HTSN: A complex workflow model based on colored Petri net

Haiping Zhu*, Peigen Li, Guojun Zhang
School of Mechanical Science & Engineering, Huazhong University of Science &Technology,
Wuhan, Hubei 430074, P R China

ABSTRACT

Traditional workflow models have obvious shortcomings in describing complex workflows. Such complexity is due not only to the hierarchical property of business process, but also to the complicated dependencies among tasks. In this paper, we first introduce a conceptual task model based on UML class graph. Then we apply colored Petri nets to model task interfaces and composite task versions which form Task State Nets (TSNs) and Task Version Nets (TVNs). The main advantage is that they explicitly express all task interfaces' internal states and thus they can easily describe the complex control dependencies. Next we further combine TSN/TVN with place refining mechanism and bring forward our Hierarchical Task State Net (HTSN). It supports the description of workflow's hierarchical structure, multi-versions configuration of composite task as well as the analysis of structural properties. Finally, we give a case study.

Keywords: Complex workflow; Colored Petri net; Task State Net; Task version; Hierarchical Task State Net

1. INTRODUCTION

Workflow management systems have been adopted by a wide range of organizations to control their daily business processes. A specific workflow is the formal expression of a process that should pass through four phases: conceptual design, model definition, process configuration and execution, where model definition phase is vital to bring a natural process description to a computerized definition.

Due to its graphical and formal semantics, classic Petri net has been widely used for workflow modeling. Adam et al. [1] categorized several types of dependencies among workflow tasks. But he only studied on some simple dependencies with Petri net. In documents [3] [5] [6], Van der Aalst systemically employed Petri net to describe workflow in the control-flow perspective and defined Workflow Net. Also in his theory some key properties, such as liveness, boundness and soundness were identified to facilitate workflow evaluation and analysis. However, since Workflow Net is just one type of classic Petri net, it is inconvenient to describe complex systems

Other researchers preferred advanced Petri nets. Cho et al. [7] used colored Petri net (CPN) to study the transactional workflow. They proposed Task Net which can differentiate workflow versions and cases with token colors. Liu et al. [8] also studied the document review workflow with CPN. Their work was basically similar to common Workflow Net except that they used token color to identify document types and choose execution routes. Stork [9] applied hierarchical predicate/transition net to active document workflow and used structured token to carry various kinds of control information, which could be used for refinement of refinable place. But generality is lost and task states are neglected here.

Some approaches other than Petri net were also used to model and analyze workflow. Joeris et al. [11] bring object-oriented theory and ECA (Event-Condition-Action) rule [17] in workflow modeling. He also studied revise rules of different workflow versions [2]. Reichert et al. [12] combined common directed graph with text description to describe workflow's dynamic change and studied several common structural adjustment approaches. He also gave some algorithms to check whether structural adjustment would result in structural errors. The common shortcoming of the above mentioned approaches is that they are lack of strict mathematic theoretic basis.

In this paper, we introduce colored Petri nets into complex workflow modeling and apply Task State Nets (TSNs), Task Version Nets (TVNs) and Hierarchical Task State Nets (HTSNs) to model task interfaces, composite task versions and hierarchical workflows respectively. They explicitly express all task interfaces' internal states and can easily describe the complex control dependencies.

The remainder of this paper is organized as follows. In Section 2, we bring forward a workflow conceptual task model. In Section 3, we define TSN, TVN and HTSN, and discuss how to use them to describe task interfaces, control-flow dependencies among tasks and workflow's hierarchical structures. In Section 4, we give a case study on the complex workflow in engineering change process. In section 5, a conclusion is drawn.

2. COMPLEX WORKFLOW MODEL

Some enterprise processes are naturally complex. Consider the engineering change (EC) process in the PDM (Product Data Management) system. As we know, PDM system manages a lot of business objects including products, parts, documents and BOMs (Bill of Material), etc [18]. After the general review process, all these objects reach their release status when any direct updates are denied. However, changes are inevitable. To maintain effective control over changes and guarantee the information consistency, complex EC workflows should be designed in PDM system.

The complexities mainly show in the following aspects:

(1) Workflow's hierarchy coming from the hierarchy of business goals and organization structures [4].

(2) Complex and dynamic dependencies. Dynamic dependencies result from business process reengineering or some unusual variations, such as adding of business object types. Except the ad hoc changes, each variation generates a new task version for certain level and these versions enjoy different application scope [2].

(3) Semi-precise. Complex workflow does not strictly differentiate between definition phase and execution phase. On one hand, imprecise details at lower level can be hidden by abstraction at higher level; on the other hand, in execution phase tasks at each level can be precisely parsed level to level through task version configuration and dynamic adjustment of task internal structure.

All these complexities increase more requirements to workflow research. Firstly, semi-precise workflow design and modeling theory should be built. Secondly, the model should possess the capability to configure tasks to improve flexibility and customization property. The key idea is to construct imprecise generic model which can be dynamically configured into precise workflow models

* Corresponding author haipzhu@263.net

according to several predefined parameters. Thus, a generic model could represent a variety form of workflows. Thirdly, the model should support verification analysis and running simulation. Finally, the workflow should be flexibly executed, that is, allowing adding/removing a task, skipping a task, changing the dependencies among tasks, etc.

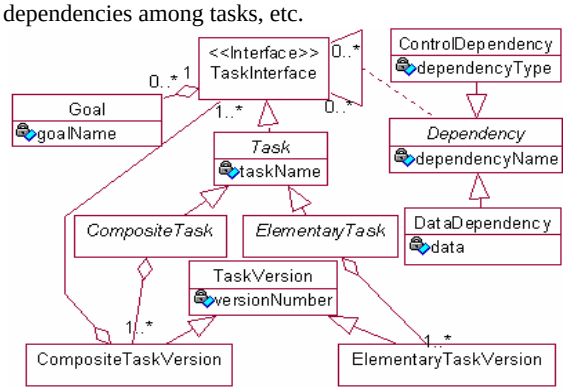

Fig.1: UML description of conceptual task model.

Fig.1 illustrates the conceptual task model introduced by author. This UML class graph includes several classes and interfaces, representing some important concepts in complex workflow. Detailed explanations are given later.

Goals [4] show why a workflow is designed and what functions it needs to provide. They can be classified into functional goals and non-functional goals. We often regard a big business flow as a root goal that can be divided into several more concrete parts. Also certain parts can be divided further. In this top-down dividing process, some parent-goals may be divided according to different approaches resulting in different sub-goal collections. This structure is expressed as a goal tree.

Def. 1 (Goal Tree). Goal tree is a 5-tuple $(G, r, necessity, option, choice)$ where G represents a finite set of goals, including all parent-goals and sub-goals; $r \in G$ represents the root goal; $necessity \in G \times G$ is a set of all necessary parent-child relationships such that for any two goals p and c, if $(p, c) \in necessity$, then realization of child-goal c is necessary to realization of parent-goal p; $option \in G \times G$ is a set of all optional parent-child relationships such that for any two goals p and c, if $(p, c) \in option$, then realization of child-goal c is optional to realization of parent-goal p; $choice \in G \times G... \times G$ is a set of all choice relationships such that for any $n+1$ goals p and $c_1,...,c_n$, if $(p, c_1, ..., c_n) \in choice$, then one and only one c_i $(1 \leq i \leqslant n)$ is the child-goal of parent-goal p.

Each functional goal node in the goal tree corresponds to a task implementation. During workflow instance's execution, each task case will pass through a series of states. In Fig. 1, task is represented as an abstract class '*Task*', which derives two abstract subclasses: *ElementaryTask* and *CompisiteTask*.

Def. 2 (Elementary Task). Elementary task is a type of task without detail structure. Its state set is S_{ET} such that $S_{ET} = \{inactive, active, skipped, running, completed, aborted\}$.

Def. 3 (Composite Task). Composite task is a type of task being composed of a series of tasks. The state s of a composite task is represented as $s = s_0 + \Sigma s_{cti}$ where $s_0 \in S_{ET}$ represents the state of the whole composite task while Σs_{cti} represents the state combination of all component sub tasks.

The dependencies among sub tasks of a composite task can be classified into external dependencies, control-flow dependencies and data-flow dependencies according to their properties. Since external dependencies are usually determined by external factors such as organization constraints and time constraints that are independent of workflow, they are beyond this paper.

At most time, we need not care about details of subtasks within a composite task. So we use interfaces to realize abstraction. Interface is a concept initially introduced in object-oriented software design theory. It declares some method definitions while does not realize them so that many different classes can realize the same interface to achieve polymorphous. The task interface acts in the same way except that what it declares are goals and required dependencies instead of method definitions.

Def. 4 (Task Interface). Task interface TI is a 3-tuple (G, CD, DD) where G represents the set of goals to be achieved, CD represents the set of control-flow dependencies to be realized and DD represents the set of data-flow dependencies to be realized.

Therefore, the dependency source and destination are both task interfaces instead of tasks. As shown in Fig.1, *TaskInterface* implemented by an abstract class '*Task*' aggregates one or multiple goals. An abstract association class '*Dependency*', which expresses dependencies among task interfaces, derives two subclasses: *ControlDependency* class and *DataDependency* class.

Since each task interface does not detail its internal structure, it can be implemented in different ways resulting in different task versions. In Fig.1, an abstract class '*TaskVersion*' derives two subclasses: *CompositeTaskVersion* class and *ElementaryTaskVersion* class representing composite task version and elementary task version respectively. In general, there are two main different forms of task versions: one is the build-time version occurring in the workflow design phase if different realizing approaches to the same goal exist. The other is the run-time version generated in the running phase when structural adjustments such as adding a task occur and these changes are permanent.

Due to the existence of multiple versions, version configuration rules are highly necessary. There are several commonly used rules. 1) Automatically choosing the latest version. It's used in the business process reengineering to immediately apply the change to all following workflow instances; 2) Always using the same version during an instance's lifecycle even if another instance generates a new version; 3) Choosing task version according to the property of workflow instances. For example, in document workflow, different documents may choose different processes according to document types which could be defined as version configuration parameters; 4) Choosing task version according to the time. That is, some task versions are only useful in certain time period or before/after certain time points.

3. HTSN OF COMPLEX WORKFLOW

In this section, we propose HTSN which is a complex workflow model based on hierarchical colored Petri net and embodies the idea of conceptual task model illustrated in Fig. 1. As mentioned above, none of the current Petri-net-based workflow models can explicitly express

task state transitions resulting in the unidentifiability of task states. Therefore, some complex dependencies, such as the soft-synchronization dependency [1] and the multi-choice synchronizing merge dependency [13] are difficult to express. Moreover, classic Petri nets do not support modularity, and thus they can not realize version configuration. We instead apply colored Petri net [15] to describe task interfaces, task dependencies and task versions. First we introduce Task State Net.

Task State Net

Fig. 2 depicts a task interface in two different perspectives and forms Task State Net which is a specific type of CPN. The external interface hides its internal details and acts like a transition. Its dependencies with other interfaces are expressed as various places. The internal structure describes the state transition process of internal states of interface under the external control dependencies and data dependencies in detail. As shown in Fig. 2(2), when initialized, an inactive instance can either be skipped to state *skipped* or be started to state *active*. If activated, such instance will appear in the task box of a certain user. So this user can select to open it and make it enter into state *running*, and then make a decision in three choices, i.e., either completing it, or aborting it, or making it back to state *active*.

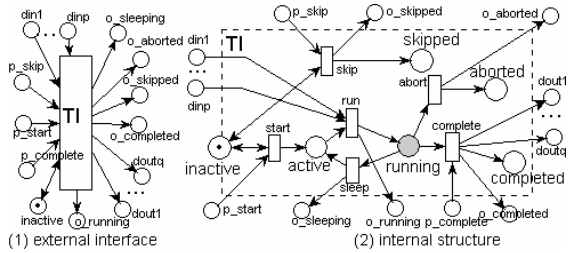

Fig. 2: Task State Net.

Def. 5. Task State Net (TSN) describes a task interface TI in a 6-tuple (Σ, P, T, A, E, I) where:

(1) Σ is a finite set of color sets including the following type of colors:

$color\ WID,\ EID\ =\ int;\ \ color\ DA\ =\ Object;$

$color\ TS\ =\ WID \times EID;\ \ color\ TD\ =\ WID \times DA;$

WID is a unique positive integer type identifier to differentiate the different workflow instances. *EID* is a unique positive integer type identifier to differentiate the different execution times of a specific task in current workflow instance. *DA* stands for all data objects. *TS* is the product set of *WID* and *EID*. *TD* is the product set of *WID* and *DA*.

(2) P is a finite set of places such that:

- $P \subseteq SP \cup DP \cup CIP \cup COP$, where:

$SP = \{inactive, active, skipped, running, completed, aborted\}$ are places representing task states;

$DP = DI \cup DO,\ DI = \{din1, ..., dinp\}$ and $DO = \{dout1, ..., doutq\}$ are several places representing inputting and outputting data dependencies respectively;

$CIP = \{p\_start, p\_skip, p\_complete\}$ are places representing inputting control dependencies.

$COP = \{o\_running, o\_skipped, o\_completed, o\_sleeping, o\_aborted\}$ are places representing outputting control dependencies.

- The token color in each place satisfies:

$$Color(p) = \begin{cases} TD & p \in DP \\ TS & p \in SP \\ WID & p \in CIP \cup COP \end{cases}.$$

(3) T is a finite set of transitions such that $T \subseteq \{skip, start, run, sleep, complete, abort\}$ which represent all task state transitions.

(4) A is a finite set of arcs such that $A \subset P \times T \cup T \times P$ and $\forall a \in A,\ a\ =\ (S,\ D)$.

(5) E is an arc expression function defined on A:

$var\ wid : WID;\ \ var\ eid : EID;\ \ var\ da : DA;$

$$E(a) = \begin{cases} 1'(wid, eid) & a.S \in SP \wedge a.D \in SP \\ 1'wid & a.S \in CIP \vee a.D \in COP \\ 1'(wid, da) & a.S \in DP \vee a.D \in DP \end{cases}.$$

(6) I is an initialization function defined on P such that:

$$I(p) = \begin{cases} 1'(wid, 1) & p = inactive \\ \varnothing & otherwise \end{cases}.$$

Description of dependencies among TSNs

In workflow system, a composite task contains multiple task versions that are composed of multiple task interfaces. There are various complex control-flow dependencies and data-flow dependencies among these interfaces. In brief, a control-flow dependency shows the mapping from the output control *cop1* of task interface *TI1* to the input control *cip2* of interface *TI2*. The arc expression of the corresponding CPN graph depicted in Fig. 3(1) is $E(a) = 1'wid$ such that $a \in \{(TI1, cop1), (cip2, TI2)\}$. The semantic indicates that *TI2*'s internal state transition can fire only when a token with color *wid* is produced into place *cop1/cip2*. For example, Fig. 3(2) represents the common completed->start dependency. After *TI1* fires its complete transition, it will reach its *completed* state when a token *(1000,1)* is produced into place *completed* and a token *(1000)* is produced into place *o_completed*. Since *o_completed* is also the input place *p_start* of transition *start* in *TI2* and at that time, *TI2*'s place *inactive* has a token *(1000,1)* which is generated at initialization time, *TI2*'s transition *start* is enabled.

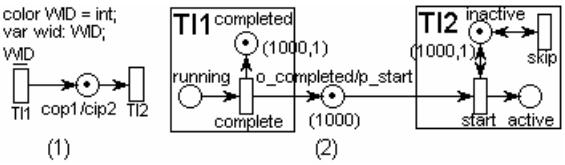

Fig. 3: Control-flow dependency.

Some complex control-flow dependencies can be constructed through *and/xor* operations. Fig. 4 shows four complex dependencies: AND-Split dependency, XOR-Split dependency, AND-Join dependency and XOR-Join dependency. For simplicity, we represent each task interface as a single transition.

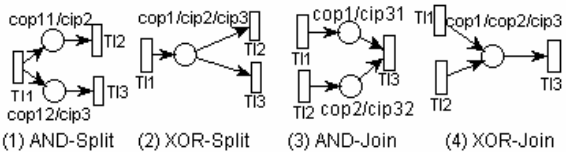

Fig. 4: Four complex dependencies.

Workflow patterns are the basic workflow control structures abstracted from practical business processes and being independent of specific modeling language realization [13]. In fact, each workflow pattern is a combination of multiple control-flow dependencies. CPN descriptions of several common workflow patterns are showed below. (*TI1, TI2* and *TI3* as three task interfaces.)

(1) Sequence pattern. It means *TI1* and *TI2* are executed or skipped sequentially. The corresponding CPN is illustrated in Fig. 5(1). *TI1* can either be completed or skipped, if it is completed, then its place *o_completed* obtains a token which enables *TI2's* transition *start*, and if it is skipped, then its place *o_skipped* obtains a token which enables *TI2's* transition *skip*.

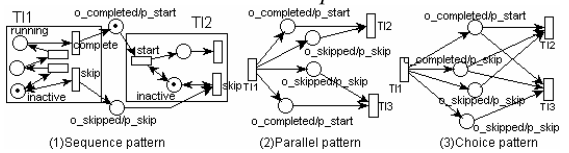

Fig. 5: Sequence, parallel and choice pattern.

(2) Parallel pattern. It means when *TI1* is completed, both *TI2*'s and *TI3*'s places *p_start* obtain a token, so both *TI2's* and *TI3's* transitions *start* are enabled; and if *TI1* is skipped, both *TI2's* and *TI3's* transitions *skip* are enabled. Its CPN is shown in Fig. 5(2).

(3) Choice pattern. It means when *TI1* is skipped, both *TI2* and *TI3* are skipped; when *TI1* is completed, *TI2* or *TI3* is started; if *TI2* is started then *TI3* is skipped, and vice versa, i.e., they can not both be started. Its CPN is shown in Fig. 5(3).

(4) Synchronization pattern. It means *TI3* can be started only if both *TI1* and *TI2* are completed and be skipped only if both *TI*1 and *TI2* are skipped. Its CPN is shown in Fig. 6(1).

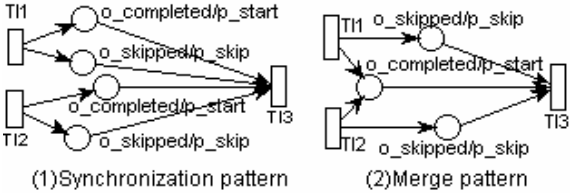

Fig. 6: Synchronization pattern and merge pattern.

(5) Merge pattern. It means *TI3* can be started if either *TI1* or *TI2* is completed and be skipped only if both *TI*1 and *TI2* are skipped. Its CPN is shown in Fig. 6(2).

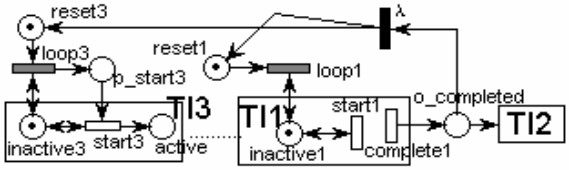

Fig. 7: CPN description of cycle pattern.

(6) Cycle pattern. It means when *TI1* is completed, either *TI2* is started or *TI3* before *TI1* is executed again. As shown in Fig. 7, one transition λ, several places *reset* and several transitions *loop* (from *TI3* to *TI1*) are inserted. If transition λ is fired, each place *reset* will obtain one token *wid*, which enables each transition *loop*. We denote the input and output arc expressions of *loop* as following:

$$E(a)=\begin{cases}1'wid & a\in\{(reset,loop),(loop,p\_start)\}\\ 1'(wid,i) & a=(inactive,loop)\\ 1'(wid,i+1) & a=(loop,inactive)\end{cases}.$$

So, *TI3* will soon change to state *active* and its execution time will automatically plus one.

(7) Soft-synchronization pattern. It means *TI2* can be started when *TI1* is either completed or skipped.

(8) Multi-choice synchronizing merge pattern. There are *n* branches (*TI_1*~*TI_n*) after *TI_start* and possibly *m* (*m*≤*n*) branches will be chosen for running. These *m* branches need to be synchronized before *TI_end* can be started. Its CPN description is shown in Fig. 8.

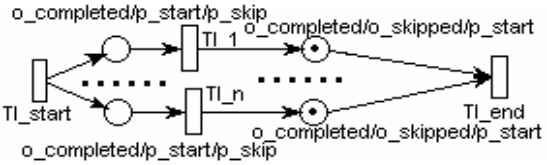

Fig. 8: Multi-choice synchronizing merge pattern.

Expression of data flow dependency

Data-flow dependencies reflect the data objects providing/using relationships. The CPN description is similar to that of control-flow dependency in Fig. 3(1) except that the connective place is not *cout1/cin2* but *dout1/din2* and its token color is not *WID* but *TD (TD=WID×DA)*.

CPN definition of composite task version

Each composite task version is made up of lots of task interfaces interconnected by several control-flow and data-flow dependencies. On the basis of TSN and CPN descriptions of dependencies, we give the following formal definition of Task Version Net.

Def. 6. A composite Task Version Net (TVN) is a CPN with the following characteristics:

(1) It contains *n (n≥1)* TSNs (task interfaces) which have the CPN structures in Fig. 2;

(2) There is one unique entry TSN whose execution indicates the beginning of this TVN, and one unique exit TSN whose completion indicates the finish of this TVN;

(3) The various dependencies among TSNs are represented by the CPN structures as shown in Fig. 3 and grouped by corresponding workflow patterns;

(4) During the initialization time, each TSN's place *inactive* will obtain a colored token (*wid, eid)*.

Hierarchical Task State Net description of workflow

In order to fulfill the mapping from one task interface to several composite task versions, we introduce the place refining mechanism. In Fig. 2(2), if *TI* is a composite task interface, its *running* place with gray background color is a refinable place that can be refined to certain task version as the region enclosed by the dash line circle in Fig. 9. The same refinable place can be refined to different structures which correspond to different task versions and such refining process can be done recursively. That is, all the places *running* of TI1~TIn in Fig. 9 can also be refinable and have their lower level structures. The following is the formal definition of Hierarchical Task State Net (HTSN).

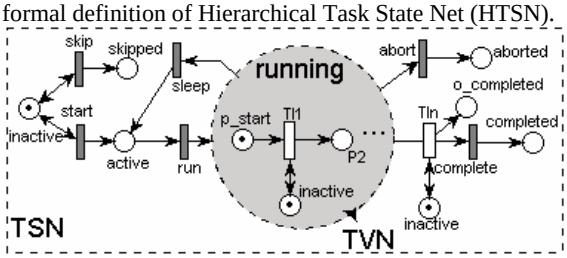

Fig. 9: CPN description of TSN's mapping to TVN.

Def. 7. HTSN is a 5-tuple (*TSN*, *TVN*, *RP*, *TVA*, *R*) satisfying the following requirements:

(1) *TSN* is a finite set: $TSN = CTSN \cup ETSN$ which includes all composite task interfaces (*CTSN*) and all elementary task interfaces (*ETSN*);

(2) *TVN* is a finite set of task versions: $TVN = \{TVN_1, ...TVN_n\}$ which includes all composite task versions at all workflow levels;

(3) Other elements are relative to place refinement:

- *RP* is a finite set of refinable places including all places *running* of the *CTSN*;

- $TVA: CTSN \rightarrow TVN$ is a task version configuration function. There is a one to several mapping from *CTSN* and *TVN*, i.e., one place *running* may be refined to multiple different task versions. Each kind of mapping corresponds to a certain rule. *R* represents such a set of configuration rules such that $R: (CTSN \times TVN) \rightarrow Boolean$.

Execution and structural analysis of HTSN

The execution of a workflow instance is just a parsing process of HTSN. As shown in Fig. 9, when a new workflow instance arrives, a new token *(wid, 1)* will be generated into the place *inactive* of the root task interface *TSN0*. Then, *TSN0* may reach its running state. If place *running* is a refinable place, then a certain task version or *TVN* must be chosen for execution according to the version configuration rule *R*. This brings the instance to the second level. At that time, all *TSNs* of the chosen *TVN* are initialized firstly (one *(wid, 1)* token is generated into each *inactive* place) and the place *p_start* of the entry TSN in this *TVN* (*TI1* in Fig. 9) obtains a token which enables its transition *start*. Next, *TSNs* are executed in an order decided by those dependencies. And similarly, during the execution of *TSNs* at the second level, multiple refinable places might also exist. Therefore, the refining process will be done recursively until a level is reached where all task interfaces are elementary and their running places are no longer refinable.

One advantage of this hierarchical structure is that it can be dynamically adjusted. Traditional workflow model regards a workflow instance as one running of the model. So, ever since the instance is initialized, the running route is totally determined. Any change of this model can only be reflected in new instance instead of the current running instance. Instance transferring always fails to fulfill due to the state space change. On the contrary, a change to the instance with a hierarchical structure affects a much smaller scope. When a task version changes, a new task version will be generated and the configuration rules will be revised. As a result, when current instance loops to or other instances reach this interface, such new version could be chosen for running.

Another advantage of HTSN is that the workflow structure can be maintained well by verifying the *TVNs'* structure. For the task version net: *TVN=(TSN0,…,TSNn)* where *TSN0* stands for the entry Task State Net and *TSNn* stands for the exit Task State Net, we denote its initial marking *Mi* and end marking *Mo* of the *eid*-th execution as follows: if every place inactive of *TSN* satisfies *M(inactive)=(wid,eid)* and any other place satisfies *M(p)=empty*, then the current making is Mi; if either place completed or skipped or aborted of *TSNn* satisfy *M(p)=(wid,eid)* and no transitions are enabled any more, then the current making is *Mo*. Just as Workflow Net [16], the following soundness property must be satisfied in order to ensure *TVN's* liveness and deadlock free.

(1) For each making *M* reached from M_i, there is a firing sequence which leads from state *M* to M_0 and no deadlock exists, i.e., $\forall M(M_i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*'} M_o)$.

(2) At the end making M_o, each *TSN* is in either *skipped* or *completed* or *aborted* state, i.e., $\forall TSN_i, \quad M(completed_i) \supseteq (wid, eid) \vee M(skipped_i) \supseteq (wid, eid) \vee M(aborted_i) \supseteq (wid, eid)$, and no tokens exist in any place *active* and *running*, i.e., $\forall TSN_i, \ M(active_i) = M(running_i) = empty$. That is, every TSN has been executed at least once and normally terminated.

There are generally two approaches to analyze the *soundness* attribute. One is the CPN mathematic analysis theory and the other is the simulation method [15].

4. CASE STUDY

To further explain the concepts of complex workflow, we provide an example of engineering change (EC) process for a large manufacturing enterprise in this section. As mentioned in Section 2.1, no doubt EC process is a representative complex workflow with hierarchical structures and complex dependencies.

Firstly, the goal tree of a simplified EC process is constructed in the conceptual design phase. As shown in Fig. 10, it contains a number of goal nodes whose three parent-child relationships, i.e., necessary relation, optional relation and choice relation are depicted by different arrowheads. The following lists these elements.

$G = \{g0, g1, g2, g3, ..., g18\}, \quad r = g0,$

$necessity = \{(g1, g3), (g1, g4), (g2, g6), (g2, g7), (g2, g8), (g7, g9), (g7, g11), (g8, g12), (g10, g15), (g10, g18)\},$

$option = \{(g2, g5), (g7, g10), (g10, g16), (g10, g17)\},$

$choice = \{(g0, g1, g2), (g8, g13, g14)\}$

At the top level, due to the different influencing degree of change, the root goal *EC workflow* has two choice child goals: *simple change* and *complex change*. The former is applied to urgent change while the later is applied to complex but less urgent change. For the goal *complex change*, its child goal *issue description* is optional while the other three child goals: *change request*, *change evaluation* and *change activity* are all necessary. Similarly, the goal *change analysis* has two necessary children: *issue classification* and *analysis summary*, and two optional children: *design analysis* and *process analysis*.

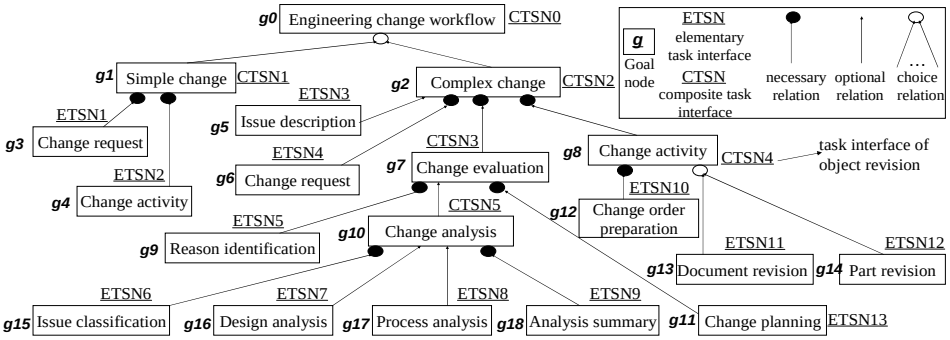

Fig. 10: Goal decomposition tree of engineering change workflow

Secondly, in the model definition phase, this goal tree should be mapped to hierarchical structures where the leaf goal nodes such as *issue classification* are mapped to elementary task interfaces while the intermediate goal nodes such as *change evaluation* are mapped to composite task interfaces. Fig. 10 depicts all task interfaces/TSNs.

$TSN = CTSN \cup ETSN,\ CTSN = \{CTSN0, ..., CTSN5\},$

$ETSN = \{ETSN1, ..., ETSN13\}$

On the one hand, each task interface must realize its goal. For example, *CTSN4*, which means the task interface of object (part or document) revision, must realize the goal *g8 (change activity)*. On the other hand, a task interface may be implemented by different task versions. For example, task interface *CTSN3* has two task versions: one is composed of two steps of *issue identification* and *change planning*; the other is composed of three steps of *issue identification*, *change analysis* and *change planning*. As an example, we use some TSNs to describe the task version *TVN0* which realizes the goal *g8 (change analysis)* and implement *CTSN5*. Its requirements and internal dependencies are listed in Table 1. Fig. 11 shows the CPN description of this task version. Each dash-line rectangle represents the task interface in the form of TSN. The dependencies between TSNs are described by several places.

Thirdly, we identify the task versions of all composite task interfaces and describe them in the form of HSTN. The details are shown in Table 2.

Table 1. Requirements and internal dependencies of *TVN0*.

Requirements	*TVN0* should contain four TSNs: *ETSN6*, *ETSN7*, *ETSN8* and *ETSN9* which represent the task interfaces implementing goals *issue classification*, *design analysis*, *process analysis* and *analysis summary* respectively.
	Since *g16 (design analysis)* and *g17 (process analysis)* are optional goals, *ETSN7* and *ETSN8* can be skipped while *ETSN6* and *ETSN9* can not.
Dependencies	If and only if *ETSN6* is completed, then *ETSN7* and *ETSN8* can be started simultaneously, but *ETSN7* and *ETSN8* can also be skipped.
	If *ETSN7* is running or skipped and *ETSN8* is running or skipped, then *ETSN9* can be started.
	If *ETSN9* is completed, then both *ETSN6* and *ETSN7* must be completed (if they are running).
	ETSN7's and *ETSN8*'s running need read data object *o1* generated when *ETSN6* is completed.
	When *ETSN9* is completed, a data object *o2* will be generated.

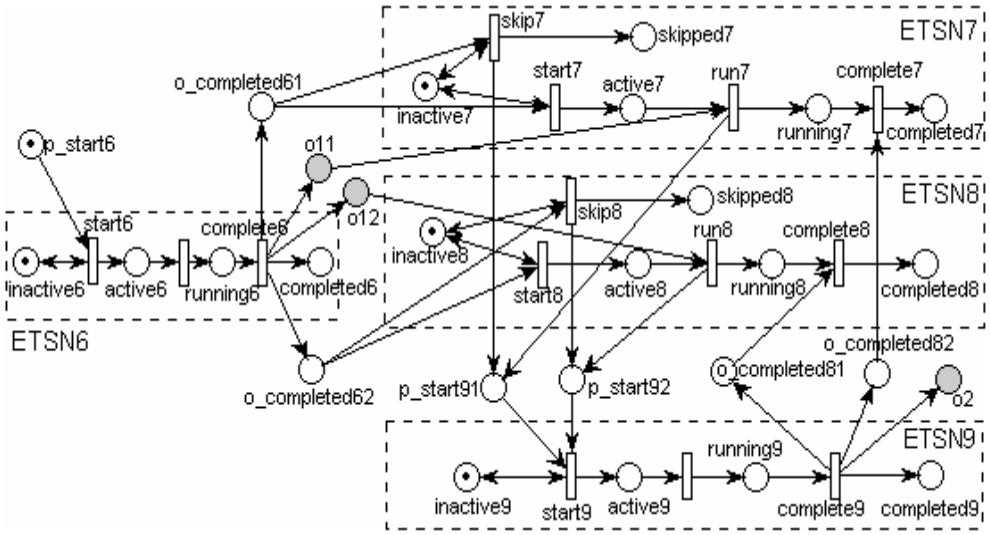

Fig. 11: CPN description of task version TVN0.

Table 2. HSTN description of the EC workflow.

CTSN	TVN	TSN	TVA	R: Version configuration rule
CTSN0	*TVN01*	*CTSN1*	*(CTSN0, TVN01)*	True if change is simple.
	TVN02	*CTSN2*	*(CTSN0, TVN02)*	True if change is complex.
CTSN1	*TVN11*	*ETSN1, ETSN2*	*(CTSN1, TVN11)*	Always true.
CTSN2	*TVN21*	*ETSN3, ETSN4, CTSN3, CTSN4*	*(CTSN2, TVN21)*	True if detail issue description is needed.
	TVN22	*ETSN4, CTSN3, CTSN4*	*(CTSN2, TVN22)*	True if issue description is not needed.
CTSN3	*TVN31*	*ETSN5, CTSN5, ETSN13*	*(CTSN3, TVN31)*	True if detail change analysis is needed.
	TVN32	*ETSN5, ETSN13*	*(CTSN3, TVN32)*	True if change analysis is not needed.
CTSN4	*TVN41*	*ETSN10, ETSN11*	*(CTSN4, TVN41)*	True if the change object is a document.
	TVN42	*ETSN10, ETSN12*	*(CTSN4, TVN42)*	True if the change object is a part.
CTSN5	*TVN51*	*ETSN6, ETSN7, ETSN8, ETSN9*	*(CTSN5, TVN51)*	Always true.

Finally, before this HTSN model takes effect, we can simulate its execution to find structural errors. Generally, each task versions should be verified. Fig. 12 illustrates our simulation tool: HTSN Modeler and Analyzer which is based on the simulating theories of colored Petri net and is implemented by Java technology. From the simulation result of reachable states analysis, we can see that no structural errors exist in the task version *TVN0*. Similarly, we can verify all other TVNs.

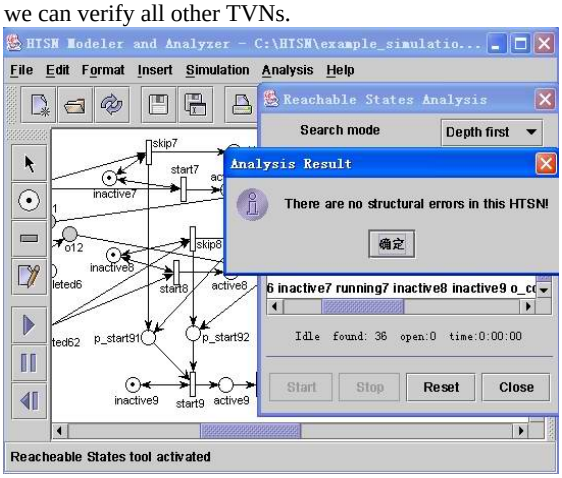

Fig. 12: Simulation result of HTSN Modeler and Analyzer

5. SUMMARY

Workflow model should have stronger description ability since workflow is used more widely and the business environments are increasingly complex. Although traditional Petri-net-based models are supported by matured theory, neither can they explicitly represent task states so as to express complex control dependencies, nor can they describe hierarchical structure so as to support task version configuration. Therefore, they are not suitable for describing complex workflow.

Based on the complex workflow conceptual task model, we use colored Petri net to describe the task interfaces, dependencies among tasks and composite task versions, and put forward HTSN of workflow. Compared with other approaches, the HTSN model has the following advantages:

(1) *Hierarchical description ability*. It supports workflow's hierarchical structure and multiple versions configuration. It enables adding of new versions and changing of version configuration rules freely and dynamically so it gives the model more flexibility.

(2) *Ability to express complex control dependencies*. Task State Net can explicitly express the different task states such as inactive, running, skipped and completed, so it is suitable for describing complex control flow dependencies.

(3) *Verification analysis and simulation.* There are already some matured CPN analyses and running simulation tools to verify the workflow model to find structural errors before its execution, while other non-Petri-net-based workflow modeling approaches are lacking of this ability.

ACKNOWLEDGEMENTS

This research was supported by the National Science Foundation of P R China under grant 50275056.

References

[1] Nabil R.Adam, Vijayalakshmi Atluri, Wei-kuang Huang. Modeling and Analysis of Workflows Using Petri Nets. Journal of Intelligent Information Systems, 10, 131-158(1998).

[2] Gregor Joeris, Otthein Herzog. Managing Evolving Workflow Specifications. 3th IFCIS International Conference on Cooperative Information Systems, New York, Aug.1998, pp. 310-319.

[3] W.M.P. van der Aalst. The application of Petri nets to workflow management, The Journal of Circuits Systems and Computers 8(1), 1998, pp. 21-66.

[4] Kueng, Peter; Kawalek, Peter. Goal-based business process models: creation and evaluation. Business Process Management Journal, Vol. 3, No. 1 (April 1997), pp. 17-38.

[5] W.M.P van der Aalst, T. Basten. Inheritance of workflows: an approach to tackling problems related to change. Theretical Computer Science 270(2002)125-203.

[6] W.M.P. van der Aalst. On the automatic generation of workflow processes based on product structures. Computers in Industry 39(1999)97-111.

[7] Injun Choi, Chulsoon Park, Changwoo Lee. Task net: Transactional workflow model based on colored Petri net. European Journal of Operational Research 136(2002)383-402.

[8] Dongsheng Liu, Jiangmin Wang, Stephen C.F.Chan, Jiaguang Sun, Li Zhang. Modeling workflow processes with colored Petri nets. Computers in Industry 49(2002)267-281.

[9] Stork, David G., van Glabbeek, Rob. Token-Controlled Place Refinement in Hierarchical Petri Nets with Application to Active Document Workflow. 23rd International Conference on Applications and Theory of Petri Nets, Adelaide, Australia, June 24-30, 2002, pages 1-394pp.

[10] Workflow M C. The workflow reference model. [WfMC1003] [R]. WFMC TC-003, Jan. 1995.

[11] Gregor Joeris, Define Flexble Workflow Execution Behaviors, TZI Technical Report 15-1999, Center for Computing Technologies (TZI), University of Bremen, 1999.

[12] Reichert, M, Dadam, P. "ADEPTflex-Supporting Dynamic Changes of Workflows Without Losing Control", Journal of Intelligent Information Systems-Special Issue on Workflow Management, 10(2), Kluwer Academic Publishers, March 1998; pp.93-129.

[13] W.M.P. van der Aalst, A.P.Barros. Advanced Workflow Patterns. Fifth IFCIS International Conference on Coopertive Information Systems,volume 1901 of Lecture Notes in Computer Science, pages 18-29.

[14] Wasim Sadiq, Mariae E.Orlowska. Analyzing Process Models Using Graph Reduction Techniques. Information Systems Vol.25, No, 2, pp, 117-134, 2000.

[15] K.Jensen. Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use-Volume I: Basic Concepts. EATCS Monographs in Computer Science, Vol.26, Springer-Verlag(1992).

[16] W.M.P.van der Aalst, Arthur H.M.Ter Hofstede. Verification of Workflow Task Structures: A Petri-Net-Based Approach.Information Systems, Vol.25, NO.1, pp.43-69, 2000.

[17] A.Goh, Y.-K.Koh, D.S.Domazet. ECA rule-based support for workflows. Artificial Intelligence in Engineering, 15(2001) 37-46.

[18] Product Data Management: The definition, CIMdata, http://www.cimdata.com, 1997.