# An Incremental Approach to Modeling Flexible Workflows Using Activity Decomposition and Gradual Refinement

Yanrong Jiang, Weihua Li, Jingtao Yang
School of Computer, Guangdong University of Technology, Guangzhou, China
Email: yrjiang@gdut.edu.cn

***Abstract*—For existing workflow models, it is still difficult to take into account both the hierarchical relations among activities and their execution orders at the same time, as well as the dynamic uncertainties of workflow processes. Aiming at these problems, an approach to modeling flexible workflow using activity decomposition and incremental refinement is presented. First, the activities as well as their decomposition relationship and the decomposition rules are analyzed to establish an activity decomposition model, where flexible activities, with verifying constraints, are used to package uncertainties. Then temporal relations are introduced into the model to ensure the proper execution orders of activities, which are tightly integrated with the hierarchical relationship as well as different granular activities to make the model perform well. The execution mechanism of the model, such as the delivering of events and states constraints of parent-child activities, is discussed in detail, and a common algorithm for activity execution is also presented. Finally, the proposed approach is applied to the PBL learning system, and the results indicate the effectiveness of the proposal.**

***Index Terms*—flexible workflow, activity decomposition, temporal activity tree, incremental refinement**

## I. Introduction

With the development of economic globalization and increasing of market competition, the business processes should adapt to the varying of business goals and requirements. The business process is always a dynamic, incomplete process having a lot of uncertainties and ambiguities [1]. While most of the traditional workflow models are often used to handle predictable, completely specified workflow process. So it is very difficult to use them to model complex business processes. A business process consists of activities, participants and resources, and some of the activities can be specified clearly, while others are always ambiguous and can not be predicated. The ambiguity of activity content and the uncertainty of activity relations make the modeling and execution of workflow very difficult. Aiming at this, improving the workflow flexibility to deal with such ambiguities and uncertainties of workflow has become one of the hot research fields recently [2, 3].

The modeling of complex business processes is normally not a one-off occurrence and may experience several rounds. It is always an incremental multi-granularity design process from coarse state to fine state, from rough frame design to specific details according to the requirements. In this process, the uncertainties of the model are gradually transformed into clearly specified elements, and obviously, it is an incremental refinement process of multiple rounds. Thus, an approach to support stepwise modeling process, which may contain multi-granularity activities as well as proper execution mechanism is of great importance.

By doing so, some research efforts were made. S. Li [4] adopted DNG (directed network graph) to model the production design process. By mapping from production tree to process templates, the runtime refinement of dynamic workflow nodes was realized. But the establishment of production tree, which often contains a large number of components and parts as well as their affiliations, and the corresponding complete process template database are time-consuming and difficult. So it is not suitable for processes with lots of uncertainties and ambiguities. S. A. Chun [5] proposed an approach to automatically generate workflow schema based on ontology. It uses domain service ontology and domain integration knowledge that serves as a model for workflow composition rules, and uses user profile to select a suitable workflow process. Its disadvantage is that the composition rules can not be moved to other domains easily, and the establishment of ontology is difficult. S. Nurcan [6] proposed a conceptual framework for flexible workflow modeling based on intention-driven methods. It offers the capacity to represent a well-structured process chunks or an ill-structured one in the process definition. A. Luntovskyy [7] tried to make a trade-off between adequate flexibility and consideration of the specifics of project, and discussed a flexible workflow management system for CAD applications. In the work of Y. Zhang [8], an activity-center modeling approach was used for software development process. The decomposition of activity was analyzed, and ECA rules were used to form a dynamic model of software process.

Manuscript received February 13, 2014; revised March 25, 2014; accepted April 1,2014.
Corresponding author: Yanrong Jiang.


doi:10.4304/jsw.9.10.2628-2637

Although the researches mentioned above can improve the flexibilities of workflow in a certain degree, there still exist some lacks. Some of the work only emphasizes the transverse connections of activities, and irrationally ignores the hierarchical relations of activities. So the support for modeling with multiple abstraction levels and multiple granularities is not sufficient. Some of the work establishes roughly the hierarchical decomposition model of workflow, but the temporal relations of activities as well as the execution constraints are not studied in depth, while this is crucial to the proper execution and the adaption of workflow model. Some other work focuses on detailed technologies, and the integration of them to improve the flexibility of workflow is still lacking.

Hence, in this paper, we present an incremental approach to modeling flexible workflows using activity decomposition and gradual refinement. There are several advantages of this approach. First, different levels of abstraction and vertical decomposition relations of activities are taken into account, which will benefit the gradually incremental modeling of workflow process. Second, the introducing of temporal relationship of activities will help to control their execution orders, and its graphical expression is easy to visualize the meaning of activity relationship. Finally, flexible activity is used to package the uncertainties of workflow process, which can reduce modeling difficulties.

## II. Activity Decomposition Model

Activities are the basic units of workflow and are not isolated. There exist various relations among activities, such as the decomposition relation and the temporal relation, and their complexity reflects the difficulties of workflow modeling. Therefore, the main task of workflow modeling is to describe activities, perhaps with multiple granularity, and the relations among them clearly and directly.

During the process of activity decomposition, vertical relationship among activities should be established to support top-down project design. By examining the relations between activity and its sub-activities, such relationship can be grouped into 3 categories: *aggregation*, *generalization*, and *attribution* relations, where the attribution relation is used to describe the properties of activities.

**Definition 1 (aggregation relation)** the aggregation relation can be denoted as $a(A_i, A_j)$, which is used to describe the "whole-part" relation (i.e., $A_i$ is a part of $A_j$). When an activity is decomposed into several sub-activities, it is referred to as a composite activity. Actualy, when the decomposed activity is also a composite activity, it can be decomposed further until all the activities decomposed are atom activities.

**Definition 2 (generalization relation)** this relation is another case, which can be used to define the general-special relation. It can be denoted as $g(A_i, A_j)$, which means that $A_i$ is a special case of $A_j$. For example, the activities such as object-oriented design, structured design and process-oriented design are the special cases of software design activity.

**Definition 3 (parent-child relation)** for any two activities $x$ and $y$, if $a(x, y)$ or $g(x,y)$ is satisfied, then $y$ and $x$ is called as the parent-child relation, which is denoted as $s(x, y)$. This relation is transitive. For example, for activities $A_1$, $A_2$ and $A_3$, if $s(A_1, A_2)$ and $s(A_2, A_3)$ are true, then $s(A_1, A_3)$ is true. To distinguish this relation from ancestor-descendant relation, the parent-child relation can be noted further as $sc(x, y)$.

**Definition 4 (Decompose operator)** $Decompose(y)$ is a unary operator, and its result is a collection of all the sub-activities of $y$. If $y$ has no sub-activity, then an empty set is returned. Recursively calling this operator, we can decompose an activity gradually. Obviously, $\forall x$, $sc(x, y) \rightarrow x \in Decompose(y)$ is true.

**Rule 1.** the achievement of goals and functions of the parent activity relies on the achievement of goals and functions of all its sub-activities. The resources and personnel of parent activity is the sum of those of all its sub-activities.

Rule 1 shows that during the process of activity decomposition, the parent activity with high level of abstraction and coarse granularity can be gradually decomposed into fine-grained sub-activity, and the function and goal of parent activity is also distributed to its sub-activities, which provides a basis for reasonable decomposition and the related decomposition algorithm. Rule 1 indicates that the decomposition standard is the independent function, rather than the structure, the advantage of which is that it can reduce the coupling between activities and improve the reusability of activities.

**Definition 5 (Activity tree)** decomposing an activity $A$ gradually will eventually generate a tree with a root of $A$, which is called an activity tree, denoted as $T(A)$. In this tree, the child node is the sub-activity of the parent node. When all the composite activities are decomposed into atom activities, the decomposition process stops. For example, Fig.1 is an activity tree of software design process, which contains aggregation and generalization relation.

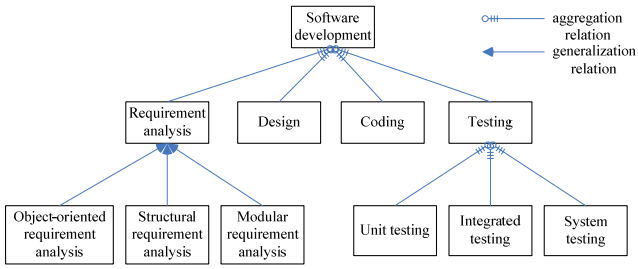


Figure 1. Activity tree for software development

The essence of the process to establish an activity tree is to gradually refine and determine the uncertain process model. This is a process from abstract to concrete, from coarse granular to fine granular, and from vague to clear process. It demands the gradual obtaining of domain knowledge and the in-depth analysis of the process to determine and refine the uncertain activities.

The activities of workflow can be classified into 3 categories: atom activity, composite activity and flexible activity. If a clearly specified activity can not be divided

into other sub-activities, it is referred to as an atom activity. Atom activity is the smallest execution unit. Composite activity is composed of several sub-activities with their relationship, and it can be considered as a sub-process. Every sub-activity of it can be an atom activity or a composite activity. Flexible activity is used to describe and package the uncertain factor of a workflow process.

**Definition 6 (Flexible activity)** flexible activity is a special activity. Due to the vague and uncertain information existing in processes, flexible activity can not be specified clearly and fully in advance. So further information is required later for the gradual determination of flexible activities. Flexible activity can be described by: $FA$=(*Id, name, state, Ctx, FAP, FAR, FAC*, *Attr, dtype*), where *Id* is the unique identify of the activity, $state \in STATE$, *Ctx* is the context, including its goal and functions to achieve, and the application scenarios. $FAP = A \cup CAS \cup FA$ is the activity pool, where $A$ is an atom activity set, *CAS* is a composite activity set, and *FAS* is a flexible activity set. $FAR = FR \cup CR$ is the rule set to describe the activity relations in the activity pool, where *FR* is the flexible rule set and *CR* is the common rule set expressed by ECA rule. *Attr* is the attribution set that the activity has. $dtype \in$ {*aggregation*, *generalization*, *has_attribution*} represents the decomposition type.

$FAC = C \cup M$ is the constraint set for the verification and determination of the flexible activity, and the generated sub-processes must follow the constraints. Unary or binary predicates, preset operations and rules are used to express the constraints. Herein, $C$ is the constraint rules used for the selection and combination of activities, determining the execution order of activities. $M$ is the adjustment rule set, used to modify the generated sub-process. For example, the rule "IF *select*($a \wedge b$) THEN *Previous*($a$, $b$)" means that if activities $a$ and $b$ are selected, $a$ must be executed before $b$.

Activities have various states, and the state set can be described by: $STATE$={"Waiting", "Ready", "Executing", "Committed", "Aborted"}, where state Waiting(W) is the initial state of activities, which means that the triggering conditions of the activity are not satisfied, and the execution is not ready; Ready(R) represents that the activity's triggering condition has been met and the activity will be executing; Executing(E) represents that the activity is running but not completed; Committed(C) represents that the activity is completed and Aborted(A) means that some exceptions occur and the execution will quit. Fig.2 illustrates the state transition of activities.

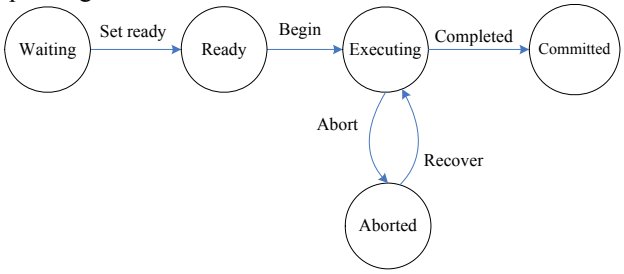


Figure 2. Transition of activity states

## III. Introducing Temporal Relations into the Activity Decomposition Model

### A. Temporal Activity Tree (TAT)

Though the activity tree can clearly express the vertical decomposition relations between activities, the horizontal relations, however, such as the temporal relations, are not expressed in the tree. While this is of great importance of the process model. Next, the definition of temporal relation will be given first, then the temporal activity tree will be established.

**Definition 7 (Temporal relation)** activities are often performed in a certain order in time, forming a sequence of activity execution. That is, the start of an activity must follow after the completion of another (or more) activity. Such constraint of timing relation is referred to as the temporal relation, denoted as $\prec = \{<A_i, A_j> | A_i \prec A_j \wedge A_i \in A \wedge A_j \in A\}$, where $A$ is the activity set of the workflow.

**Definition 8 (Predecessor-successor relation)** for activity $A_i$ and $A_j$, if the completion of $A_i$ is a prerequisite to the start of activity $A_j$, $A_i$ is referred to as the predecessor of $A_j$, and $A_j$ is the successor of $A_i$, which is denoted as $A_i \prec A_j$. All the predecessors of $A_i$ form a predecessor set, $\mathrm{Pre}(A_i)=\{a | a \prec A_i\}$, and all the successors of $A_i$ form a successor set, $\mathrm{Succ}(A_i)=\{a | A_i \prec a\}$. If there does not exist an activity $A_k$, which satisfies $A_i \prec A_K$ and $A_K \prec A_j$, then $A_i$ is called a direct predecessor of $A_j$, and $A_j$ is the direct successor of $A_i$.

**Definition 9 (SERIAL relation)** if the activity $A_i$ and $A_j$ satisfy predecessor-successor relation, and there does not exist an activity $A_k$, which satisfies $A_i \prec A_K$ but not $A_K \prec A_j$, then $A_i$ and $A_j$ are said to satisfy SERIAL relation.

**Definition 10 (AND relation)** if there exists a nearest common predecessor $A_s$ and a nearest common successor $A_e$ for activity $A_i$ and $A_j$, and satisfy $A_s \prec A_i$, $A_s \prec A_j$, $A_i \prec A_e$, $A_j \prec A_e$, and there exists no temporal relation between $A_i$ and $A_j$, and $A_i$ and $A_j$ execute in parallel, then $A_i$ and $A_j$ is referred to as the AND relation between $A_s$ and $A_e$.

**Definition 11 (OR relation)** if there exists a nearest common predecessor $A_s$ and a nearest common successor $A_e$ for activity $A_i$ and $A_j$, and satisfy $A_s \prec A_i$, $A_s \prec A_j$, $A_i \prec A_e$, $A_j \prec A_e$, and there exists no temporal relation between $A_i$ and $A_j$, and there is only one activity of they can be performed, then $A_i$ and $A_j$ is referred to as the OR relation between $A_s$ and $A_e$.

**Definition 12 (Iteration relation)** for activity set $A=(A_1, A_2, ..., A_k)$, if the execution sequence consisting of all the activities is complete and repeated, and satisfies $A_1 \prec A_2 \prec ... \prec A_k$, there exists an interation relation $<A_1, A_2, ..., A_k>$ in the activity set $A$. $A_1$ is the start activity of the interation, and $A_k$ is the terminal activity. We often use a directed edge pointing from $A_k$ to $A_1$ in workflow graphical model to express the interation relation.

After introducing temporal relations, activity tree can be transformed into temporal activity graph. In this graph, nodes represent activities, directed edges represent temporal relations and decomposition relations. The attribution information of activities can be added to nodes.

Next, an approach to transform the activity tree into temporal activity graph will be introduced, which is discussed according to the type of decomposition relation between parent and child activities.

**Case 1.** aggregation relation

For an activity $A$, its decomposed sub-activities are $A_1$, $A_2$, .., $A_n$, which satisfy $a(A_i, A)$, $1 \le i \le n$, then we get: 1) if these sub-activities are in the relation of SERIAL, just determine the execution order of the activities, and mark the activity $A$ as SERIAL type, which is called the node type; and 2) if these sub-activities execute in parallel, i.e., in the relation of AND, then mark the activity $A$ as AND type. If the execution order of sub-activities is not certain, the default type of the parent activity will be marked as AND for the purpose of execution efficiency.

**Theorem 1.** For a parent activity $A$, if the relations between its sub-activities contain both SERIAL and AND relations, then the AND relation can be eliminated by way of packaging the related sub-activities in AND relation into a composite activity.

**Proof.** Set the sub-activities, $a_1$, $a_2$,..., $a_m$, $1<m<n$, to be in AND relation, and their nearest common predecessor and successor are $A_s$ and $A_e$. From Definition 10, we get that after packaging these sub-activities into a composite activity $A_i$, $A_s$ is the predecessor of $A_i$, and $A_e$ is the successor of $A_i$. So $A_s$, $A_i$ and $A_e$ are in the relation of SERIAL. Do the same operations to all the sub-activities that are in AND relation, then we eventually get the sub-activities containing only SERIAL relation.

**Case 2.** generalization relation

For an activity $A$, its decomposed sub-activities are $A_1$, $A_2$, .., $A_n$, which satisfy $g(A_i, A)$, $1 \le i \le n$, then the parent activity $A$ can be marked as OR type. As a special case of the parent activity, the sub-activity inherits the features and functions of the parent activity, and has its own application conditions, which can be used to select the most suitable activity to execute according to the runtime contexts.

Fig.3 is a temporal activity graph containing temporal relation after using the above approach. In this figure, atom activity, composite activity and flexible activity are represented by different kind of nodes. Every parent node is marked as SERIAL, OR, or AND type. For a node marked as SERIAL type, it means the execution order of its sub-activities is by sequence, from left to right. AND type means that the node's sub-activities execute in parallel, and OR type means that only one of the sub-activities will be selected to execute. For an interation, if any, its start node and terminal node are used to represent the iteration. For example, the interation(Interation-1) in Fig.3 is represented by the start node C22 and the terminal node C23.

Thus, we established a hierarchical temporal process model with multiple granularity and refinement degree. In this model, temporal relations indicate the execution orders, and the different level of abstraction embodied by granularity will help to the gradual modeling and refinement. To emphasize the vertical relation of the temporal activity graph, we call it the temporal activity tree.

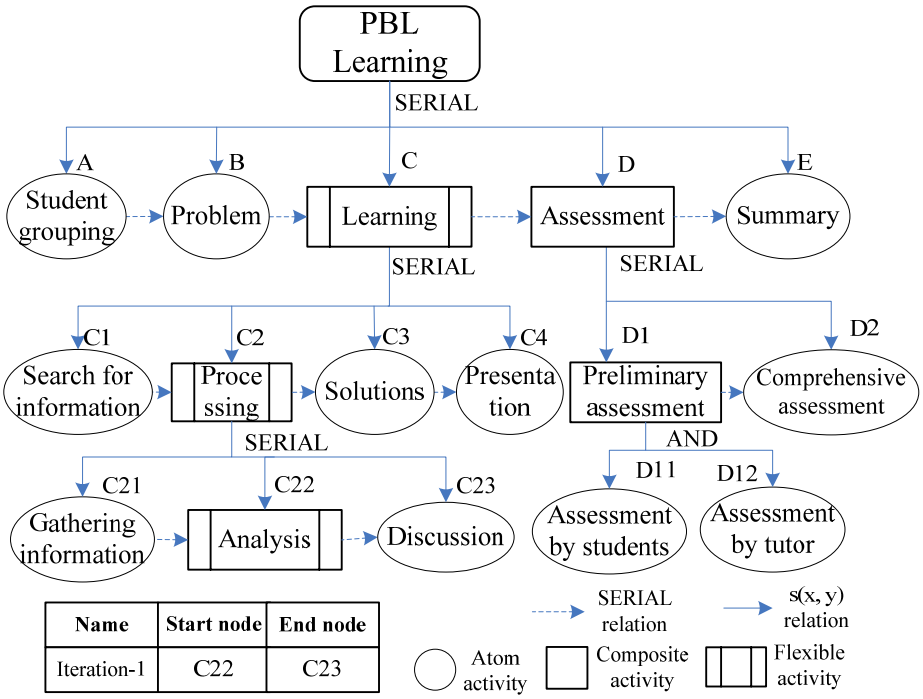


Figure 3. Activity tree with temporal relation

## B. Execution Constraints of TAT

For the temporal activity tree, a certain mechanism to restrain the proper execution and the states of parent activity and sub-activities is required greatly. From rule 1, functions and goal of the parent activity are distributed into its sub-activities to achieve, and the sub-activities, in generalization relation to the parent, also inherit from the parent and are the embodiments of the parent activity. The state of a sub-activity, therefore, depends on not only the state of the parent activity, but the manner that the parent was decomposed in. That is, it is also related to the node type of the parent activity as discussed below.

**Rule 2.** the state of the sub-activity relies on the state of the parent as well as the node type.

Set the parent activity is $P$, and its state is $s$. Use *Decompose* ($P$) to get its sub-activity set, $\{c_1, c_2, ..., c_n\}$. Set $s_i$ is the state of $c_i$, then the corresponding state set is $S=\{s_1, s_2, ..., s_n\}$.

1) $P$ is marked as SERIAL type

The sub-activities of $P$ form a sequence of activities, $c_1, c_2, ..., c_n$, executing successively.

a) $s = W$, then $\forall s_i \in S$, $s_i = W$;

b) $s = R$, then $\forall s_i \in S$, $s_i = R$;

c) $s = E$, then $s_1 = E$;

d) $\exists s_i \in S$, $s_i = C$, $1 \le i \le n-1$, then $s_{i+1} = E$;

e) $\forall s_i \in S$, $s_i = C$, or $s_n = C$, then $s = C$;

f) $\exists s_i \in S$, $s_i = A$, then $s = A$, $undo(c_1, ..., c_{i-1})$;

From the above rule items, we can see that there exist dependencies between sub-activities and parent activity. Herein, Waiting($W$) indicates that the execution conditions are not ready. So if the state of parent is $W$, the state of all its sub-activities is $W$. If the state of the parent is Ready($R$), then the state of all the sub-activities is Ready. For sub-activities, only when the previous activity is completed, may the subsequent activity start. When all the sub-activities are completed, the parent is completed. When any one of the sub-activities is aborted, the state of the parent is Aborted, meanwhile rollbacking all the finished activities, i.e., $c_1,...,c_{i-1}$.

2) $P$ is marked as AND type

a) $s = W$, then $\forall s_i \in S$, $s_i = W$;

b) $s = R$, then $\forall s_i \in S$, $s_i = R$;

c) $s = E$, then $\forall s_i \in S$, $s_i = E$;

d) $\forall s_i \in S$, $s_i = C$, then $s = C$;

e) $\exists s_i \in S$, $s_i = A$, then $s = A$, $undo(c_j)$, $j=1,...,m$ and $s_j = C$;

Due to the parallelism of sub-activities, when the parent activity starts, all the sub-activities begin to start; when any one of the sub-activities is aborted, the parent activity will be aborted, and rollback all the completed sub-activities.

3) $P$ is marked as OR type

a) $\exists s_i \in S$, $selected(s_i)$, and $s \in \{W, R, E\}$, then $s_i = s$;

b) $\exists s_i \in S$, $selected(s_i)$, and $s_i = C$, then $s = C$;

c) $\exists s_i \in S$, $selected(s_i)$, and $s_i = A$, then $s = A$;

Obviously, the state of the triggered sub-activity relies on the parent's state (i.e., $W$, $R$, $E$). After the sub-activity is completed, the parent activity is also completed. When exceptions occur in sub-activity, the parent activity is aborted.

After introducing the temporal relation into the model, the mechanism of decomposing and transferring the parent activity's execution conditions and trigger events to sub-activities should also be considered, and it is crucial to the proper execution of activities. Recalling the modeling process, actually, we often establish the parent activities and their relations first, and then build the decomposed sub-activities and their relations later by means of from coarse granularity to fine granularity. Obviously, the sub-activities should inherit the decomposed execution conditions and the context from the parent activity at runtime. The events triggering the execution of parent activity should also be delivered to the sub-activities to trigger their executions.

**Rule 3.** the execution conditions and the trigger events of sub-activities rely on those of the parent activity. It is a top-down delivering process, from parent to child activities. The following discussion is based on the parent's type.

**1. the delivering and decomposition of the condition $c_B$ of the parent activity**

1) the parent is marked as SERIAL type. Deliver $c_B$ to the first sub-activity. The execution conditions of other sub-activities are set as TRUE.

2) the parent is marked as AND type. Deliver $c_B$ to every sub-activity.

3) the parent is marked as OR type. Decompose $c_B$ into several sub-conditions according to the requirements of each path, and deliver the sub-conditions to the corresponding sub-activities in each path.

**2. the delivering of events**

The events that can cause a state change of parent activity should be delivered to sub-activities to change their states. We mainly consider the event that the previous activity is completed (i.e., event Done). Set the predecessor of the parent activity is $A$. After the event Done($A$) occurs, we have:

1) the parent is marked as SERIAL type. Deliver the event Done($A$) to the first sub-activity. According to the rule 2, the trigger events of other activities rely on the completion of the previous activity.

2) the parent is marked as AND type. Deliver the event Done($A$) to every sub-activity.

3) the parent is marked as OR type. Deliver the event Done($A$) to the selected sub-activity.

### *C. Evaluation Indexes of the Decomposition Model*

In order to characterize the modeling process, some evaluation indexes with the corresponding calculation equations are defined.

**Definition 13 (Dom operator)** *Dom* operator is used to obtain the atom activities for a node $A$ by: 1) if $A$ is an atom activity, $Dom(A)=\{A\}$; 2) if $A$ is a composite activity, $Dom(A)= Dom(A_1) \cup Dom(A_2) \cup ..... Dom(A_n)$, where $sc(A_i, A)$, $1 \le i \le n$; 3) if $A$ is an empty composite activity (i.e., having no child), $Dom(A)=\varnothing$.

**Definition 14 (Certainty)** the certainty can be described by: $\alpha(T) = \dfrac{|Dom(T)|}{counts(T) - fc_{prvs} - 1}$, $\alpha \in [0,1]$, where $|Dom(T)|$ is the count of atom activities in the tree $T$, $counts(T)$ is the count of all the nodes in the tree, $fc_{prvs}$ is the count of flexible and composite activities in the tree at the previous modeling stage. 1) $\alpha=1$, the activity tree consists entirely of atom activities, and the certainty is the biggest; 2) $\alpha=0$, the activity tree consists of composite or flexible activities but no atom activity, and the certainty is the smallest; 3) $\alpha \in (0,1)$, the activity tree consists of atom, composite and flexible activities, the value of $\alpha$ is between 0 and 1.

**Definition 15 (Layer plot ratio)** the layer plot ratio is used to describe the average decomposition degree of

activities, $\beta(T)=\frac{counts(T)-1}{height(T)}$, where *height*(*T*) is the height of the tree *T* (excluding the root). When $\beta$=1, the decomposition degree of activities reaches the minimum, and the activity tree degenerates into a linear list.

**Definition 16 (Ambiguity)** the ambiguity can be described by: $\gamma(T)=\frac{FAcounts(T)-f_{prvs}}{counts(T)-1}$, where *FAcounts*(*T*) is the count of the flexible activities of tree *T*, $f_{prvs}$ is the count of flexible activities at the previous modeling stage. When γ=0, the activity tree has no flexible activity and has a minimum ambiguity. When the tree consists entirely of flexible activity, it has the biggest ambiguity.

**Definition 17 (Activity modeling difficulty)** the activity modeling difficulty is used to describe the efforts and difficulties for modeling an activity, $eff(A)=\frac{counts(A)}{counts(T)-1}$, where *A* is a flexible or composite activity. It means the proportion of activities generated by activity *A* in all the activities of the tree when a modeling stage is completed (e.g., when flexible activity *A* is entirely determined).

## IV. Interpreting and Execution of the Decomposition Model

The most important issue to run a workflow model is the way to interpret and execute activities, and to control their execution orders. For the former, the difficulty is the execution of flexible activity [9-12], and for the latter, ECA rules as well as the workflow engine for interpreting and performing ECA rules are always adopted. The event and condition of an ECA rule can be merged as one under certain conditions, which is called a trigger [13]. After that, an ECA rule consists of two parts (trigger, action). When the structure of the workflow process is emphasized, the condition of an ECA rule can be ignored, and the ECA rule can be rewritten as $a \rightarrow b$, which represents when activity *a* is completed, *b* will be triggered to start.

**Definition 18 (Flexible rule)** an ECA rule consisting of a composite or flexible activity is referred to as a flexible rule. An important aim of flexible rule is to support the gradual refinement process of temporal activity tree, and to maintain the simplicity and maintainability of the rule base.

Next, the incremental refinement algorithm for flexible rules will be proposed. It can be used to refine the flexible rule when the coarse granular activity in the rule is decomposed into several fine granular activities. In this algorithm, *P* is the composite or flexible activity to be refined, *A* is the activity set. For simplify, the rule containing activity *P* is called the related rule of *P*. For a rule $a \rightarrow b$, *a* is called the antecedent of the rule, and *b* is called the consequent.

**Algorithm 1.** FRule_ Refinement
**Input:** *P*, and *R* (the related rule of *P*)
**Output:** the refined rule set *R*'

```
R'←∅;
A←Sort(Decompose (P));
If P is marked as SERIAL{
    for each element a_i in A{
        if 1≤i<|A| {
            create ECA rule "a_i→a_{i+1}" ;
            add rule "a_i→a_{i+1}" to R';
        }
    } //end for
    for each rule r in R {
        if P is in the consequent of r{
            Replace P with a_1 for r;
            add r to R';
        }
        if P is in the antecedent of r {
            Replace P with a_|A| for r;
            add r to R';
        }
    }//end for
}//end if
If P is marked as AND {
    Create the AND expression "a_1∧…∧a_|A|";
    for each rule r in R {
        replace P with "a_1∧…∧a_|A|" for r;
        add r to R';
    }
} //end if
If P is marked as OR {
    for each rule r in R {
        for each element a_i in A{
            r'←r ;
            Replace P with a_i for r';
            add r' to R';
        }
    } //end for
} //end if
return ExecutionConstrains(R');
```

In the above algorithm, the function Sort() is used to sort the sub-activities according to the SERIAL relation, and ExecutionConstrains() is used to add corresponding execution conditions to the generated rule. Next, a common execution algorithm, i.e., GeneralActivity_Execution(*a*), for activities including atom, composite and flexible activities is described (see Fig.4), and its main steps include:

1. Determine the type of activity *a*. if it is an atom activity, execute it directly.

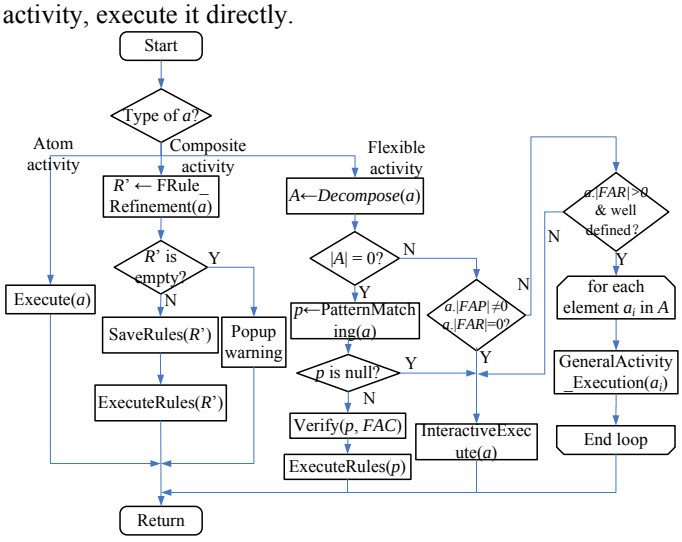

Figure 4. Flowchart of common activity execution algorithm

2. If $a$ is a composite activity, call the algorithm 1 to refine it. If the result of the refinement is not empty, store the returned rule set and execute the rules; if the result is empty, popup the corresponding tips and then execute the next activity.

3. If $a$ is a flexible activity, do the following steps accordingly.

(1) $a$ is an empty flexible activity. Search the process template library and try to find the process matching the goal and function of $a$. If the matching is successful, return the selected process template; or else, run the flexible activity in a manner of user interaction.

(2) $a$ is not an empty flexible activity. Analyze its sub-activities. If there exist clearly specified atom activities or composite activities in the activity pool, but the relationship among the activities has not been defined, add the activity relations into the flexible activity in a user interactive manner and then execute it. If there exists well defined activity relation set, for each sub-activity $a_i$ in the relation set, execute the algorithm GeneralActivity_Execution($a_i$) recursively according to the execution order of sub-activities. In other cases, execute the activity in a user interactive manner.

(3) use the constraints *FAC* to verify the generated process. If the verification is OK, the process can be executed, or else the warning message will be popup and the control logic turns to regenerate the process again.

## V. Applications

The proposed approach was applied to the intelligent tutoring system[14] to develop an intelligent learning management system based on PBL(Problem-based Learning). In this system, flexible workflow technology is adopted to model and manage the learning process, which improves the system's capability to deal with uncertainties greatly.

PBL is a student-centered learning based on problems in which students learn about a subject through the experience of problem solving[15]. The learning process includes steps such as heterogeneous grouping of students, presenting problem scenario, self-directed learning of students, and the assessment and summary of the learning by tutors. In these steps, heterogeneous grouping (A), introducing problem scenario (B) and the summary (E) are clearly specified atom activities, which can be executed directly. Assessment (D) is a composite activity and need to be further decomposed. There exists ambiguity in the learning of students (C), which can be packaged as a flexible activity, and then be refined gradually according to the presented problem and the detailed application it belongs to (see stage 1 in Fig.5). For the assessment activity, it can be decomposed into two sub-activities: preliminary assessment (D1) and comprehensive assessment (D2), and the temporal relations is: $D1 \prec D2$.

The domain characteristics should be considered when the learning of students (C) is decomposed. According to the character of computer-related courses, it can be decomposed into sub-activities: search for information (C1), processing information (C2), forming the solutions (C3), sharing the result of learning (C4), and $C1 \prec C2 \prec C3 \prec C4$ (see stage 2 in Fig.5). In these sub-activities, C2 is packaged as a flexible activity due to the fuzziness in it. In the preliminary assessment, the views of both students and tutors should be considered. So D1 can be further decomposed into the students mutual assessment (D11) and the tutor assessment (D12), which are performed in parallel (AND relation). Processing information (C2) is roughly divided into gathering information (C21), analysis (C22) and discussion (C23), which satisfy $C21 \prec C22 \prec C23$. This incremental modeling process of PBL learning can be seen in Fig.5, where the right of the figure is the varying of rule set, and function *FRR*() is used to generate the rule set corresponding to the activity tree.

There exists uncertain and fuzzy information in the modeling process. For example, to the learning activity, the task itself and its temporal relation can not be determined. If a traditional method is used, the vague and incomplete information may result in great difficulty of the modeling. For example, considering so much possibility makes the model bloated, and the whole learning process can not be expressed properly. While in our incremental modeling approach, the uncertain activity is packaged into a flexible activity, which can be refined to determined activities gradually, with the deepening of the project and the acquisition and accumulation of domain knowledge. This is an iterative, incremental modeling process, and as a result, a clearly specified process model will be formed.

Fig.6 illustrates the varying of certainty and ambiguity of the model during the incremental modeling process. From stage 1 to stage 3, with the transition of flexible elements and the gradual decomposition of composite activities, the certainty of the model increases, the ambiguity decreases, and the whole model gets more and more closer to a determined process model. Fig.7 shows that the varying of the layer plot ratio is not big, which indicates that the decomposition degree of activities is rational, avoiding a linear or flat process structure, and thus helps to reduce the difficulty of modeling. Fig.8 shows the modeling difficulties of the composite activities and flexible activities. We can see that activity C is with the highest modeling difficulty, and activity D and C2 are the next ones, which is consistent with the fact. In these activities, the ambiguity of C is the highest, so the effort put to decompose and to determine it is the most. As the calculation of modeling difficulty will be gradually corrected, its value will be more and more closer to the true value.

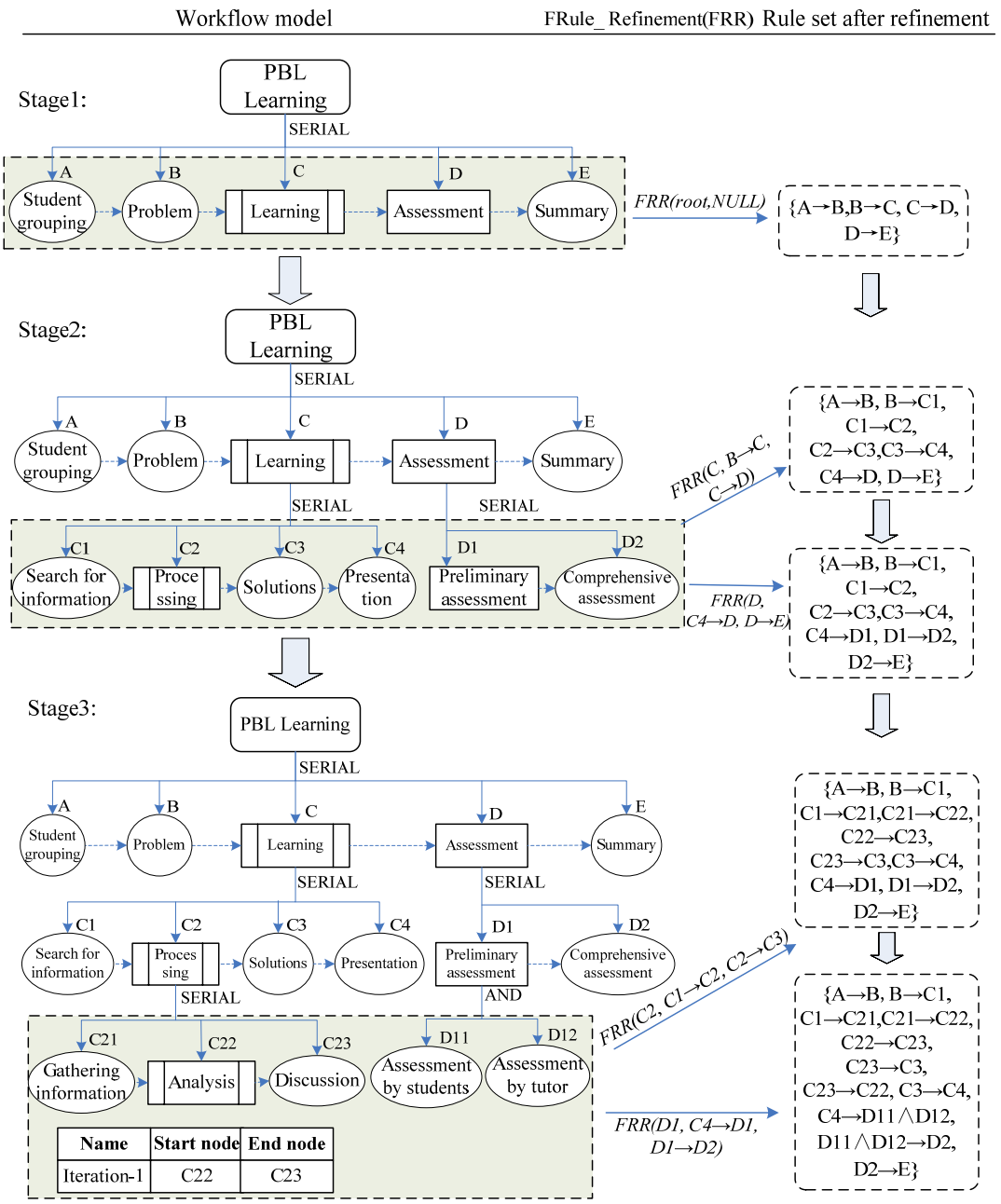


Figure 5. Incremental modeling and refinement for PBL learning

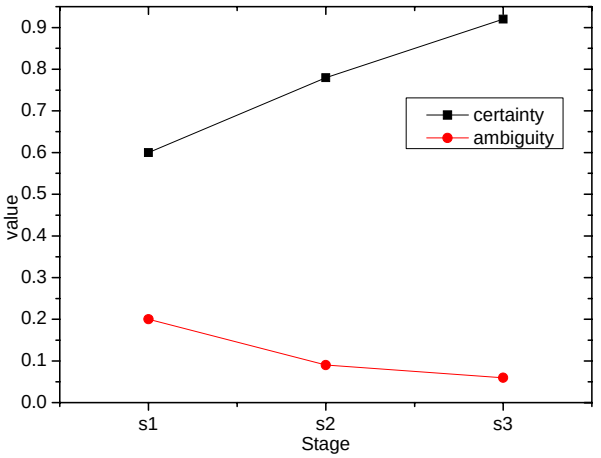


Figure 6. The variant of certainty and ambiguity of process model

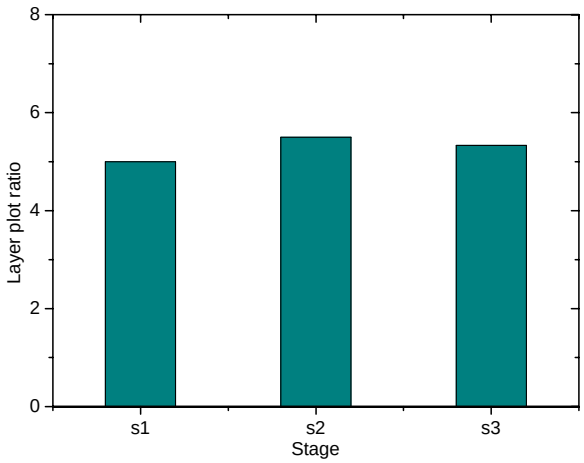


Figure 7. The variant of the floor area ratio of process model

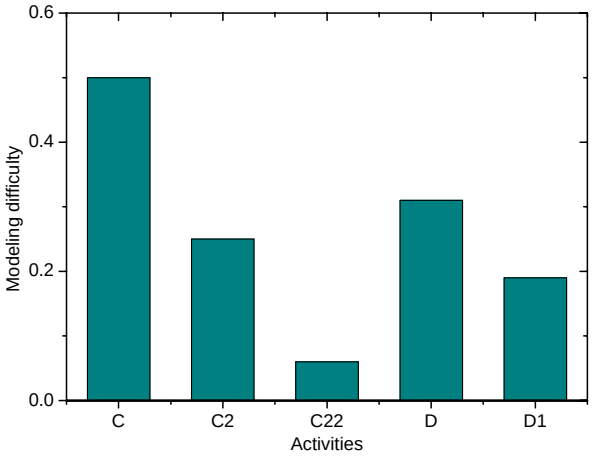


Figure 8. The modeling difficulty of activities

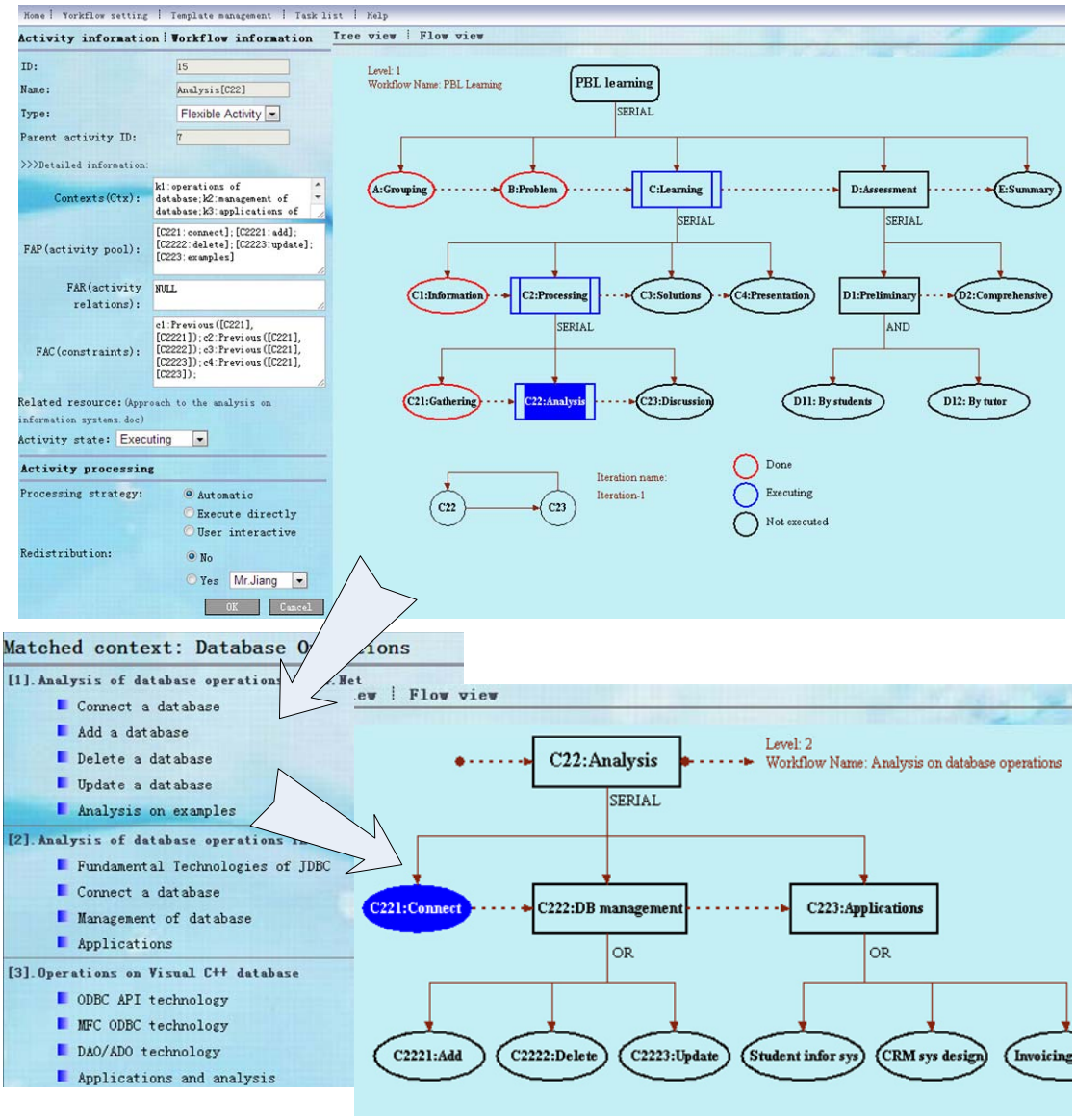


Fig.9 Interface of the system and a determination process

Fig.9 illustrates a determination process of flexible activity C22, where red nodes represent the completed activities, green nodes represent the executing activities, and black nodes represent activities that are not executed yet. The detailed information of C22 can be seen in the left part of the main interface. Its context indicates that this activity is suitable for database operations. In the activity pool *FAP*, there exist some preset activities such

as connect (C221), add (C2221), delete (C2222), update (C2223), and examples (C223). The constraint *FAC* includes: *Previous*([C221], [C2221]), *Previous*([C221], [C2222]), *Previous*([C221], [C2223]), *Previous*([C221], [C223]). This context will help to find the best suited process template in the database for the activity. If it succeeds, the matched process templates will be listed according to their priorities. The default strategy for selecting process template is *automatic*, while other ways such as *user interactive*, and *execute directly* are also provided. After selecting the proper process template, the constraint *FAC* will used to verify the process. If it is OK, the process template will be applied to the flexible activity to execute (see Fig.9).

## VI. Conclusions

A pure traditional process model is much more suitable for describing temporal relations among activities than hierarchical relations often used for gradually workflow modeling and activity refinement; while the model consisting of only hierarchical relations is hard to describe and control the execution order of activities. Only the work integrating both of them can overcome the weakness and improve the flexibility of the workflow. That is, from the start of vertical decomposition of activities, one can introduce temporal relations into the decomposition model, and use a flexible activity to package uncertain factors and then gradually refine it. Therefore, an approach to modeling flexible workflow by activity decomposition and activity incremental refinement is presented. Activities as well as the decomposition relations and rules are analyzed and defined, and an activity decomposition model with different granularity and multiple abstract levels is established. After introducing the temporal relations into the model, the execution constraints and the related rules are discussed. In order to give an insight of the varying of model characteristics and to assess the model, some evaluation indexes are described. A common activity execution algorithm is proposed for multiple activity categories. The proposed approach was applied to an intelligent learning platform to develop a workflow application based on PBL, and the results indicate the effectiveness.

## Acknowledgments

This work was supported by National Natural Science Foundation of China (61171141, 61142012), Science and Technology Planning Project of Guangdong Province, China (2012B010600014, 2012B010500025), and Open project of key laboratory of innovation method and decision management system of Guangdong Province (2011A060901001-13B, 2011A060901001-09D).

## References

[1] S. Zhang, S. Li, S. Zhang, and N. Gu, "Workflow Based on Interaction and Machine Learning," *Mini-micro Systems*, vol.26, pp.1270-1274, July 2005.
[2] J. Li, W. Wang, and F. Yang, "Review on approaches of flexible workflow," *Computer Integrated Manufacturing Systems*, vol.16, pp.1569-1577, August 2010.
[3] G. Yang, Y. Zheng, and G. Wang, "An Application Research on the Workflow-based Large-scale Hospital Information System Integration," *Journal of Computers*, vol.6, pp.106-113, January 2011.
[4] S. Li, X. Shao, and J. Chang, "Dynamic workflow modeling oriented to product design process," *Computer Integrated Manufacturing Systems*, vol.18, pp.1136-1144, June 2012.
[5] S. A. Chun, V. Atluri, and N. R. Adam, "Domain knowledge-based automatic workflow generation," in *13th International Conference on Database and Expert Systems Applications (DEXA 2002)*, Berlin, Germany, 2002.
[6] S. Nurcan, and M. H. Edme, "Intention-driven modeling for flexible workflow applications," *Software Process Improvement and Practice*, vol.10, pp.363-377, Oct. 2005.
[7] A. Luntovskyy, S. Uhlig, and D. Guetter, "A flexible Workflow Management System for CAD of Telecommunication Networks," *Pomiary Automatyka Kontrola*, vol.56, pp.1166-1169, 2010.
[8] Y. Zhang, "The Design and Implementation of Software Process Modeling Language Based on Activity Decomposition & ECA Rule," Hunan university, 2004.
[9] G. Ye, X. Li, D. Yu, Z. Li, and J. Yin, "The Design and Implementation of Workflow Engine for Spacecraft Automatic Testing," *Journal of Computers*, vol.6, pp.1145-1151, June 2011.
[10] J. Bae, H. Bae, S. Kang, and Y. Kim, "Automatic control of workflow processes using ECA rules," *IEEE Transactions on Knowledge and Data Engineering*, vol.16, pp.1010-1023, August 2004.
[11] S. Zhang, Y. Xiang, Y. Shen, and M. Shi, "Workflow execution mechanism in grid workflow generation," *Journal on Communications*, vol.29, pp.43-50, June 2008.
[12] C. Yan, H. Luo, Z. Hu, X. Li, and Y. Zhang, "Deadline guarantee enhanced scheduling of scientific workflow applications in grid," *Journal of Computers*, vol.8, pp.842-850, April 2013.
[13] J. Hu, S. Zhang, and X. Yu, "A Workflow Model Based on ECA Rules and Activity Decomposition," *Journal of Software*, vol.13, pp.761-767, April 2002.
[14] Y. Jiang, J. Han, and W. Wu, "An Adaptive Approach to Personalized Learning Sequence Generation," *Computer Science*, vol.40, pp.204-209, August 2013.
[15] J. R. Savery, "Overview of Problem-based Learning: Definitions and Distinctions," *Interdisciplinary Journal of Problem-based Learning*, vol.1, pp.9-20, May 2006.

**Yanrong Jiang** was born in 1976. He received his Ph.D. degree in 2007 in Computer Science from South China University of Technology, China. His research interests include human-robot social interaction, affective computing, attentional control, and context-aware computing, etc. He is now working at the School of Computer, Guangdong University of Technology, China.

**Weihua Li** was born in 1957. She received her Ph.D degree in Computer Science from Sun Yat-Sen University, China. She is now a professor at the School of Computer, Guangdong University of Technology, China. Her research interests include Agent oriented computing, intelligent software, etc.

**Jingtao Yang** was born in 1971. He received his Ph.D. degree in 2005 in Computer Science from South China University of Technology, China. His research interests include web service, software engineering, etc.