Study of the Reusable Workflow System

Haibo Li[1], Dechen Zhan[2]

1 Centre of Intelligent Computing of Enterprises, School of Computer Science and Engineering, Harbin Institute of Technology, Harbin, Heilongjiang 150080, China
2 School of Engineer, Northeast Agriculture University, Harbin, Heilongjiang 150030, China
Email: lihaibo@hit.edu.cn; dechen@hit.edu.cn

Abstract: Reusable development can promote the productivity of large workflow systems development. However it has not precluded developers from designing workflow system tailoring to users' specific needs, though workflow management coalition standardized the five kinds of abstract interfaces of workflow enactment service in workflow reference model. Specific business process characteristics are still supported by specific workflow system. Method of software reuse is introduced to enhance reusability of the core of workflow system, i.e. workflow engine. In component environment, general functionalities of workflow engine are abstracted from business component, resulting in that the reusability of business component is extended into workflow engine. So two aspects are considered, i.e. component-based business processes development and reusable workflow engine development. The proposed approach is supported by a set of composition methods and reuse strategies. Through application and comparison, we show that different business requirements are met by reusing the workflow engine**.** [The Journal of American Science. 2005;1(2):51-60].

Keywords: workflow engine; software reuse; reuse strategy; component

1 Introduction

As a technology for modelling and execution of business processes, workflow management has emerged for many years. Workflow management coalition (WfMC) proposed a reference model (WFMC, 1994), as a common framework in order to give a guideline for developing workflow management system. At the heart of this framework is workflow engine. Five kinds of interfaces, i.e. modelling tools, client application, external applications, administration and monitoring tools, other workflow enactment services are defined for execution service and used as a standard. However the definition for these interfaces focus on the syntax and has no clear enough semantics and usage (Sheth, 1999). Therefore, different interpretations to the interfaces are given and different viewpoints are proposed by developers when they confront with specific requirement. Heterogeneous business processes executed by users have their own features, so different requirements need to be met accordingly. For example, task allocation is clear in some processes such as document or order auditing so the processes can be handled by computer completely, however it is unclear, such as equipment repair process, in which workers of different levels for skill must be chosen by supervisor. Handling key data in some processes, such as storage or finance management, demands those functionalities supporting transaction management and exception handling, so as to enhance reliability of system (Puusjari, 1997). In those timed processes, workflow system must provide mechanism of event triggering automatically. From these examples, we can see that features of processes determine that of the workflow engine. The numerous commercial workflow management products also can be found that they all offer their specific architectures, can integrate application from third part easily, and can even embed functionality of developing application to some extent, but these functions can seldom be utilized completely except for basic ones.

To sum up, in order to customize workflow system according to business characteristic freely and tailor some features conveniently without influencing other functionalities in system, the core of workflow system, i.e. workflow engine should be reusable first. Reusability of workflow engine means that new workflow engine can be created through reconfiguring, tailoring and inheriting existing workflow engine, so as to meet new business requirement. Software component technology is a way to raise efficiency and system's reusability (Batory, 1992; Novak, 1997; Lattanzi, 1998; Mili, 1997; Rajlich, 1996). So it is an interesting topic to apply the software component concept and method to workflow engine development. In order to improve reusability, many workflow systems support different open mechanism. Some provide API to developers (WorkMovr, 2001; Drala, 2001), and some provide source code to developers, such as Fujitsu (2001) and Vivtek (2001). Some researchers started with analyzing architecture of workflow directly (Dragon, 2001; Zhuge, 2003). This article is written in the belief that workflow is process logic separated from information system. To discuss the reusability of workflow engine cannot be independent of the analysis to business component. So this article starts with some relevant definitions to business component, then abstracts basic and required functionalities from business object to comprise workflow engine.

The motivation of this paper is to change traditional development approach of workflow system, to investigate the mechanism of reusability of workflow engine, and to propose method and strategy to support the component-based workflow engine development. Developers can develop a new workflow engine by reusing an old one, customize, tailor and extend functions so that specific requirements of different users can be met. Therefore, the management ideology of enterprises would be laid out as exactly as possible. Some unavoidable relevant definitions are provided in this paper.

In the following section, we first discuss the design principles and required features of workflow engine, and then define some conceptions. In section 3, the composition mode of workflow engine components is given. Section 5 presents method of reuse a workflow engine. An application example is shown in section 5, and Section 6 gives a conclusion.

2 Workflow Engine Component

2.1 Design principles

Almost the definitions for all workflows represent a workflow management system as automates the process logic. So automating, dispatching artificially and controlling process are the most key core function of a workflow engine. A reusable workflow management system should be a system whose functionalities could be extended from the workflow engine. A reusable workflow engine should have the power to support process control, further more it is reusable. The research for reusability of workflow engine mainly involves reusability of process control. Aiming at necessity, flexibility and low cost, but completeness and complexity, only basic functionalities and features are realized. Control flow among activities, probing conditions and managing organization are considered primarily in this paper, while the rest functionalities, such as event triggering automatically, transaction management and exception handling are not involved.

2.2 Definition of workflow

By introducing concepts and methods relating to software component into workflow engine, generally, reconfiguration of workflow engine is represented as "According to specific business requirement, following specific principle, workflow engine can be converted from existing form to target". If a new component *C'* can be defined by reusing a certain form of an existing component *C*, we say component *C* is reused by component *C'*.

A workflow system usually consists of two parts: workflow modeling and workflow execution. The former uses definition tools to generate workflow specification in some middle language which can be interpreted by computer finally. Workflow specification contains a set of data definitions, such as the work list, the logic sequence between tasks, workflow relevant data, the role model data and a set of control condition. All these data comprise the properties and behaviors. Activity and control structure are denoted by a node several kinds of routing control nodes respectively. So we can use a directed acyclic graph (DAG) to represent its process structure. Arrows in a DAG denote partial order (i.e., logic order) between nodes. Workflow engine constructs and executes activities by participant and determines logic order of activities according to some conditional constraints. The part of workflow execution provides an executable environment for building, running and managing workflow system. So the definitions can be summarized as follows.

Definition 1 (*Workflow specification*). A workflow specification is denoted as a 5-tuple $ws = < TN, CN, D, O, R >$, where

(i) $TN=\{tn_1, tn_2, ..., tn_n\}$ is a set of task nodes.

(ii) $CN=\{cn_1, cn_2, ..., cn_n\}$ is a set of control nodes.

(iii) D is a set of workflow relevant data (see next section).

(iv) O is a set of organization model.

(v) R is a superset, $R=(DR, CR)$, where DR is a set of data dependency and CR is a set of control dependency (see next section).

Definition 2 (*Workflow*). A workflow is a 4-tuple, $wf=(id, ws, A, \prec)$, where

(i) *id* is an identifier assigned to the workflow.

(ii) *ws* is the associated workflow specification.

(iii) A is a set of activities and

(iv) $\prec$ is a set of partial order.

Definition 2 describes that a workflow system constructs, executes and controlling routing according to workflow specification.

2.3 Required features of workflow engine

The definition of workflow from section above is only a common one without features of software component. In order to design a completely reusable workflow engine, we need a further investigation and start with business component, abstracting the features of workflow, involving workflow relevant data, data dependency and control dependency. So some definitions about business component are presented below. The application in software component environment is composed of entities (i.e. business object) describing business domain. Business object descript information entity and physical entity with respective semantics, such as order, report, equipment and staff in diary business processes. The conception of business object is different from that of object-oriented in software engineer domain. It characterizes those existent business entities in enterprise, not nonobjective in software systems.

Definition 3 (*Business object*). A business object is a 6-tuple, can be denoted as $bo=(id, A, M, S, \delta, \psi)$, where

(i) *id* is an identifier assigned to the business object.

(ii) A is a set of properties a_i of bo, $(i=1,2,\ldots, m)$, $a_j(bo)$ denotes property a_j of bo.

(iii) M is a set of business operations m_i which act on bo, $(i=1,2,\ldots, n)$.

(iv) S is a state space of bo which contains k states $s_1, ..., s_k$. $s_j(bo)$ denotes state s_j, $S=\phi$ denotes that bo is a business object without state. The state of business object depends on its properties. The transformation of business object property depends on that of state and the transformation process from one state to another is called state transformation.

(v) δ: $S \rightarrow P(2^A)$ is a map function from state to property. $\delta(s_i)=P(a_{i1}, a_{i2}, ..., a_{iu})$ or $P(A_i)$ denotes that state s_i of bo can be denoted by u properties $A_i=\{a_{i1}, a_{i2}, ..., a_{iu}\} \subseteq A$. P is a set of function or predication expressions which represents value characteristics of properties in the state s_i and constraint between them.

(vi) $\psi: S \times M \rightarrow S$ is transformation function. $\psi(s_i, m)=s_j$ denotes that a state s_i can be transformated to s_j by executing method m in state s_i. The transformation set of bo is denoted as $\psi(bo)$, $\psi_{ij}(bo, m)$ denotes there exists a m such that $\psi(s_i, m)=s_j$.

Definition 4 (*Business activity*). A business activity is a set of business operations which are executed uninterruptedly by a specific role r, denoted as a 3-tuple $ba=(r, OP, \prec)$, where

(i) r is a set of roles.

(ii) OP is a set of business operations op_i $(i=1,2,\ldots, k)$

(iii) $\prec$ is a set of partial orders between business operations.

A business activity is composed of a series of business operations in workflow domain, equivalent to the set of business operations in component domain. Start states of an activity are exhibited as start states of one or more business objects. From start states, different business operations uninterruptedly transform states of activities, to target states of activities (i.e. a series of business objects). So the start and end states determine precondition and postpositional conditions respectively, and simultaneously the routing of business process depends on certain states.

Definition 5 (*Workflow relevant data*). Workflow relevant data can be denoted as a 4-tuple *rd*=(*id, BO, A, S'*), where

(i) *id* is an identifier assigned to the data.

(ii) *BO* is a set of business objects containing *rd*, which only relates to start, end states and routing between activities but other states of business objects interacting with each other.

(iii) *A* is a set of activities containing business operations on business objects set.

(iv) *S'* is a state space of *BO*. It is obvisous that *S'* only relates to start, end states of activities and routing states between activities, but other states of business objects in activity.

Workflow relevant data describes the minimal set to support control process in workflow system such that $S' \subseteq S$, where S is a set of states of business objects in *BO*. It also describes properties which only relate to starting an activity, completing an activity and routing between activities.

The granularities of data and operation in workflow domain are larger than those in software component domain. Here we focus on the consideration including starting, completing and controlling an activity. We are only concerned about these aspects in workflow engine design. The interaction between business objects in an activity is handled by mechanism of software component. Workflow relevant data provides basal data for starting, completing and controlling activities. In workflow domain, the generation of an activity state s_j depends on the generation of other activity state s_i, denoted as $s_j \rightarrow s_i$. Furthermore, state is predication of property, so that the dependency between properties depends on that of states. To stand out the dependency between properties, in workflow domain, we separate property (i.e. workflow relevant data) from predication representing state.

Definition 6. Suppose that $\delta(s_i)=P_i(D_i)$, $\delta(s_j)=P_j(D_j)$, the dependency between s_i and s_j is denoted *as* $P_i(D_i) \rightarrow D_j$, means that there exists a data dependency from D_i to D_j on predication P_i, and then generates state s_i.

Predication P_j is a condition for checking up validity of D_j because P_j is a predication on D_j. If s_i is a start state and s_i is an end state of an activity, P_i and P_j are called precondition and postpositional conditions respectively.

Business process contains a series of activities, especially a start activity and an end activity. There exists a partial order between activities, and give any two activities a_i and a_j , $a_i \prec a_j$ is denoted that a_i is executed before a_j. This routing between activities is controlled by a set of predications, denoted as $P= \{P_1, P_2,\ldots,P_n\}$, where P_i is a condition which must be met when an activity a_i completes and triggers next activity a_j. If there always exists possible partial order between a_i and $a_{j1},a_{j2},\ldots,a_{jn}$, we say that there is a type of control rule between a_i and $a_{j1},a_{j2},\ldots,a_{jn}$. Control rules determine workflow control routing according to predication P', where $P' \subseteq P$ and one type of control rule forms one type of control structure. The six types of control structures involving sequence, AND-Split, And-Join, OR-Split, OR-Join, and LOOP have been proven effective for business process automation and have widespread support in current workflow products (Aalst, 2000). Control structures set is denoted as *CN={SEQUENCE,ANDSPLIT,ANDJOIN,ORSPLIT,ORJOIN,LOOP}*, where

- *SEQUENCE*: Activities are executed in order under a single thread of execution, which means that the succeeding activity cannot start until the preceding activity is completed. The condition of routing between activity a_i and a_j is always true, denoted as $P(a_i, a_j)=TRUE$, and $Completed(a_i)= TRUE$, where *Completed* is postpositional condition for completing activity a_i.
- *ANDSPLIT*: A single thread of control splits into two or more threads which are executed in parallel within the workflow, allowing multiple activities to be executed simultaneously. The condition between predecessor activity a_i and every successor activities $a_{j1},a_{j2},\ldots,a_{jn}$ must satisfy $a_i \prec a_{j1} \wedge a_i \prec a_{j2} \wedge \ldots \wedge a_n \prec a_{jn}, P_1(a_i, a_{j1})= TRUE \wedge P_2(a_i, a_{j2})= TRUE \ldots \wedge P_n(a_i, a_{jn})= TRUE$, where P_j is a predication of routing between activity a_i and a_{ji}, and $Completed(a_i)= TRUE$.
- *ANDJOIN*: Two or more parallel executing

activities converge into a single common thread of control. The condition between every predecessor activities $a_{i1},a_{i2},...,a_{ik}$ and successor activity a_j must satisfy $a_{j1} \prec a_i \wedge a_{j2} \prec a_i \wedge ... \wedge a_{jn} \prec a_n$, and $Completed_1(a_{i1})=TRUE \wedge Completed_2(a_{i2})= TRUE \wedge ... \wedge Completed_n(a_{in})=TRUE$, and $P_1(a_{j1},a_i)= TRUE \wedge P_2(a_{j2},a_i)= TRUE ... \wedge P_n(a_{jn},a_i)= TRUE$.

- *ORSPLIT*: A single thread of control makes a decision upon which branch to take when encountered with multiple alternative workflow branches. The condition between predecessor activity a_i and every successor activities $a_{j1},a_{j2},...,a_{jn}$ must satisfy $Completed(a_i)=TRUE, P_1(a_i, a_{j1})=FALSE \wedge P_2(a_i, a_{j2})= FALSE \wedge ... \wedge P_i(a_i, a_{ji})=TRUE \wedge P_{i+1}(a_i, a_{ji+1})= FALSE \wedge ... \wedge P_n(a_i, a_{jn})= FALSE$, must be met.
- *ORJOIN*: Two or more alternative workflow branches re-converge to a single common activity as the next stop within the workflow. No synchronization is required because of no parallel activity execution. The condition between every predecessor activities $a_{i1},a_{i2},...,a_{ik}$ and successor activity a_j must satisfy there exists only one $a_{ij} \in \{a_{i1},a_{i2},...,a_{ik}\}$, such that $Completed(a_{ij})= TRUE \wedge P(a_{ij},a_j)=TRUE$.
- *LOOP*: A workflow cycle involves the repetitive execution of one (or more) workflow activities until a condition is met.

Definition 7 (*Control dependency*). For $\forall a_i,a_j \in A$, if a partial order $a_i \prec a_j$ or $a_j \prec a_i$ can be determined by predication *P*, controlled by the six control structures, there exists control dependency between activities a_i and a_j, denoted as $a_i \xrightarrow{P} a_j$.

Control dependency represents that there exists partial orders between activity notes *AN* according to prediction *P*, controlled by control nodes *CN*.

Control structures in workflow must meet the following rules. The three rules are used to guarantee completeness of process logic, so that deadlock can be eliminated (Chang, 2002).

Rule 1: An AND-Split control condition should have its matching AND-Join control condition.

Rule 2: An OR-Split control condition should have its matching OR-Join control condition.

Rule 3: A non-sequential control structure can be completely contained in another non-sequential control structure, but two non-sequential control structures should not be partially overlapped. Here, we say that two control structures are partially overlapped if an activity in one control structure is inside the other control structure.

The feature of workflow extracted from business component domain is showed by Figure 1.

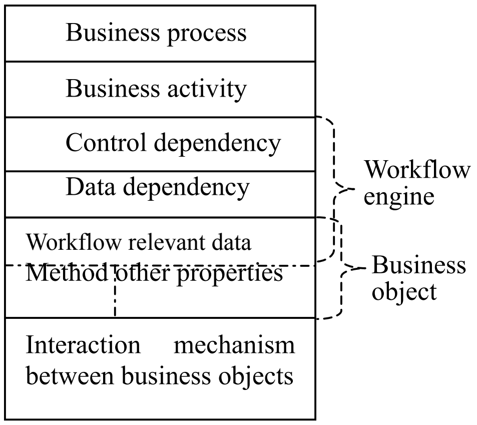

Figure 1. Reusable system supported by workflow

2.4 Divide reusable workflow engine component

To divide component, a common principle which must be followed is that one component should contain those business objects with a tight association (Katharine, 2002). So, to realize the characteristic above, reusable workflow engine contains the following components.

(1) **Condition management component.** Three kinds of conditions are involved in workflow engine, i.e. start condition of activity, completion condition of activity and transfer condition between activities. It is possible to probe the value of predications by a single component, so that specific users can modify predications to meet their specific requirement without inflecting other factors.

(2) **Control node component.** Every routing controls mentioned above section can be considered as one control node components. Common characteristics can be abstracted from these control nodes (Figure 2). At run-time, a control component is loaded into memory according to workflow definition and runs. After choosing routing and determining the next activity, the control component is destroyed from memory. Control component has two advances. Firstly, at build-time, to encapsulate control node can guarantee process completeness (guaranteed by rule 1, 2, 3). Secondly, at run-time, execution efficiency can be enhanced because relevant control node only run when needed for routing and cost can be cut.

(3) **Organization component.** Organization component supports changes of organization and role, determining participant of an activity and successive development of other components, such as resource management component. Workflow engine push business process by interacting among components.

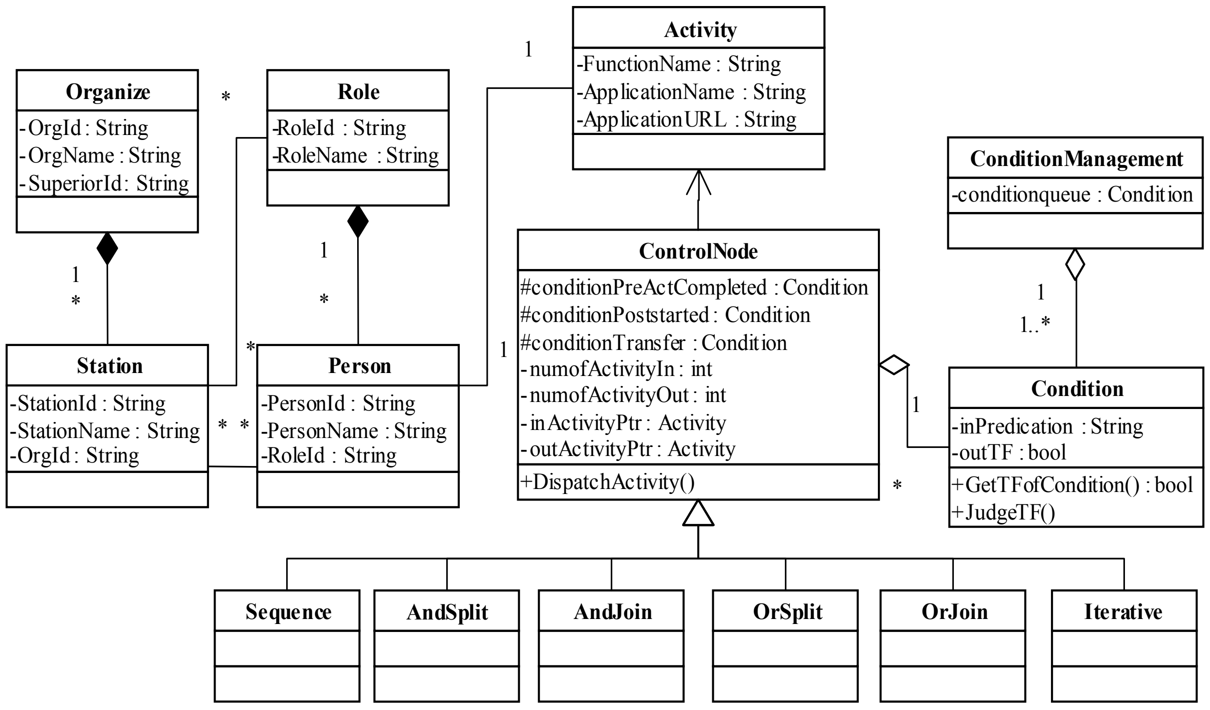

Figure 2. Class diagram of reusable workflow engine component

3 Composition of Workflow Engine Components

The composition of components means to compose their conceptual level, such as the input/output parameters, restrictions, etc. New component can be built by the connection between existing control nodes components.

(1) SEQUENCE structure

The composition of existing control node components can build a new component *C*, denoted as

C=<C_1,C_2,…,C_n>, where C_1,C_2,…,C_n must satisfy the following restrictions:

A. $OutActivity(C_i)$ = $InActivity(C_j)$,$i<j$, and there does not existing k, meeting $i<k<j$.

B. $C_i.numofActivityOut$ = $C_j.numofActivityIn$

After composition, Component C satisfies:

A. $InActivity(C)$=$InActivity(C_1)$

B. $OutActivity(C)$=$OutActivity(C_n)$

C. $C.numofActivityIn$ = $C.numofActivityOut$ = 1

(2) ANDSPLIT/ ANDJOIN structure

ANDSPLIT/ANDJOIN structure is a control type of 1 to many. The composition of this type can be denoted as C=<C_0,C_1,C_2,…,C_n,C'>, where C_0 is a component of ANDSPLIT type and C' is ANDJOIN. In C, C_i may be basic component or composition of components. This composition mode meets the rule 1, so completeness of control structure can be satisfied (Figure 3). C_0,C_1,C_2,…,C_n,C' must satisfy the following restrictions.

A. $OutActivity(C_0)$ = {$InActivity(C_1)$, $InActivity$ (C_2),…, $InActivity$ (C_n)}

B. $C_0.numofActivityOut$ = n

C. $OutActivity(C_i)$= $InActivity$ (C'),i=1,…,n

D. $C_i.numofActivityOut$ = 1

E. $C'.numofActivityIn$ = n

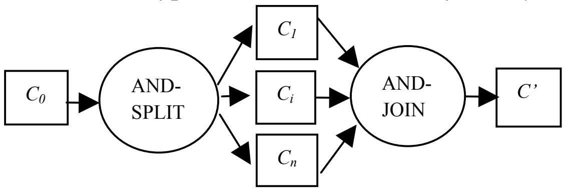

Figure 3. Composition of ANDSPLIT/ANDJOIN structure component

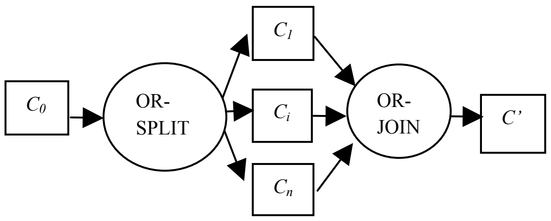

Figure 4. Composition of ORSPLIT/ORJOIN structure component

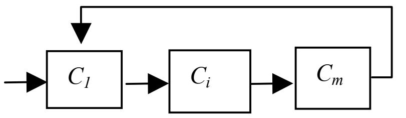

Figure 5. Composition of ITERATE structure component

After composition, Component C satisfies:

A. $InActivity(C)$=$InActivity(C_0)$

B. $OutActivity(C)$=$OutActivity(C')$

(1) ORSPLIT/ ORJOIN structure

ORSPLIT/ORJOIN structure is a control type of 1 to many also. The composition of this type can be denoted as C=<C_0,C_1,C_2,…,C_n,C'>, where C_0 is a component of ORSPLIT type and C' is ANDJOIN. In C, C_i may be basic component or composition of components. This composition mode meets the rule 2, so completeness of control structure can be satisfied. See Figure 4.C_0,C_1,C_2,…,C_n,C' must satisfy the following restrictions.

A. $OutActivity(C_0)$ = $InActivity(C_i)$,$InActivity(C_i)$ ∈{ $InActivity(C_1)$,$InActivity$ (C_2),…, $InActivity$ (C_n)}

B. $C_0.numofActivityOut$ = 1

C. ∃C_i, so that the predicate on P satisfies P=$TRUE$, which are labeled on the arc between C_0 and C_i .

D. $OutActivity(C_i)$= $InActivity$ (C'),i=1,…,n

E. $C_i.numofActivityOut$ = 1

F. $C'.numofActivityIn$ = 1

After composition, Component C satisfies.

A. *InActivity*(*C*)=*InActivity*(C_0)
B. OutActivity(*C*)=OutActivity(*C'*)

(2) ITERATE structure

A process is executed repeatedly until a condition is met (Figure 5). Component of ITERATE structure must satisfy the following restrictions.
A. O*utActivity*(C_m) = *InActivity* (C_1)
B. C_m.*numofActivityOut* = 1
C. C_1.*numofActivityIn* = 1
D. At a time, conditional predication satisfies *P=TRUE* after executing C_m.

After composition, Component *C* satisfies:
A. *InActivity*(*C*)=*InActivity*(C_0)
B. *OutActivity(C)=OutActivity(C')*

4 Reusability of Workflow Systems

The reusability of workflow system is categorized into reuse of workflow engine and that of business components. On one hand, to reuse a workflow engine, we designed and developed a set of abstract workflow engine components at first. To meet different requirements of enterprises to workflow engine, in future development stage, when developer begins to build a workflow engine, they only reuse existing workflow engine, or do some inheritable developments to meet specific requirement and characteristic of business process. These succeeding workflow engine components are all stored into workflow component repository.

The reuse mode of business component includes two aspects.

(1)**Process reuse.** This type of reuse can be divided into two types further, i.e. complete reuse and partial reuse. The former reuse a component through a complete copy, involving not only business, but also control structure of process. Partial reuse inherits existing business process partly. Some identifier can be used to reference existing relevant description of existing business process, while not copying the whole business process.

(2)**Structure reuse.** Developer only inherits existing structure of business process but business process. For the reason that the reusable workflow engine is mainly composed of six types of control nodes, and that these control node components can be reused to form a new engine, developer can inherit control structure of business process conveniently, then reconfigure the input and output parameters of control nodes and define new business process rapidly. Similarly, this kind of reuse can be divided into complete structure reuse and partial structure reuse.

Because data repository in ERP systems of existing enterprises are based on relational databases, two methods for component repository here are proposed when creating a new component by redefining an existing component. The first uses relational tables to store new component. By copying existing data and appending new to a table, a new component repository can be realized. The method has a disadvantage of data redundancy. The other method is to utilize user-defined language to describe component repository. The following language is an example.

```
<Component List>::=COMPONENT<Component id>
        REUSEDFROM <ProcessId>
        REUSEDTYPER < ALL| PART|ALLSTRUCT|
        PARTSTRUCT>
        [COMPONENTINHERITED< Component
        List,Component type >]
        COMPONENT < Component List ,Component type>
        END
```

In the definition above, “REUSEDFROM” denotes business process to reuse, “REUSEDTYPER” is type of reuse, “COMPONENTINHERITED” is component inherited when inheriting partly, “COMPONENT” is new component. Existing workflow engine component can be referenced though middle language by this method.

5 Application

The proposed approach has been applied to the development of CERP system in many enterprises of China and several projects of the National High-Tech Research and Development Plan of China. These applications are developed in our integrated framework (Figure 6). This framework has implemented software reconfigure effectively and enhanced the development efficiency of enterprise applications. Data for modeling an ERP system is inputted into modeling tool, called ERP Modeling which is developed by our team, as shown on the left part of Figure 6: (1) the modeling tool, i.e. ERP Modeling used to model applications and workflow systems; (2) model description exporting from ERP Modeling as XML format, including access control data, function items data and interface items data; (3) business component repository used to provide business component and to composite function and interface item components.

In middle part of Figure 6, workflow engine in this framework receives model data and is employed to schedule function item components, interface item components and different roles so that software systems can be executed accurately. Workflow Specification comes from modeling tool and provides information to workflow execution service. A set of workflow engine components in every similar domain developed stage is stored into workflow engine component repository.

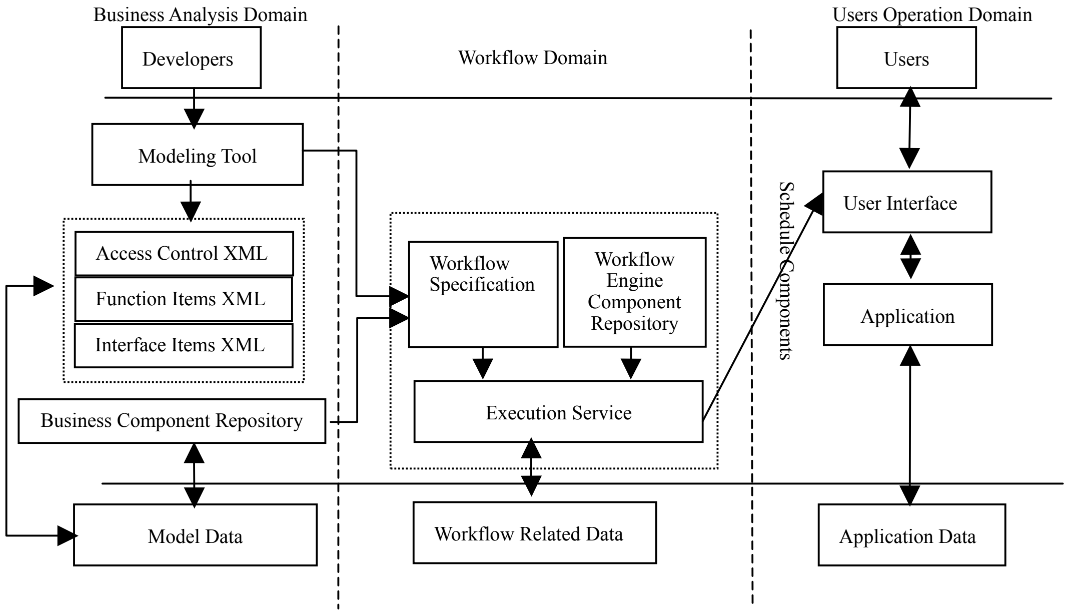

Figure 6. Component-based workflow development framework

6 Conclusion

Instead of developing every workflow management system from the ground up, it should be possible to come up with a generic and reusable set of functionality that provides the basic capabilities of a workflow engine. Its development effectiveness will keep on improving with the increasing number of the developed systems, especially in similar domains. Reliability of system based on this method could be guaranteed. This paper has focused on the development of such a reusable workflow engine. At the same time, the reusability of business component is also discussed.

Comparing with traditional workflow engine development approaches, reusable workflow engine has several advantages. (1) The architecture of reusable workflow engine is oriented from business component, so their connection is seamless. (2) Not only it is a black box reuse, but also users can understand its semantics through the process scenario. (3) Control node component runs only when routing control is needed between activities. This characteristic could save resource and enhance efficiency. (4) Development experiences can be inherited based on the development of reusable workflow engine.

The ongoing work is concerns two aspects. The first is to enrich workflow engine component repository through developing other business domains. This work can verify those engine components better. The second is to introduce knowledge management mechanism into business component repository, which is more meaningful work than the current one, in which we adopt model copy mode to reuse business component.

Acknowledgments

The authors also wish to acknowledge the financial support of the National High-Tech. R&D Program for

CIMS, China, Grant 2003AA4Z3210, 2003AA413023 and 2002AA413310.and another cooperative project from European Union.

Correspondence to:
Haibo Li
Centre of Intelligent Computing of Enterprises
School of Computer Science and Engineering
Harbin Institute of Technology
Harbin, Heilongjiang 150080, China
Working: School of Engineer
Northeast Agriculture University
Harbin, Heilongjiang 150030, China
Telephone: 01186-451-8641-2664
Email: lihaibo@hit.edu.cn

References

[1] A Frame Software. WorkMovr API Set Wverview, Available from http://www.a-froma.com /HTMDocs /PDF/APIset-pdf, 2001

[2] Sheth A, van der Aalst W, Arpinar I. Processes driving the networked economy, IEEE Concurrency, July-September, 1999:18-31.

[3] Dragon. A Micro-workflow: A Workflow Architecture Supporting Compositional Object-oriented Software Development. Ph.D. thesis, University of Illinois at Urbana-Champaign, 2001.

[4] Duk-Ho Chang, Jin Hyun Son, Myoung Ho Kim. Critical path identification in the context of a workflow. Information and Software Technology 2003;44:405-17.

[5] Drala Software. Drala Workflow Engine. Available from http://www.dralasoft.com/products/ workflow, 2001.

[6] Batory D, O'Malley S. The design and implementation of hierarchical software systems with reusable components, ACM Transactions on Software Engineering and Methodology. 1992;1(4):355-98.

[7] Hai Zhuge. Component-based workflow systems development. Decision Support Systems. 2003;35:517-36.

[8] Fujitsu Software Corporation. i-Flow Architecture White Paper,. Available from http://www.i-flow.com, 2001.

[9] Novak Jr. GS. Software reuse by specialization of generic procedures through views. IEEE Transactions on Software Engineering 1997;23(7):401-17.

[10] Puusjari J, Tirri H, Veijalainen J. Reusability and modularity in transactional workflows. Information Systems 1997;22(2/3):101-20.

[11] Katharine Whitehead. Component-based Development. Addison Wesley Longman, Inc. 1st edition ISBN: 0201675285, 2002.

[12] Hollingsworth D. Workflow Management Coalition: The Workflow Reference Model. Document Number WFMC-TC00 -1003, Brussels,1994.

[13] Lattanzi M, Henry S. Software reuse using C++ class, the question of inheritance, Journal of Systems and Software 1998;41:127-32.

[14] Mili R, Mili A, Mittermeir RT. Storing and retrieving software components: a refinement based system. IEEE Transactions on Software Engineering 1997;23(7):445-60.

[15] Vivtek, wftk: Open-source Workflow Toolkit. Available from http:// www.vivtek.com/wftk/, 2001.

[16] Rajlich V, Silva JH. Evolution and reuse of orthogonal architecture. IEEE Transactions on Software Engineering 1996;22(2):153-7.

[17] van der Aalst WMP, Barros AP, ter Hofstede AHM, Kiepuszewski B. Advanced workflow patterns. The Seventh International Conference on Cooperative Information Systems (CoopIS). 2000:18-29.