

Modeling and Analysis of Workflow Based on TLA

CHEN Shu

Wu Han University Computer Science Department, Wu Han China
 Email: Chenshu181@yahoo.com.cn

WU Guo Qing

Wu Han University Computer Science Department, Wu Han China
 Email: wgq@whu.edu.cn

Abstract— We proposed an approach in modeling and analysis of workflow based on temporal logic of action. A workflow model is divided in to two parts: the description of scheduler as well as database updating and the description of properties, each part is expressed in a TLA formula. The analysis of properties is composed by scheduling analysis and database properties analysis and their proof process is equivalence to the verification of the implementation relationship between the model formula and the formula of its properties. Thus, established an unified framework for modeling and property analysis of workflow in deferent levels.

Index Terms— workflow, TLA, modeling, property analysis

I. INTRODUCTION

A workflow is a collection of activities that designed for the purpose of carrying out a target process, while a workflow model is a formalized description for the abstract process as well as their dependencies[1]. Workflow technology is widely used to manage services composition, business-to-business e-Commerce process integration and e-Science application. A workflow model consists of several components, including activities, controllers, database, roles etc. Thus a workflow model stands on the base of its running. Besides, the property analysis of a workflow model is becoming a critical step. However, the workflow property analysis needs the workflow be modeled in a formalized way and with a well-defined semantics. The modeling of a workflow would concern two layers, control layer and database layer. Control layer focuses on the scheduling of the workflow activities, while the underlying database layer reflects the update of the data states. An intact framework of a workflow consists of a control graph, triggers and properties. Control graph as showed in Figure 1 provides a global view for the dependencies of the activities. A typical control graph consists of a initial state, a set of termination states, a set of activity states, and a set of directed edges that connect these state nodes. Figure 1 shows a super control graph that eliminates the pseudo nodes. The edges that connect the nodes could be

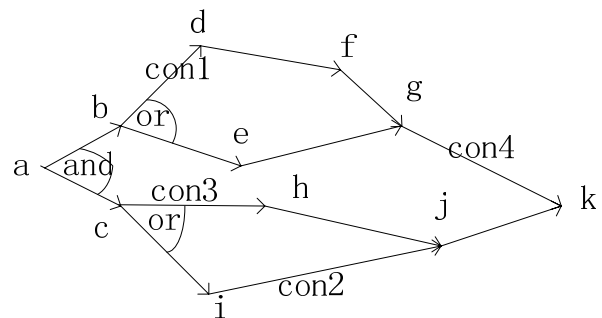


Figure 1. Control Graph for Workflow.



Figure 2. Trigger for Workflow.

considered as a conditional transition. As part of the control graph, triggers are interpreted by ECA rules, that is event, condition and action. As showed in Figure 2, when event happen and condition holds, then the corresponding action would be activated. Events could also be considered as special activities and incorporate them directly into the control graph in appropriate spots.

Under the control layer lays the database layer, which reflects the update of the data states by the operations of users.

Properties of workflow are divided into control properties and data properties, in which control properties defines the scheduling constraints such as sequence or dependencies of the running activities, while data properties describes the data constraints during the workflow execution life cycle.

Common approaches for the modeling of workflow are Petri-net[2, 3], UML activities graph[4, 5], process algebra[6] etc. However, these approaches did not provide an unified framework for the modeling of control and data as well as property analysis. The most popular approach is to use Petri-net to model workflow, however,

This paper is supported by 863 National High-Tech Research and Development Plan, China, (2007AA01Z185)

has ignored defects in the interpreting of data states. UML activity graph is also used in modeling workflow, but have difficulties in reasoning and verification of properties because of its semi- formalized semantics. Process algebra such as CCS, CSP, LOTOS modeling workflow by making every activity as a process, and executed in the system transition. Properties are described in temporal logic, and verified by model checker, however, can not avoid the state space explosion. And more over, this approach still has problems in interpreting data states.

Logic could provide a precise description for the workflow model, and comprehensive mechanism for reasoning and verification of properties. Davulcu[7] used concurrent transaction logic(CTR) to model workflow, however, limited in control layer, thus can not provide the reasoning and verification in data properties.

For the problems that existed, we proposed an approach to model and analysis workflow based on temporal logic of actions (TLA), not only supplies the modeling of control and data, but also provides reasoning and verification for their properties.

The paper is organized as following: section II shows an over view of temporal logic of actions, and section III discuss about the modeling of workflow using TLA, as well as the property analysis. Besides, a case for our approach is also given in section III to demonstrate its using. Finally, we give conclusion and further works in section IV.

II. OVER VIEW OF TLA

Temporal logic of actions as an extension of traditional temporal logic, is used for developing, describing and reasoning about concurrent and reactive systems [8, 9]. The semantic of TLA is based on behaviors, which is a finite or infinite sequence of states. A state is a mapping from the variable to its value. Let s be a state, x as a variable, then the value of x is $s(x)$. We then give the following definition that may appear in TLA:

(1) A state function is a non-boolean expression built from variables and constant symbols, such as x^2+y+3 , and always written as ' f '. The semantic of a state function is expressed as $[f]$, which means the value of that function under a given state s . Formally described as:

$$s[f] = f(\forall v': s[v]/v).$$

(2) A state predicate, called a predicate for short, is a boolean expression built from unprimed variables and constant symbols, always write as P . like state function, semantic of predicate is expressed as $s[P]$, which means the boolean value of that predicate under a given state s . Formally described as :

$$s[P] = P(\forall v': s[v]/v).$$

(3)An action is a boolean-valued expression formed by unprimed variables, primed variables, and constant symbols. always write as A . Primed variables are labeled

by “'”, which means the after-value of a variable after an action was executed. Semantic of action is expressed as $s[A]t$, thus an action is considered as a before and after predicate, and specifies a possible step in a behavior of a system, in which unprimed variables refer to the first state of a step and primed variables refer to its second state. Besides, a predicate could be considered as an action with out primed variables. Formally described as:

$$s[A]t = A(\forall v': s[v]/v, t[v]/v).$$

The pair of states s, t is called an “A step” if and only if $s[A]t$ equals true. If action ' A ' represents an atomic operation of a system, then s, t is an A step if and only if executing the operation in an old state s of the system can produce a new state t .

A typical raw TLA formula called RTLA “ for ” is a composition of state functions, predicates and actions that introduced above. Its syntax could be described in BNF format as:

$$for \triangleq p \mid \square[A]_{(f)} \mid \neg for \mid for \wedge for \mid \square for .$$

A TLA formula is on the base of RTLA by adding initial condition, and fairness property. In order to make it clearer to describe, we call them as T-TLA, short for Traditional-TLA. Thus, a T-TLA formula is consist of three parts such as : $\phi \triangleq Init_{\phi} \wedge \square[N]_{(f)} \wedge F$, and we give descriptions for each part as following:

- Init: a predicate that specifying the initial values of variables.
- N: donates the next-state relation which represent the executions of individual atomic performances.
- f: donates all the flexible variables in a n-tuple.
- F: represents the conjunction of formulas of the form of $WF_f(A)$ or $SF_f(A)$ which donates the weak or strong fairness of a system, and symbol A donates an atomic action in the formula.

T-TLA formulas that added process algebra are called P-TLA[10]. P-TLA differed with T-TLA in adding concurrent operators on them, to express the dependencies of the TLA actions. Let A and B be different TLA actions, we then have the following P-TLA operator definitions:

- $A; B$: a sequential execution, which means execute A and then B.
- $A \oplus B$: an indeterminate execution, which means execute A or B, can be used to express “or” relation in control graph of a workflow.
- $A \parallel B$: a concurrent execution, which means interleave execute A and B, can be used to express “and” relation in control graph of a workflow.
- $(A)^*$: a loop execution, which means keep doing A forever.
- $A \uparrow$: an interrupt of a loop, which means execute A, then exit from the innermost containing ()*.

Each action or called P-TLA sub-expression that connected by the operators showed above could be considered as an RTLA formula, however, extended with a “label”, such as $l : A$. A label can be attached to any sub-expression of a P-TLA formula, and should be unique within the corresponding P-TLA formula. Then the following actions and predicates are defined:

- $l(pc, v_1, v_2, \dots, v_n)$: A TLA action. A step of this action consists of performing a step of one of the sub actions of A.
- $AT(l(pc, v_1, v_2, \dots, v_n))$: This predicate asserting that control is at the beginning of the sub-expression.
- $IN(l(pc, v_1, v_2, \dots, v_n))$: This predicate asserting that control is at the beginning or somewhere inside the sub-expression.
- $AFTER(l(pc, v_1, v_2, \dots, v_n))$: This predicate asserting that control is immediately after the sub-expression.

Assume a control variable pc , distinct from all variables that appear in primitive actions. We could have the semantics for P-TLA formulas by defining, for each P-TLA p :

- A collection of constants L_p called labels, with a distinguished element $AT(p)$ called the ‘AT’ label.
- Collection of actions A_p of the form $(pc = l) \wedge (pc' = k) \wedge A$, where A is an P-TLA sub-expression action in which pc does not occur, $l \in L_p$, and k is either a label in L_p or one of the special constants “Done” or “Exit”, which means the termination of P-TLA actions.
- Hiding is expressed with the ordinary TLA quantifier \exists . The formula $\exists x : F$ is true of a behavior if and only if there exists some sequence of values, one in each state of the behavior that can be assigned to the variable x that will make formula F true.

With the introduction for TLA given above, we could start using TLA for the modeling and analysis of workflow.

III. MODELING AND ANALYSE OF WORKFLOW

A. Modeling workflow

Control graph of a workflow is a directed graph, and could be defined by a four group: $\langle A, T, INI, FIN \rangle$, in which A is a set of activity nodes, T is a set of edges that connect those activity nodes. INI is the initial node, while FIN is a set of terminated nodes. Just like we stated above, the model of a workflow is divided into two parts, the part that describe the control graph with the underlying database, and the part that describe properties. A control graph is described in a P-TLA formula, updates of data base state are represented as P-TLA sub-expressions that reflects to each activity. Properties are defined in a T-TLA formula in which contains an initial predicate, a

next state relation and a fairness. The following TLA formulas describes the model showed in Figure 1.

$$\begin{aligned}
 &flow(pc) \triangleq \\
 &a; \left(b; \left(\begin{array}{l} (con1; d; f) \\ \oplus e \end{array} \right); g; con4 \right) \parallel \\
 &\left(\begin{array}{l} (con3; h) \\ \oplus (i; con2) \end{array} \right); j; l_k : k \\
 &Spec \triangleq \exists pc, hidden : \\
 &\wedge initial(hidden) \wedge AT(flow(pc)) \wedge constrs \\
 &\wedge \square [flow(pc)]_{(pc, hidden, state)} \wedge SF_{pc}(flow(pc))
 \end{aligned}$$

Formula $flow(pc)$ demonstrate how the control graph of a workflow could be transformed into a P-TLA formula. Activities in the control graph are labeled by $a...b$, and single activity appears in the form of $(pc: action)$, which describes the before-and-after data state of the execution of that activity. For example, activity ‘a’ in this model could be defined as:

$$a(pc) \triangleq l_a : act_a(x_1, x_2, \dots, x_n)_{(state - \{x_1, x_2, \dots, x_n\})}$$

in which l_a is the value of parameter ‘ pc ’ that used to label an activity. TLA action $act_a(\dots)$ describes the next-relation of that execution. State function $state$ shows the current data state. By the definition of TLA action, a TLA action is a binary relation over a pair of states such as $\langle s1, s2 \rangle$, thus a TLA action could be considered as an update of a global date state “state”. In the example above, the execution of activity $a(pc)$ update the value $x1, x2, \dots, xn$ in state and leaves other values unchanged, this was expressed by adding an stuttering step:

$$\langle state - \{x_1, x_2, \dots, x_n\} \rangle.$$

Symbols $con1..con4$ are predicates, defines the transition condition in the control graph. As predicate is a special TLA action, thus predicates could be treated as a special query activity in TLA workflow model without updating data state.

T-TLA formula $Spec$ defines the basic properties of the workflow, and consisted by four parts: Initial condition, control properties, next relation and fairness constraints. In the example above, initial condition $AT(flow(pc))$ asserts that the workflow should start from the initial activity. Control properties $constrs$ defines the scheduling constraints. TLA action $\square [flow(pc)]$ shows that the workflow should keep on working. Strong fairness asserts that either the activities in workflow is eventually executed, or its execution is not infinitely often possible.

B. Analyse of workflow

According to the modeling approach that we have introduced previously, a workflow model is divided into P-TLA part that defines the global dependencies of the activities with data update, and the T-TLA part that defines the properties. The properties made restrictions on the scheduling as well as database. However, the initial condition, next-step relation and fairness are all

generic constraints, thus specific constraints should be established for every certain workflows to meet the requirements. In TLA, properties could be simply divided into invariance properties and eventuality properties. An invariance property is always expressed by a TLA formula $\Box P$, where P is a predicate, and an eventuality property is always expressed by $\Diamond P$ or $P \rightsquigarrow Q$. Based on these two kinds of properties, we would extend them in control properties and data properties. For example, TLA formula $\Box (T < 100)$ describes a data based invariance property, which has the meaning of “system temperature should not beyond 100 degree”, and TLA formula:

$$\nabla I_b \rightsquigarrow \nabla I_g$$

describes a control based eventuality property, and has the meaning of “if activity b is executed, then activity g should be executed eventually”.

In the modeling approach that introduced in in section III part A, predicate *constrs* describes a control property that a workflow should hold. predicate *constrs* is a conjunction of a set of sub predicates, in the way of:

$$constrs = c_1 \wedge c_2 \dots \wedge c_n.$$

and determined by requirements. So let *flow* be a P-TLA workflow model, then we would have the relation as:

$$flow' = flow \wedge constrs.$$

in which *flow'* donates the P-TLA workflow model *flow* that filtered by constraints *constrs*. The relation between *flow'* and *flow* are represented by a special operator *spec* defined as :

$$flow' = spec(constrs, flow).$$

The meaning of *spec* is that applying constraints *constrs* on model *flow* and get a model *flow'* that satisfies *constrs*. By the use of *spec*, the example showed in section III part A should be modified as:

$$\begin{aligned} Spec &\triangleq \exists pc, hidden : \\ &\wedge initial(hidden) \wedge AT(spec(constrs, flow(pc))) \\ &\wedge \Box [spec(constrs, flow(pc))]_{(pc, hidden, state)} \\ &\wedge SF_{pc}(spec(constrs, flow(pc))) \end{aligned}$$

and equivalence to the following formula:

$$Spec \triangleq init_{flow'} \wedge \Box [flow']_{(state)} \wedge SF_{pc}(flow').$$

We first discuss about control properties. Based on the operator *spec*, we give the following axioms:

$$\begin{aligned} spec(\Diamond \nabla a, a) &= a & (1) \\ spec(\Diamond \nabla a, b) &= null \text{ if } a \neq b & (2) \\ spec(-\Diamond \nabla a, a) &= null & (3) \\ spec(-\Diamond \nabla a, b) &= b & (4) \\ spec(\Diamond \nabla a, a \oplus b) &= a & (5) \\ spec(\Diamond \nabla a, a \parallel b) &= a \parallel b & (6) \end{aligned}$$

Let T and K be two different T-TLA formula, we then have the following deduction Inferences from the axioms above:

$$spec(\Diamond \nabla a, T; K) = spec(\Diamond \nabla a, T); K \vee T; spec(\Diamond \nabla a, K) \quad (7)$$

$$spec(\Diamond \nabla a, T \oplus K) = spec(\Diamond \nabla a, T) \vee spec(\Diamond \nabla a, K) \quad (8)$$

$$spec(\Diamond \nabla a, T \parallel K) = spec(\Diamond \nabla a, T) \parallel K \vee spec(\Diamond \nabla a, K) \parallel T \quad (9)$$

if $G \triangleq T \parallel K$ and have $spec(\Diamond \nabla a, T) = T$, $spec(\Diamond \nabla b, K) = K$, then we have

$$spec(\Diamond \nabla a, G) = spec(\Diamond \nabla b, G) \quad (10)$$

if $G \triangleq T \oplus K$ and have $spec(\Diamond \nabla a, T) = T$, $spec(\Diamond \nabla b, K) = K$, then we have

$$G = spec(\Diamond \nabla a, G) \oplus spec(\Diamond \nabla b, G) \quad (11)$$

Examples of how to use *spec* to describe properties are given as following:

- Let G a P-TLA model of a workflow, property $\Phi \triangleq \nabla a \rightsquigarrow \nabla b$, which means in model G , if a is executed, then b would eventually be executed, is represented as :

$$spec(\Phi, G)$$

and $spec(\Phi, G) \triangleq spec(\Diamond \nabla b, spec(\Diamond \nabla a, G)) \vee spec(-\Diamond \nabla a, G)$

- Property “work flow would terminate normally” could be represented as:

$$\bigcup_{f \in FIN} spec(\nabla f, G)$$

in which FIN is the set of terminated activities.

Now we come to data properties. Invariance properties are always expressed in the form of $\Box P$, in which P is a predicate, such as $\Box (T < 100)$, $\Box (x > y)$. Eventually properties are always represented in the way of $\Diamond P$, $P \rightsquigarrow Q$, where P and Q are both predicates. Let c be a data property, then the proof of c is equivalence to the proof of $Spec \Rightarrow c$, where *Spec* is the basic constraint defined in model.

Proof of invariance properties could be reasoned by the following common rules(details of proof rules are showed in Leslie Lamport[8]):

$$(P \wedge (f' = f) \Rightarrow (\Box P \equiv P \wedge \Box [P \Rightarrow P']_f)) \quad (12)$$

$$\Box I \Rightarrow (\Box [N]_f \equiv \Box [N \wedge I \wedge I']_f) \quad (13)$$

$$(I \wedge [N]_f \Rightarrow I') \Rightarrow ((I \wedge \Box [N]_f) \Rightarrow \Box I) \quad (14)$$

To the rules above, P is a predicate, I is an invariant, N is a TLA action, f is a state function. According to the expression $Spec \triangleq init_{flow'} \wedge \Box [flow']_{(state)} \wedge SF_{pc}(flow')$, the goal is to proof the implementation relation $Spec \Rightarrow \Box P$, and the main idea is to find out an invariant I to satisfy the following:

$$Init \Rightarrow I \quad (15)$$

$$I \Rightarrow P \quad (16)$$

$$I \wedge [flow]_{(state)} \Rightarrow I' \quad (17)$$

$$P \wedge [flow]_{(state)} \Rightarrow P' \quad (18)$$

If premise (15) ~ (17) hold, then solution (18) holds.

According to the definition of *Spec* and formula (1), we have the implementation relation:

$$Spec \Rightarrow init \wedge \Box [flow]_{(state)}$$

and then have $Spec \Rightarrow I \wedge \Box [flow]_{(state)}$, by the formulas (18) and (14), we could have the result $Spec \Rightarrow \Box P$. The key idea of this proofing is to find out the invariant *I*. A state predicate *I* is an invariant of a specification *S*, iff *S* implies $\Box I$. However, sometimes predicate *P* could be treated as *I*.

Different with the proof of invariance properties, the proof of eventuality properties relies on the fairness predicates. Common rules for its proof are listed as following:

$$P \wedge [N]_f \Rightarrow (P' \vee Q') \quad (19)$$

$$P \wedge \langle N \wedge A \rangle_f \Rightarrow Q' \quad (20)$$

$$P \Rightarrow Enable(A)_f$$

(21)

$$\Box [N]_f \wedge SF_f(A) \Rightarrow (P \rightsquigarrow Q) \quad (22)$$

If premise formula (19) ~ (21) hold, then formula (22) holds.

According to the rules above, let *P* and *Q* be two different predicates, the key idea of proofing $Spec \Rightarrow (P \rightsquigarrow Q)$ is to find out a sub control flow *flow'* which is enabled by predicate *P* and the execution of *flow'* make predicate *Q* holds, then let action *A* be substituted with *flow'*, action *N* be substituted with *flow*, then we would have the formula: $(Spec' \triangleq \Box [flow]_f \wedge SF_f(flow')) \Rightarrow (P \rightsquigarrow Q)$ holds. As we have $Spec \Rightarrow Spec'$, then we would have the implementation relation: $Spec \Rightarrow (P \rightsquigarrow Q)$.

We have discussed about control properties and data properties above and separately. However, when in practice, data properties are always associate with control constraints, which means a data properties should only necessary to be hold at the assumption of some certain selected control path. A simple example would demonstrate this: only when a customer chooses to buy a Book-A, would its stock amount decreases. This simple property contains two elements: one is of a control assumption that, a customer chooses to buy a Book-A, the other is of a data property that the stock amount of Book-A should decrease. However, this data property only needs to be hold under the assumption of the control selection, otherwise, need not to be hold. Thus a unified framework for specifying both control and database state of workflow would take the advantage in reasoning the properties of such condition.

C. Case Study

The following case shows a workflow for a trip planning. We first give a short description for this case, and then we demonstrate how to model the workflow and how the properties are verified. As we could see in Figure 3 which shows the control graph of this case, the control starts from a *start* activity and demand for customer's feedback. When customer agree with the trip scheduling, they could choose to purchase trip canceling insurance, accident insurance or purchase nothing. When decision was made, they wait for the traveling day to come. This may come to a contretemps that the trip may be canceled, or had accident during the trip or fortunately, started and finished normally. If the customer had purchased cancel insurance or accident insurance, they would have the corresponding compensation. As we could see in this example, the flow is consisted of a set of activities as well as its guard and several predicates. Under the control flow, lays the un-observational data base and when activities are executed during a running life cycle, data base state might be changed each. Time related events may be concerned in the way that, when time arrivals the corresponding activity should be executed. Table I demonstrates the description of every activities and predicates that appear in the control graph. By the descriptions, we first give the P-TLA model as following:

$$\begin{aligned} & flow(pc) \triangleq \\ & Start; Sol_feedback; con1; Travel_reg; \\ & \left(\begin{array}{l} (con2; Pur_c_ins) \\ \oplus (con3; Pur_a_ins) \\ \oplus con4 \end{array} \right); \\ & Pre_travel; \\ & \left(\begin{array}{l} con5; Start_travel; \\ \left(\begin{array}{l} (con6; Acc_occur; ((con8; Comp_cus))) \\ \oplus con9 \end{array} \right) \\ \oplus con7 \end{array} \right); End \end{array} \right) \\ & \oplus \\ & \left(\begin{array}{l} (con10; Cancel_travel; ((con11; Comp_cus))) \\ \oplus con12 \end{array} \right); \\ & End \end{aligned}$$

According to requirement, our properties could be described as "if customer has purchased accident insurance, then if accident occurs, the customer would eventually be compensated". According to this properties, we could see that it has a control property "accident occurs", and under this control constraint, a data constraint should hold. Let *pconstrs* be control constraint, and *sconstrs* be data constraint, the T-TLA formula and proof goal could be represented as:

$$\begin{aligned} & Spec \triangleq \exists pc, hidden : \\ & \wedge initial(hidden) \wedge AT(spec(pconstrs, flow(pc))) \\ & \wedge \Box [spec(pconstrs, flow(pc))]_{(pc, hidden, state)} \\ & \wedge SF_{pc}(spec(pconstrs, flow(pc))) \end{aligned}$$

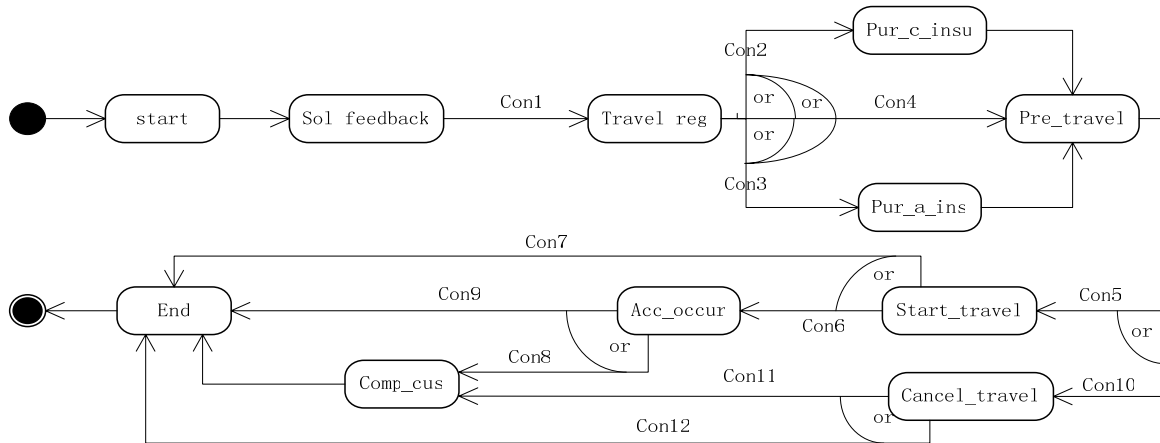


Figure 3. Control Graph for Workflow.

TABLE I.
DESCRIPTIONS OF ACTIVITIES AND PREDICATES

Start	Start to execute the flow	Sol_feedback	Soliciting customer's feedback
Travel_reg	Registering traveling	Pur_c_ins	Purchase cancel insurance
Pur_a_ins	Purchase accident insurance	Pre_travel	Prepare for traveling
Start_travel	Start traveling	Cancel_travel	Cancel traveling
Acc_occur	Accident occurs	Comp_cus	Compensate customers
End	End executing	Con1	If customer agree to travel
Con2	If customer agrees to purchase cancel insurance	Con3	If customer agrees to purchase accident insurance
Con4	If customer does not agree to purchase any insurance	Con5	If time reaches and travel starts
Con6	If accident occurs	Con7	If accident does not occur
Con8	If customer has accident insurance	Con9	If customer does not have accident insurance
Con10	If time reaches and travel canceled	Con11	If customer has cancel insurance
Con12	If customer does not have cancel insurance		

$Spec \Rightarrow sconstrs$

As we have introduced in section III part A, a single activity of a workflow is defined as:

$$a(pc) \triangleq l_a : act_a(x_1, x_2 \dots x_n)_{\langle state - \{x_1, x_2 \dots x_n\} \rangle}.$$

According to this, we give definitions for primary activities as following:

First define database *state* that would be used in our activity definition:

$$state = \left\langle \begin{array}{l} time : INT, cancelinsu : BOOL, \\ accinsu : BOOL, hascancel : BOOL, \\ hasaccident : BOOL, paycus : BOOL, \\ agreeaccinssu : BOOL \dots \dots \dots \end{array} \right\rangle$$

The state defined above donates the data base of our example: time, a integer type simply donates the time progress, and reflects the time based event in the way that

when an activity is executed, the variable *time* would increase, and when the time satisfies the guard of a time based event, the corresponding activity would be activated. Variable *cancelinsu* represents whether a customer choose to purchase a cancel insurance, while *accinsu* represents whether a customer choose to purchase an accident insurance. Variable *hascancel* represents whether a trip has been canceled and *hasaccident* represents whether accidents has occurred during the trip. Variable *paycus* donates whether the customer has had compensation and *agreeaccinssu* represents whether a customer already got an accident insurance. Notice that, when in practice, a time constraints would be much more complex, thus more definitions should be made on time such as time progression, clock or timer, however, this is not what our paper focus, thus we only give a raw definition in the way to increase variable time in every execution of a activity. Based on the data state given above, the primary activities are given as below:

```

// action Purchase cancel insurance
act_Pur_c_ins ≜ cancelinsu = false ∧
cancelinsu' = true ∧ time' = time + 1
// action Purchase accident insurance
act_Pur_a_ins ≜ accinsu = false ∧
accinsu' = true ∧ time' = time + 1
// action Start travelling
act_Start_travel ≜ time = deadline ∧ time' = time + 1
// action Cancel travelling
act_Cancel_travel ≜ hascancel = false ∧
hascancel' = true ∧ time' = time + 1
// action Accident occurs
act_Acc_occur ≜ hasaccident = false ∧
hasaccident' = true ∧ time' = time + 1
// action Compensating
act_Comp_cus ≜ paycus = false ∧
paycus' = true ∧ time' = time + 1
// predicate con3
con3 ≜ agreeaccinsu = true
// predicate con5
con5 ≜ time = deadline ∧ hascancel = false
// predicate con6
con6 ≜ hasaccident = true
// predicate con8
con8 ≜ accinsu = true
    
```

Thus property “if accident occurs, the customer would eventually be compensated”, could be represented as:

$$\begin{aligned}
 pconstrs &\triangleq \nabla Acc_occur \\
 sconstrs &\triangleq (accinsu = true) \rightsquigarrow (paycus = true)
 \end{aligned}$$

The validation of this property is equivalence to the proof of implementation relation:

$$Spec \Rightarrow sconstrs .$$

Let $flow'(pc) = spec(pconstrs, flow(pc))$, by the rules of $spec$ given in previously, the expression of $flow'(pc)$ is showed as following:

$$\begin{aligned}
 flow'(pc) &\triangleq \\
 &Start; Sol_feedback; con1; Travel_reg; \\
 &\left(\begin{array}{l} (con2; Pur_c_ins) \\ \oplus (con3; Pur_a_ins) \\ \oplus con4 \end{array} \right); Pre_travel; \\
 &con5; Start_travel; \\
 &\left(\begin{array}{l} (con6; Acc_occur; ((con8; Comp_cus)) \\ \oplus con9 \\ \oplus con7 \end{array} \right); End
 \end{aligned}$$

$$\begin{aligned}
 Spec' &\triangleq \exists pc, hidden : \\
 &\wedge initial(hidden) \wedge AT(flow'(pc)) \\
 &\wedge \square [flow'(pc)]_{(pc, hidden, state)} \wedge SF_{pc}(flow'(pc))
 \end{aligned}$$

By the definition of $Spec$, it is clearly that we have $Spec \Rightarrow Spec'$. Let $P \triangleq personassu = ture$, and $Q \triangleq paycus = ture$, the proof is equivalence to $Spec' \Rightarrow P \rightsquigarrow Q$, by the proof rules given previously, we have the following formulas:

$$P \wedge [flow'(pc)]_f \Rightarrow (P \vee Q) \tag{23}$$

$$P \wedge \langle flow'(pc) \wedge flow''(pc) \rangle_f \Rightarrow Q \tag{24}$$

$$P \Rightarrow Enable(flow''(pc))_f \tag{25}$$

$$\square [flow'(pc)]_f \wedge SF_f(flow''(pc)) \Rightarrow (P \rightsquigarrow Q) \tag{26}$$

If (23),(24) and (25) hold, then (26) holds. According to (28) and the definition of predicate P , we could see that in $flow''(pc)$, after activity Pur_a_ins was executed, then would have predicate P true. According to (24) and the definition of predicate Q , we see that after activity $Comp_cus$ was executed, then would have predicate Q true. Thus if a sub-flow $flow''(pc)$ in which Pur_a_ins and $Comp_cus$ eventually executed:

$$\begin{aligned}
 flow''(pc) &\triangleq \\
 &spec(\nabla Comp_cus, spec(\nabla Pur_a_ins, flow'(pc))) \\
 \text{and} \\
 flow''(pc) &\triangleq \\
 &Start; Sol_feedback; con1; Travel_reg; \\
 &con3; Pur_a_ins; Pre_travel; con5; Start_travel; \\
 &con6; Acc_occur; con8; Comp_cus; End
 \end{aligned}$$

Then according to formula (26) and formula $Spec' \Rightarrow \square [flow'(pc)]_f \wedge SF_f(flow''(pc))$, we could have the implementation relation: $Spec' \Rightarrow P \rightsquigarrow Q$ holds, and the eventually property $P \rightsquigarrow Q$ holds. Which means, under the control constraint $pconstrs$, the data constraint $sconstrs$ holds.

From this example, we see that the modeling and validate of properties of a workflow are done in an unified framework and data properties are always related with control constraints. When in practice, situation would be much more complex, however, special operator $spec$ would help us to minimize the control path and efficient algorithm could be designed.

IV. CONCLUSION AND FURTHER WORKS

In this paper, we used TLA for the modeling and analysis of workflow, and established an unified framework for the reasoning and validation of control and data properties. Compared with other modeling approaches, TLA has its advantages in providing a simpler and uniform way to describing both control and database factors, as well as for their properties reasoning and verification. We demonstrate several comparisons showed in Table II.

In our approach, control graph will be constructed by TLA concurrent operators, and represented by a P-TLA formula in which each activity is represented by a “label and action” pair, and reflect the database update in every execution. Properties are represented by a T-TLA

TABLE II.
COMPARISON OF DIFFERENT WORKFLOW MODELING APPROACHES

Modeling approach	Formalization	Has data description	Has control description	Features
Notation	General	no	yes	Easy to use
UML activity	General	No	yes	Easy to use
Petri net	Strict	No	yes	Good at analysis in control properties
TLA logic	Strict	no	yes	Good at analysis in control and data properties

formula, and introduced operator “*spec*” to TLA to filtrate the model that satisfies the corresponding control constraints. By the rules of the specific operator, efficient algorithm could be designed for the property verification.

The reasoning of properties is simple like the proof of normal TLA formulas. As we can see, the validation of data property may base on the control properties. Thus, a unified framework would be better for properties reasoning. Besides, automated TLA prove method and the corresponding tools are existed, so it is possible for us to automatically reason the workflow properties.

On the base of this approach, the composition of workflow model would be listed in our further works. Task refinement as well as its soundness validation is one kind of workflow composition approaches. More over, the way of describing the semantics of time constraints, resource and other factors of workflow would be concerned into our further study.

REFERENCES

- [1] D.Georgakopoulos, M. Hornick, A. Sheth, An overview of workflow management: From process modeling to workflow automation infrastructure. Distributed and Parallel Databases, 1995, 3 (3): 119~153.
- [2] Salimifard K, Wright M. Petri net based modeling of workflow systems: An overview. European Journal of Operational Research , 2001, 134 (3): 664~676.
- [3] W.M.P van der Aalst. The application of Petri nets to work flow management. The Journal of Circuits, Systems and Computers, 1998, 8 (1): 21~66.
- [4] Marlon Dumas and Arthur H.M. ter Hofstede. UML Activity Diagrams as a Workflow Specification Language. In Proceedings of the UML' 2001 Conference, Cooperative Information Systems Research Centre, Queensland University of Technology.
- [5] R.Eshuis, R.Wieringa. Verification support for workflow design with UML activity graphs. The 24th Int'l Conf on Software Engineering, Orlando, USA, 2002.
- [6] Salomie, Ioan Cioara, A Layered Workflow Model Enhanced with Process Algebra Verification for Industrial Processes. Intelligent Computer Communication and Processing, 2007 IEEE.
- [7] Davulcu H, Kifer M, Ramakrishnan C, Ramakrishnan I. Logic based modeling and analysis of workflows. In : Proceedings of the ACM Symposium in PODS ' 98, Seattle, USA, 1998. 25~33.
- [8] Leslie Lamport. The Temporal Logic of Actions[J]. ACM Transactions, 1994.
- [9] Leslie Lamport. verification and specification of concurrent programs. proceedings of a REX Workshop, Netherlands, June, 1993.
- [10] Leslie Lamport. Adding Process Algebra to TLA. <http://research.microsoft.com/users/lamport/pubs/adding-process-algebra.pdf>.

First A. Author Chen Shu, 01/30/1981

Bachelor degree in computer science WuHan university computer science department WuHan P.R.China, 2003.

Master degree in software engineering WuHan university software department WuHan P.R.China, 2006.

PHD candidate in software engineering WuHan university software department WuHan P.R.China.

Majored in software engineering, software formalization and automation.

Second B. Author Wu Guo Qing, 10/1954

Professor, doctoral tutor in WuHan university computer science department.

Majored in software engineering, software formalization and automation.