, \textit{Time} \}, \{ \textit{Invitation List}, \textit{Time} \}) + \text{Priority}(\text{sending invitation letters}) + \text{Priority}(\text{writing invitation letters}) + \text{Priority}(\text{buying invitation cards}) + \text{Priority}(\text{finding a place});$$

$$= \text{Priority}(\text{sending invitation letters}) + \text{Priority}(\text{writing invitation letters}) + \text{Priority}(\text{buying invitation cards}) + \text{Priority}(\text{finding a place});$$

Algorithm:

MinMetaPath($D_1, D_2$){
    $T' \leftarrow \Phi$;
    $D' = D_2 - D_1$; Let $D' = \{ d_1, d_2, ..., d_k \}$;
    For $i=1$ to $k$ do {
        $T[i] \leftarrow \Phi$;
        For each task $t$ in TASK{
            If $d_i \in t.o$ then $T[i] = T[i] \cup \{ t \}$;
        }/* for_TASK*/
    }/* for$*$
    if $T[i] = \Phi$ then return $-1$; /*Fail, $D_1$ cannot produce $D_2$*/
/* for i */
Minimum=∞;
For each task set $T'$ in $T[1] \times T[2] \times \ldots \times T[k]$ do {
    TaskCost=0;
    for each task $t$ in $T'$ do $TaskCost = TaskCost + Priority(t)$;
    PathCost = MinMetaPath($D_1$, Input($T'$))
    if $Minimum > PathCost + TaskCost$
        then $Minimum = PathCost + TaskCost$;
}
return Minimum;

Suppose there are $N_T$ elements in $T$ and $k$ values in ($D_2$ - $D_1$), then the for loop 
“each task set $T'$ in $T[1] \times T[2] \times \ldots \times T[k]$ ” takes $N_T^k$ times in worse case. We are 
unable give a precise close form for computing the running time of MinMetaPath 
because of unstable inputs ($D_2$-$D_1$) as recurrence. In the worst case, the entire running 
time is obviously exponential.

4.2. Unary operation

Unary operations include SELECT, COMBINED_INPUT, and COMBINED_OUTPUT.

4.2.1. SELECT Operation

As a basic relational algebra operator, SELECT operation in this context can be used 
for retrieving tasks (or data) whose attribute values satisfy a specified condition. 
SELECT operation is denoted as $σ$.

**DEFINITION:** Given a set $S$, $S$ is either a set of data or a set of tasks. SELECT 
operation returns the elements in $S$, each of which satisfies the selection condition.
Various approaches for implementing Selection operation in very large databases have been discussed in the literature [Elma00]. Due to the relative fewer data elements in the context of personal processes, we will just use sequential search to implement this operation. In this case, the running time of the operations $\sigma$ is $N$ as there are $N$ elements in $S$.

### 4.2.2. COMBINED_INPUT Operation

COMBINED_INPUT returns the minimal set of data items needed to execute a specific set of tasks. It is denoted as $\uparrow_i$.

**DEFINITION:** Given a set of tasks $T$, COMBINED_INPUT operation 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 $T$. Formally, $\uparrow_i (T) \equiv \{ \cup t.i \mid t \in T \} - \{ \cup t.o \mid t \in T \}$

**Algorithm:**

```
COMBINED_INPUT(T){
    S←\emptyset;
    For each task $t$ in $T$
        $S$←$S \cup t.i$;
    For each task $t$ in $T$
        $S$←$S - t.o$;
    Return S
}
```

Suppose there are $N_T$ elements in $T$, and each task in $T$ has $N_{in}(N_{out})$ elements in input (output) data attribute. The running time of the above algorithm for computing $\uparrow_i$ is $(N_T* N_{in} + N_T* N_{out})$. 

~35~
4.2.3. COMBINED_OUTPUT Operation

COMBINED_OUTPUT returns the set of data items produced as a result of executing a specific set of tasks. COMBINED_OUTPUT operation is denoted as $\uparrow_o$.

**DEFINITION:** Given a set of tasks $T$, COMBINED_OUTPUT operation returns a set of data, each of which is produced by some task in $T$. Formally, $\uparrow_o(T) \equiv \{ \cup t.o : t \in T \}$

**Algorithm:**

```
COMBINED_OUTPUT(T)
{
    S \leftarrow \emptyset;
    For each task $t$ in $T$
        $S \leftarrow S \cup t.o$;
    Return $S$
}
```

Suppose there are $N_T$ elements in $T$, and each task in $T$ has $N_{out}$ output data items.

The running time of the above algorithm for computing $\uparrow_o$ is $(N_T \cdot N_{out})$. 

~36~
Chapter 5. Transformation and Optimization of Query Expression

In the previous chapter, we have proposed several primitive operations that can be combined in various ways to express user’s queries. For a given request, there may exist more than one way to express it. Two query expressions are said to be equivalent if they are destined to generate the same result based on any kind of data set. Query optimization aims to identify an expression with the least cost among all equivalent expressions. In order to find such an optimal query expression, expression rules that generate equivalent expressions have to be consulted. An expression rule specifies how to transform an expression into a logically equivalent one. We enumerate the expression rules associated to our proposed operations in this chapter. Moreover, heuristics in applying these rules will also be discussed and demonstrated through examples.

5.1. Transformation of Query Expression

For notational purpose, we use $\theta$ or $\theta_i$ to denote a predicate, $D$ or $D_i$ to denote domain of data, and $T$ or $T_i$ to denote domain of tasks.

5.1.1. UNION, INTERSECTION, DIFFERENCE Operation

Set operations satisfy the properties of idempotency, commutativeity, and associativity. Equivalence rules of set operations ($\cup$, $\cap$, $\neg$) have been exhaustively studied in set theory [Corm90]. To make this theses self-contained, we lists these rules in the following:

1. Empty set laws:
\[ D \cup \phi = D \quad \ldots \ldots \quad (S1a) \]
\[ T \cup \phi = T \quad \ldots \ldots \quad (S1b) \]
\[ D \cap \phi = \phi \quad \ldots \ldots \quad (S1c) \]
\[ T \cap \phi = \phi \quad \ldots \ldots \quad (S1d) \]

2. Idempotency laws:
\[ D \cup D = D \quad \ldots \ldots \quad (S2a) \]
\[ T \cup T = T \quad \ldots \ldots \quad (S2b) \]
\[ D \cap D = D \quad \ldots \ldots \quad (S2c) \]
\[ T \cap T = T \quad \ldots \ldots \quad (S2d) \]

3. Commutative laws:
\[ D_1 \cup D_2 = D_2 \cup D_1 \quad \ldots \ldots \quad (S3a) \]
\[ T_1 \cup T_2 = T_2 \cup T_1 \quad \ldots \ldots \quad (S3b) \]
\[ D_1 \cap D_2 = D_2 \cap D_1 \quad \ldots \ldots \quad (S3c) \]
\[ T_1 \cap T_2 = T_2 \cap T_1 \quad \ldots \ldots \quad (S3d) \]

4. Associative laws:
\[ D_1 \cup (D_2 \cup D_3) = (D_1 \cup D_2) \cup D_3 \quad \ldots \ldots \quad (S4a) \]
\[ T_1 \cup (T_2 \cup T_3) = (T_1 \cup T_2) \cup T_3 \quad \ldots \ldots \quad (S4b) \]
\[ D_1 \cap (D_2 \cap D_3) = (D_1 \cap D_2) \cap D_3 \quad \ldots \ldots \quad (S4c) \]
\[ T_1 \cap (T_2 \cap T_3) = (T_1 \cap T_2) \cap T_3 \quad \ldots \ldots \quad (S4d) \]

5. Distributive laws:
\[ D_1 \cup (D_2 \cap D_3) = (D_1 \cup D_2) \cap (D_1 \cup D_3) \quad \ldots \quad (S5a) \]
\[ T_1 \cup (T_2 \cap T_3) = (T_1 \cup T_2) \cap (T_1 \cup T_3) \quad \ldots \quad (S5b) \]
\[ D_1 \cap (D_2 \cup D_3) = (D_1 \cap D_2) \cup (D_1 \cap D_3) \quad \ldots \quad (S5c) \]
\[ T_1 \cap (T_2 \cup T_3) = (T_1 \cap T_2) \cup (T_1 \cap T_3) \quad \ldots \quad (S5d) \]
6. Absorption laws:

\[ D_1 \cup (D_1 \cap D_2) = D_1 \] (S6a)

\[ T_1 \cup (T_1 \cap T_2) = T_1 \] (S6b)

\[ D_1 \cap (D_1 \cup D_2) = D_1 \] (S6c)

\[ T_1 \cap (T_1 \cup T_2) = T_1 \] (S6d)

7. DeMorgan’s laws:

\[ D_1 - (D_2 \cap D_3) = (D_1 - D_3) \cup (D_1 - D_2) \] (S7a)

\[ T_1 - (T_2 \cap T_3) = (T_1 - T_3) \cup (T_1 - T_2) \] (S7b)

\[ D_1 - (D_2 \cup D_3) = (D_1 - D_2) \cap (D_1 - D_3) \] (S7c)

\[ T_1 - (T_2 \cup T_3) = (T_1 - T_2) \cap (T_1 - T_3) \] (S7d)

5.1.2. PLACE_OVERLAP Operation

PLACE_OVERLAP (\( \circ_p \)) accepts two task sets and returns a task set. It obeys the following rules:

1. Distributive law with UNION operation

\[ (T_1 \cup T_2) \circ_p T_3 = (T_1 \circ_p T_3) \cup (T_2 \circ_p T_3) \] (PO1)

2. Distributive law with INTERSECTION operation

\[ (T_1 \cap T_2) \circ_p T_3 = (T_1 \circ_p T_3) \cap (T_2 \circ_p T_3) \] (PO2)

3. Distributive law with DIFFERENCE operation

\[ (T_1 - T_2) \circ_p T_3 = (T_1 \circ_p T_3) - (T_2 \circ_p T_3) \] (PO3)

4. Associative law with MAKE_EXECUTABLE operation

\[ (D \rightarrow T_1) \circ_p T_2 = D \rightarrow (T_1 \circ_p T_2) \] (PO4)
5. Commutative law with TIME_OVERLAP operation

\[(T_i \circ_i T_2) \circ_p T_3 = (T_i \circ_p T_3) \circ_i T_2 \ldots \ldots \ldots \text{(PO5)}\]

5.1.3. TIME_OVERLAP Operation

TIME_OVERLAP behaves as PLACE_OVERLAP except that it is time (rather than place) that determines whether two task sets overlap. Therefore, its expression rules are the same as that of PLACE_OVERLAP and are enumerated as follows:

1. Distributive law with UNION operation

\[(T_i \cup T_2) \circ_i T_3 = (T_i \circ_i T_3) \cup (T_2 \circ_i T_3) \ldots \ldots \ldots \text{(TO1)}\]

2. Distributive law with INTERSECTION operation

\[(T_i \cap T_2) \circ_i T_3 = (T_i \circ_i T_3) \cap (T_2 \circ_i T_3) \ldots \ldots \ldots \text{(TO2)}\]

3. Distributive law with DIFFERENCE operation

\[(T_i - T_2) \circ_i T_3 = (T_i \circ_i T_3) - (T_2 \circ_i T_3) \ldots \ldots \ldots \text{(TO3)}\]

4. Associative law with MAKE_EXECUTABLE operation

\[\left( D \rightarrow_{dt} T_i \right) \circ_i T_2 = D \rightarrow_{dt} (T_i \circ_i T_2) \ldots \ldots \ldots \text{(TO4)}\]

5. Commutative law with PLACE_OVERLAP operation

\[(T_i \circ_p T_2) \circ_i T_3 = (T_i \circ_i T_3) \circ_p T_2 \ldots \ldots \ldots \text{(TO5)}\]

5.1.4. MAKE_EXECUTABLE Operation

MAKE_EXECUTABLE takes a data set and a task set as input and returns a task set. It obeys the following expression rules:

1. Distributive law with UNION operation

\[D \rightarrow_{dt} (T_i \cup T_2) = D \rightarrow_{dt} T_i \cup (D \rightarrow_{dt} T_2) \ldots \ldots \ldots \text{(ME1)}\]
2. Distributive law with INTERSECTION operation

\[ D \rightarrow (T_1 \cap T_2) = (D \rightarrow T_1) \cap (D \rightarrow T_2) \]  \( \ldots \ldots \) (ME2)

3. Distributive law with DIFFERENCE operation

\[ D \rightarrow (T_1 - T_2) = (D \rightarrow T_1) - (D \rightarrow T_2) \]  \( \ldots \ldots \) (ME3)

4. Associative law with PLACE_OVERLAP operation

\[ D \rightarrow (T_1 \circ_p T_2) = (D \rightarrow T_1) \circ_p T_2 \]  \( \ldots \ldots \) (ME4)

5. Associative law with TIME_OVERLAP operation

\[ D \rightarrow (T_1 \circ_o T_2) = (D \rightarrow T_1) \circ_o T_2 \]  \( \ldots \ldots \) (ME5)

5.1.5. NEED_TASK Operation

NEED_TASK generates a meta path of least cost that connects two data sets. Due to its algorithmic complication, expression rules does not exist or difficult to identify.

5.1.6. SELECT Operation

SELECT operation behaves the same as its counterpart (\( \sigma \)) in relational algebra. It can be moved across many types of operations in the query tree. In fact, these rules have been widely used in today’s DBMSs for finding optimal query plans. More precisely, SELECT operation follows the following expression rules:

1. Distributive law with UNION operation

\[ \sigma_\theta(D_1 \cup D_2) = (\sigma_\theta(D_1)) \cup (\sigma_\theta(D_2)) \]  \( \ldots \ldots \) (S1a)

\[ \sigma_\theta(T_1 \cup T_2) = (\sigma_\theta(T_1)) \cup (\sigma_\theta(T_2)) \]  \( \ldots \ldots \) (S1b)

2. Distributive law with INTERSECTION operation

\[ \sigma_\theta(D_1 \cap D_2) = (\sigma_\theta(D_1)) \cap (\sigma_\theta(D_2)) \]  \( \ldots \ldots \) (S2a)
\[ \sigma_{\phi_{1}}(T_{1} \cap T_{2}) = (\sigma_{\phi_{1}}(T_{1})) \cap (\sigma_{\phi_{1}}(T_{2})) \] . . . . . . (S2b)

3. Distributive law with DIFFERENCE operation

\[ \sigma_{\phi}(D_{1} - D_{2}) = (\sigma_{\phi}(D_{1})) - (\sigma_{\phi}(D_{2})) \] . . . . . . (S3a)

\[ \sigma_{\phi}(T_{1} - T_{2}) = (\sigma_{\phi}(T_{1})) - (\sigma_{\phi}(T_{2})) \] . . . . . . (S3b)

4. Associative law with PLACE_OVERLAP operation

\[ \sigma_{\phi}(T_{1} \circ p T_{2}) = \sigma_{\phi}(T_{1}) \circ p T_{2} \] . . . . . . (S4)

5. Associative law with TIME_OVERLAP operation

\[ \sigma_{\phi}(T_{1} \circ \gamma T_{2}) = \sigma_{\phi}(T_{1}) \circ \gamma T_{2} \] . . . . . . (S5)

6. Distributive law with MAKE_EXECUTABLE operation

\[ \sigma_{\phi}(D_{\rightarrow}T) = D_{\rightarrow}(\sigma_{\phi}(T_{1})) \] . . . . . . (S6)

5.1.7. COMBINED_INPUT Operation:

COMBINED_INPUT takes a task set as input and produces a data set. It obeys the following expression rules:

1. Distributive law with UNION operation

\[ \hat{\tau}_{1}(T_{1} \cup T_{2}) = (\hat{\tau}_{1}(T_{1})) \cup (\hat{\tau}_{1}(T_{2})) - (\hat{\tau}_{0}(T_{1})) \cup (\hat{\tau}_{0}(T_{2})) \] . (CI1)

2. Distributive law with INTERSECTION operation

\[ \hat{\tau}_{1}(T_{1} \cap T_{2}) = ((\hat{\tau}_{1}(T_{1})) \cap (\hat{\tau}_{1}(T_{2})) - (\hat{\tau}_{0}(T_{1})) \cap (\hat{\tau}_{0}(T_{2})) \] . (CI2)

3. Distributive law with DIFFERENCE operation

\[ \hat{\tau}_{1}(T_{1} - T_{2}) = ((\hat{\tau}_{1}(T_{1})) - (\hat{\tau}_{1}(T_{2})) - (\hat{\tau}_{0}(T_{1})) - (\hat{\tau}_{0}(T_{2})) \] . (CI3)
5.1.8. COMBINED_OUTPUT Operation:

COMBINED_OUTPUT takes a task set as input and produces a data set. It obeys the following expression rules:

1. Distributive law with UNION operation
   \[ \uparrow_o (T_1 \cup T_2) = (\uparrow_o (T_1)) \cup (\uparrow_o (T_2)) \ldots \ldots \ldots \text{(CO1)} \]

2. Distributive law with INTERSECTION operation
   \[ \uparrow_o (T_1 \cap T_2) = (\uparrow_o (T_1)) \cap (\uparrow_o (T_2)) \ldots \ldots \ldots \text{(CO2)} \]

3. Distributive law with DIFFERENCE operation
   \[ \uparrow_o (T_1 - T_2) = (\uparrow_o (T_1)) - (\uparrow_o (T_2)) \ldots \ldots \ldots \text{(CO3)} \]

5.2. Query Optimization

By applying the set of expression rules discussed above, we can transform an initial query expression to many other equivalent ones. In this thesis, we adopt a simple heuristic that applies the more restrictive and low cost operations as early as possible in the execution query plan. We say an operation is more restrictive if the size of its output is smaller than that of its operands. Among all the proposed operations, SELECT(\(\sigma\)), TIME_OVERLAP(\(\circ_t\)), PLACE_OVERLAP(\(\circ_p\)), and MAKE_EXECUTABLE(\(\rightarrow\)) are considered restrictive as they all return a subset of one input operand. Specifically, we define selectivity of an operation as the ratio of the size of its output to the size of its operand of the same type. Note that \(\uparrow_i\), \(\uparrow_o\), and \(\rightarrow\) do not have selectivity because the types of their output are different from
those of their input. In terms of running time of operations, SELECT (σ) is the cheapest and NEED_TASK (→) is the highest due to its intrinsic recursion structure.

Table 3 summarizes the running time and selectivity of each operation:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Running time</th>
<th>Selectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>σ</td>
<td>(N)</td>
<td>&lt;=1</td>
</tr>
<tr>
<td>∩</td>
<td>(N₁<em>₁</em>N₂)</td>
<td>&lt;=1</td>
</tr>
<tr>
<td>−</td>
<td>(N₁<em>₁</em>N₂)</td>
<td>&lt;=1</td>
</tr>
<tr>
<td>∪</td>
<td>(N₁<em>₁</em>N₂)</td>
<td>&gt;1</td>
</tr>
<tr>
<td>↑₁</td>
<td>(N₁<em>N₂</em>N₁*N₂)</td>
<td>N/A</td>
</tr>
<tr>
<td>↑₀</td>
<td>(N₁*N₂)</td>
<td>N/A</td>
</tr>
<tr>
<td>→₁</td>
<td>(N₁<em>N₂</em>N₂<em>N₁</em>₂)</td>
<td>&lt;=1</td>
</tr>
<tr>
<td>→₂</td>
<td>(N₁<em>N₂</em>N₂<em>N₁</em>₂)</td>
<td>&lt;=1</td>
</tr>
<tr>
<td>→₃</td>
<td>(N₁<em>N₂</em>N₂<em>N₁</em>₂)</td>
<td>&lt;=1</td>
</tr>
<tr>
<td>→₄</td>
<td>exponential</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Based on the heuristics that operations with low running time and small selectivity should be applied first, we come up with the following steps:

1. Move σ down in the query tree.
2. Move ∩ and − down in the query tree.
3. If the first operation of →₁ (of type D) is of small size, move it down in the query tree.
In the following, we give examples that apply these optimization steps to optimize queries. We refer to the party holding process shown in Figure 2, and assume that three tasks ‘buying decoration stuff’, ‘buying food’, ‘buying invitation cards’ can be executed at the same place and time.

**Example 1:** Suppose a set of tasks, ‘planning’, 'finding a place', ‘buying invitation cards’, ‘buying decoration stuff’ and ‘buying food’ in the party-holding process, have all been executed, consider the following query:

*Given the current available data set, which tasks are executable?*

The initial query expression could be

\[ \sigma_{s = \text{unexecuted}}((\sigma_{s = \text{available}} \rightarrow T)) \]

By applying step 1, we get the following equivalent query expression:

\[ (\sigma_{s = \text{available}} \rightarrow \sigma_{s = \text{unexecuted}} T) \]

On our prototyped system running on Palm Pilot OS, the execution time of the above optimized query expression saves 0.88 seconds (from 2.08s to 1.2s), due to the decrease of selectivity of \( \sigma \) (from 1/2 in the initial query expression to 2/7 in the optimized query expression).

**Example 2:** Suppose ‘planning’ and ‘finding a place’ have both been executed. Consider the following query:

*As I am approaching a supermarket, I would like to find the tasks that can be executed at supermarket at 5pm.*

An initial query can be expressed as:

\[ (\sigma_{s = \text{available}} \rightarrow \sigma_{p = \text{supermarket}} T) \cap (\sigma_{s = \text{available}} \rightarrow \sigma_{t = 5pm} T) \]

By applying step 2, we obtain the following equivalent query expression:
\[ \sigma_{\text{availuable}} D \rightarrow (\sigma_{p \geq \text{supermarket}} T \cap \sigma_{t \geq \text{5pm}} T) \]

The revised query saves 0.79 seconds (from 3.64s to 2.85s) in execution time and only executes \( \rightarrow \) operation once.

**Example 3**: Suppose the party hold process has just been specified, and no tasks have been executed. Consider the following query:

I would like to know the set of tasks that are executable at current place and the current time

The initial query expression could be formulate as follows:

\[ \sigma_{s=\text{available}} D \rightarrow(\sigma_{s=\text{unexecuted}} T \circ p \ \text{CURRENT} \circ t \ \text{CURRENT}) \]

In this case, as the first operand of \( \rightarrow \) is of small size, by applying step 3 we get the revised query as follows:

\[ (\sigma_{s=\text{available}} D \rightarrow\sigma_{s=\text{unexecuted}} T) \circ p \ \text{CURRENT} \circ t \ \text{CURRENT} \]

The revised query save 0.67 seconds in processing time (from 3.45s to 2.78s). This is because the selectivity of \( \rightarrow \) operation in the revise query expression is 1/7, while it is 1/1 in the initial query expression.

The performance of the three queries with and without query optimization is shown in Figure 6.
Figure 6. Performance of query optimization
Chapter 6. SQL-Like query language and Rule model

To ease the job of specifying a query on personal processes, in this chapter we propose a SQL-like query language that enable users to describe queries in a declarative way. As a personal workflow system is designed to run at a PMC, actively reminding the mobiler the right things to do at the right place and the right time is essential. To serve this purpose, we propose a rule model that resembles event-condition-action (ECA) rules proposed for active database systems.

6.1. SQL-Like query language

We design a language construct SELECT-FROM-GIVEN that is similar to the basic SELECT-FROM-WHERE blocks of SQL. The operation to be performed and its output attributes are specified in ‘select’ clause. The process(es) to which this operation is applied is declared in ‘from’ clause. The ‘given’ clause describes the operands as well as the constraints on data, tasks, time, and place. The declarative language syntax is defined as Backus Naur Form (BCN) in Table 4.

Table 4. The Declarative Language

| SELECT [<operation>] (<task attribute>| <data attribute>)* |
| FROM  (<process>)+ |
| GIVEN  (<task attribute>| <data attribute>) <condition> <value> |
| {<selection condition> (<task attribute>| <data attribute>) <condition> <value>} |

<operation>::=(MAKEEXECUTABLE | NEEDTASK | COMBINEDINPUT | COMBINEDOUTPUT)
We use notation in [Elma00], where non-terminal symbols are shown in angled brackets <…>, optional parts are shown in square brackets […], repetitions are shown in braces {...}, and alternatives are shown in parentheses (...|...|...). The details about each clause are described below:

1. **SELECT clause**: In the declarative language, <operation> specifies one of the proposed operations, namely \( \rightarrow_{dt} \), \( \rightarrow_{dd} \), \( \uparrow_i \), or \( \uparrow_o \). <task attribute > and <data attribute > indicate the operands associated with <operation>.

2. **FROM clause**: An angled bracket <process> specifies the processes this entire query is to be applied to.

3. **GIVEN clause**: It specifies the first operand and, optionally, the second operand (by using WITH quantifier). Other constraints are specified by quantifiers AFTERTASK, TIMEOVERLAP and PLACEOVERLAP. AFTERTASK specifies a future state based on which the operation in SELECT clause is executed. TIMEOVERLAP and PLACEOVERLAP subclauses constrain the output task set on time and place dimensions respectively. Each subclause in ‘GIVEN’ clause is composed of <task/data attribute> <condition> and <value> parts to filter the task (data) set of interest.

For example, “To find a set of tasks that can be co-executed with ‘buying invitation cards’, when ‘planning’ and ‘finding a place’ are both completed”, the following algebraic expression serves this purpose:
Alternatively, we can specify the same expression by our proposed language. It describes as shown bellow:

SELECT MAKEEXECUTABLE  TASK.Name
FROM  party-planning-process
GIVEN  DATA.Status IS AVAILABLE
WITH  TASK.Status IS UNEXECUTED
AFTERTASK  Name IN ‘planning, finding a place’
TIMEOVERLAP  Name IS ‘buying invitation cards’
PLACEOVERLAP  Name IS ‘buying invitation cards’

Such a high-level query language will in turn be used for designing a user-friendly interface for users to express their queries.

6.2. Rule model

ECA rules are first proposed in active database systems to fire appropriate rules when events of interest occur and conditions are satisfied. Event of an ECA rule includes structure operation (such as insert, delete, update), clock (such as absolute, relative, periodic), transaction (such as commit, abort, prepare-to-commit), user-defined, behavior invocation, exception, and other external event [Pato99]. Trigger mechanism in our PWFS is similar to ECA rules except that event types are focused on operations, temporal events and locational events. Besides, our proposed queries can be specified in both event and action parts. We illustrate each event type in the following by
showing an example:

1. LOCATION-CHANGE: Quite a number of locating technologies have been developed and deployed in commercial products to trace mobilers’ locations. Two well known technologies are GPS and cellular systems such as GSM. Depending on the adopted technology, ‘location change’ event can be generated. For example, in a cellular system, this event occurs when a mobiler moves into a new cell (with handoff operation). In a GPS based system, such an event happens when a mobiler make a move that is far more than a specific distance (e.g. 100m) from its previous location. The following rule monitors the location change event, and if any task is found to be executable (with input data being available and time/place being suitable), inform the mobiler about this task.

```sql
DEFINE RULE location-change
ON LOCATION-CHANGE
IF

EXISTS ( 
SELECT MAKEEXECUTABLE ( TASK.Name AS T ) 
FROM processname 
GIVEN DATA.Status IS AVAILABLE 
WITH TASK.Status IS UNEXECUTED 
TIMEOVERLAP CURRENT 
PLACEOVERLAP CURRENT 
)

THEN

DISPLAY T
```

Note that CURRENT is a dummy task with time and place attributes being the current time and current place respectively.

2. Temporal: Temporal events could be absolute (e.g., 08:00:00 hours on 1 July
2001), relative (e.g., 10 minutes after a specific task is executed), or periodic (e.g., 08:00:00 hours every day, the first day of every week/month). Rules of this type are primarily used to the tasks that could be immediately executed as specific time approaches. The following three examples demonstrate the power of these rules. We copy the syntax proposed in [Cast99] to specify temporal events.

**Absolute time event:**

**DEFINE RULE** temporal-absolute

**ON** 

**<specific time>**

**IF**

**EXISTS**(

**SELECT** MAKEEXECUTABLE (TASK.Name AS T)

**FROM** processname

**GIVEN** DATA.Status IS AVAILABLE

**WITH** TASK.Status IS UNEXECUTED

**TIMEOVERLAP** CURRENT

**PLACEOVERLAP** CURRENT

)

**THEN**

**DISPLAY** T

**Relative time event:**

**DEFINE RULE** temporal-relative

**ON**

(ELAPSED(<delay>) SINCE (TASKEND(<specific task>)) )  |  (ELAPSED(<delay>) SINCE  <specific time> )

**IF**

**EXISTS**(

**SELECT** MAKEEXECUTABLE (TASK.Name AS T)

**FROM** processname

**GIVEN** DATA.Status IS AVAILABLE
WITH TASK.Status IS UNEXECUTED
TIMEOVERLAP Name IS CURRENT
PLACEOVERLAP Name IS CURRENT

THEN

DISPLAY T

Periodical time event:

DEFINE RULE temporal-period
ON (EVERY(<delay>) SINCE TASKEND(<specific task>)) |
(EVERY(<delay>) SINCE <specific time>)

IF

EXISTS(

SELECT MAKEEXECUTABLE (TASK.Name AS T)
FROM processname
GIVEN DATA.Status IS AVAILABLE
WITH TASK.Status IS UNEXECUTED
TIMEOVERLAP Name IS CURRENT
PLACEOVERLAP Name IS CURRENT
)

THEN

DISPLAY T

<specific time> in temporal-absolute rule is expressed as an absolute time such as “2001-06-01 08:00:00”. In temporal-relative rule, event can be specified by two ways: TASKEND and specific time. For example, “ELAPSED (00:10:00) SINCE TASKEND (planning)” means that event will occur after 10 minutes since ‘planning’ task have been executed. “ELAPSED (00:10:00) SINCE (2001-06-01 08:00:00)” means that event will occur 10 minutes after 2001-06-01 08:00:00. Finally, the term ELAPSED can be replaced by EVERY for representing periodical events in temporal-period rules.

3. TASK-END: This event occurs when a specified task completes. For example,
the following rule will remind the mobiler the next tasks to do when ‘planning’ is completed:

DEFINE RULE after-taskend
ON TASK-END(planning)
IF

EXISTS(

SELECT MAKEEXECUTABLE (TASK.Name AS T )
FROM party-planning-process
GIVEN DATA.Status IS AVAILABLE
WITH TASK.Status IS UNEXECUTED
)
THEN
DISPLAY T

4. DATA-AVAILABLE: This event occurs when a specific data item becomes available. For example, the following rule will remind the mobiler the next tasks to do when ‘place’ becomes available.

DEFINE RULE after-dataavailable
ON DATA-AVAILABLE(place)
IF

EXISTS(

SELECT MAKEEXECUTABLE (TASK.Name AS T )
FROM party-planning-process
GIVEN DATA.Status IS AVAILABLE
WITH TASK.Status IS UNEXECUTED
)
THEN
DISPLAY T
Chapter 7. Implementation Status

To prove our concept, we have implemented a prototype PWFS that includes the process definition and query expression/processing capabilities on Palm Pilot PDA. We choose J2ME-CLDC (Java 2 Micro Edition, Connected, Limited Device Configuration) as the development tool. J2ME-CLDC was released by Sun corporations and aimed at providing portable Java platform to small, resource constrained, connected devices.

7.1 Process definition

This component enables the mobiler to define a personal process that constitutes tasks, data, and attributes associated to tasks and data. Figure 7 displays the screenshot for defining a task. A task is defined by filling values for task name, task input data, task output data, place and time information, and an execution priority (required by NEED_TASK operation). Figure 8 displays a screenshot for entering input data set. The right lower box in Figure 8 lists the data items defined previously. The mobiler can insert a new data item on the top edition box and selectively choose a subset of data items listed in the right lower box.
7.2 Query processing

A SQL-like query language serves as the basis for designing a user-friendly interface to express queries. Figure 9,10,11,12,13,14 show several screenshots for specifying various clauses of the following example query.

```sql
SELECT MAKEEXECUTABLE TASK.Name
FROM party-planning-process
GIVEN DATA.Status IS AVAILABLE
WITH TASK.Status IS UNEXECUTED
AFTER TASK Name IN 'planning, finding a place'
TIMEOVERLAP Name IS 'buying invitation cards'
PLACEOVERLAP Name IS 'buying invitation cards'
```
Figure 9. Specifying <operations> in SELECT clause

Figure 10. Specifying (<task attribute> | <data attribute>) in SELECT clause

Figure 11. Specifying <selection condition> in GIVEN clause

Figure 12. Specifying (<task attribute> | <data attribute>) in GIVEN clause
In fact, an even higher-level query interface can be designed by making use of the SQL-like declarative query language we proposed in Chapter 6. In this case, the processing of the declarative query language is viewed as a middleware by the higher level application designers. Alternatively, to guide the mobiler is specifying his/her queries, we have designed another interactive interface in the application layer. It is composed of three sequential steps. First, the desired process is selected. Then the function of interest is selected from a list-box shown Figure 15. Finally the required operands are chosen (Figure 16). Note that a screen like Figure 16 may be displayed once or twice depending on the type of the function selected in the second step (unary or binary).
Implementation issue on active mechanism is rule tracing. The capability of rule tracing includes event detection, condition evaluation and action execution. One way
of controlling executions of rules is to integrate with some public productive inference tool. However, embedded the inference engine in our PWFS will probably have problems in smaller devices, e.g. limited storage and dissimilar executable OS platform, and integrated with event detection unit which delivers message from dependency event (temporal, task, data, and location change). Due to the lack of location tracing capability in our current prototype, we have not implemented the rule model we proposed in Chapter 6. We view the implementation of rule model as part of our future work.
Chapter 8. Conclusion

As wireless technologies are getting their maturity, supporting personal services on mobile devices will be a trend. In this thesis, we have proposed an architecture for a personal workflow system in support of pervasive computing. We introduced the model of the PWFS and operations for expressing user’s inquiries. We further discussed the issue of transforming query expressions and implemented some of these operations. The goal of this work is to investigate issues related to the design of the process storage and query processing components of a PWFS. It is the first step toward a full-fledged, client-server based, personal workflow system. Our future work includes the design and implementation of the server-side components in a PWFS.
Reference


~63~


