A PROPERTY SPECIFICATION LANGUAGE FOR WORKFLOW DIAGNOSTICS

E.E.Roubtsova
Technical University Eindhoven
Den Dolech 2, PO 513, 5600MB, Eindhoven, the Netherlands

Keywords: Workflow log, workflow mining, property specification language, property check.

Abstract: The paper presents a declarative language for workflow property specification. The language has been developed to help analysts in formulating workflow-log properties in such a way that the properties can be checked automatically. The language is based on the Propositional Linear Temporal Logics and the structure of logs. The standard structure of logs is used when building algorithms for property checks. Our tool for property driven workflow mining combines a tool-wizard for property construction, property parsers for syntax checks and a verifier for property verification. The tool is implemented as an independent component that can extend any process management system or any process mining tool.

1 INTRODUCTION

Development of different workflow mining methodologies is triggered by the increasing attention to security of business and to performance of companies. Changes in the business situation demand from companies to be more flexible and react faster on every deviation of a business process from a correct process or on every unexpected group of events. That is why recording, monitoring and diagnosing logs of events becomes a normal practice of any good organized business.

A log is a textual file containing information about events recorded in the order of happening. Workflow logs are huge files, and their effective diagnostics is not possible without formal specification of properties and tool support for property checks. Modern enterprize processes are site-defined, i.e. an enterprize defines its own processes. Process management makes sure that different processes will be made according to the site-defined rules or properties. To do this the process managers fulfill diagnostics of workflow logs with respect to such properties.

The monitoring and diagnosing logs can be organized in many different ways (Aalst W.M.P. van der, B.F.van Dongen, J.Herbet, L.Maruster, G.Schimm, A.J.M.M.Weijters, 2003). Some authors direct their research on discovering specific workflow metrics like entropy, periodicity, causality or concurrency (Cook J.E., A.L.Wolf, 1998). Those metrics provide measures to quantify discrepancies between a reference model and a logged process (Cook J.E., A.L.Wolf, 1999). Other approachers derive a visual workflow model from a log (Schimm G., 2002),(Aalst W.M.P. van der, T. Weijters, L. Maruster, 2004). Such a workflow model can be compared with a standard reference model. The correspondence between a workflow constructed from a log and the standard reference model indicates correctness of a logged business process.

However, the reference model is often too complex, or even not available at all. To reason about correctness of a logged business process an enterprise defines some properties of a correct process. However, presentation of such properties is usually informal and ambiguous. This ambiguity complicates diagnostics of workflows and makes tool support for such diagnostics impossible.

In this paper we present a language for specification of workflow properties and a tool for property checks. The language is based on the Propositional Linear Temporal Logic (Pnueli, 1981) and the structure of workflow logs (Dongen B.F. van, W.M.P.van der Aalst, 2004). Using PLTL-based language as a driving forse for data mining of workflow logs is a new approach, although temporal logic has been used for modelling, scheduling (Davulcu H., M.Kifer, C.R. Ramakrishman, I.V. Ramakrishman, 1998) and analy-

E.Roubtsova E. (2005).
A PROPERTY SPECIFICATION LANGUAGE FOR WORKFLOW DIAGNOSTICS.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 297-302
DOI: 10.5220/0002516502970302

sis of workflows (Chuang Lin , Yang Qu , 2004).

The reminder of the paper is organized as follows. Section 2 formalizes a workflow log. Section 3 shows the grammar and semantics of our workflow property language. Section 4 presents examples of workflow properties and their expressions in our language. Section 5 concerns the implementation issues of constructing, parsing and checking of property-expressions. Section 6 concludes the paper.

2 FORMAL REPRESENTATION OF A LOG

To define a language for workflow properties specification we formalize a log. An event is represented in a log by an *audit trail entry* (ate). An ate is a tuple with the following fields:

$$ate = (WE, ET, TS, O),$$

WE : $String$ *(workflow element),*
ET : $String$ *(event type),*
TS : $double$ *(time stamp)* and
O : $String$ *(originator).*

A sequence of $ate's$ in a log is called a *process instance* (Figure 1). A process instance defines a total order relation on the set ATE of audit trail entries $T = \{(ate_1, ate_2)| \ ate_1, ate_2 \in ATE \ \wedge \ ate_2 \ follows \ ate_1\}$:

$$p = (pname, (WE, ET, TS, O), T).$$

A set P of process instances defines a workflow $w = (wname, P)$. A set W of workflows is grouped into a log: $l = (lname, W)$.

According to the definition, a process instance has a twofold nature: it is both a sequence of ate's and a relation. Making diagnostics of a process instance, we formulate the properties of the process instance considering the position of a current record in the sequence and the values of the fields of this record or of the records that follow the current one.

3 A LANGUAGE TO SPECIFY PROPERTIES OF WORKFLOWS

3.1 Properties of an audit trail entry

Any appearance of a field identifier for a specific ate derives for this ate the value of the field with the corresponding name. To compare the values of the string-fields WE, ET, O with some site-defined values we use operations equal " $=$ " and not equal "$!=$". To work with the time stamp field TS we use the complete set of comparison operators. So, an elementary property of an audit trail entry is represented by a property of a field in form of one of the following expressions:

$< element >::= true \mid false \mid$
$WE \ = < string > \mid WE \,!= < string > \mid$
$ET \ = < string > \ \mid ET \,!= < string > \ \mid$
$O \ = < string > \ \mid O \,!= < string > \mid$
$TS \ = < double > \mid TS \,!= < double > \mid$
$TS \ < < double > \ \mid TS \ > < double > \ \mid$
$TS \ >= < double > \ \mid TS \ <= < double > .$

The set of elementary property expressions also contains some functions, for example, to derive the time information from the time stamp:

$$< element >::= HOURS(TS) =< double > .$$

For the sake of simplicity, we do not define here the complete set of functions.

We also assume using variables to memorize field values and the arithmetic and the comparison operations on variables. To assign values of fields to variables at the level of elementary properties we define the function $assign$: " $:=$ ", which always returns value $true$:

$< element >::=$
$< Variable >:= WE \mid < Variable >:= ET \mid$
$< Variable >:= TS \mid < Variable >:= O).$

Such extension of the language allows analysts to specify some additional properties of business processes.

Elementary properties are combined into logical expressions by means of logical operations (the operations are represented in order of decreasing priority):

$< expr >::=< element > \ |\neg \ < expr > \ \mid$
$< expr > \ \& \ < expr > |< expr > \ || \ < expr > .$

3.2 Properties of a process instance

A property of an audit trail entry is an elementary property of a process instance. To express the ordering properties of a process instance we adapt the set of the linear temporal operators: $NEXT$, $ALWAYS$, IN_FUTURE (Bernard B., M.Bidoit, A.Finkel, F.Laroussinie, A.Petit, L.Petrussi, Ph.Schnoebelen, 2001). We also introduce useful temporal operators $EXISTS\_UNTIL$ and $ALWAYS\_UNTIL$ with intuitively clear names. So, a set of possible process instance properties can be represented inductively:

$< tempexpr >:=< expr > \mid \neg < tempexpr > \ \mid$
$< tempexpr > \ \& \ < tempexpr > \mid$
$< tempexpr > \ || \ < tempexpr > \mid$
NEXT $(< tempexpr >) \mid$
IN_FUTURE $(< tempexpr >) \mid$
ALWAYS $(< tempexpr >) \mid$
$(< tempexpr >)$

EXISTS_UNTIL $(< tempexpr >)$ |
$(< tempexpr >)$
ALWAYS_UNTIL $(< tempexpr >)$.

The definition above allows analysts nesting of properties, which means changing the position of the ate in the process instance for which the property must hold.

The semantics of the properties is defined by a satisfaction relation. To define the semantics we construct a Kripke structure (Alur R., C. Courcoubetis, D.L. Dill, 1993): $M_{tempexpr} = (ATE, T, \nu)$, where ATE is a finite set of audit trail entries being states of a process instance;
T is a binary ordering relation on audit trail entries which defines the initial audit trail entry and a single transition from each ate to the next one;
$\nu : ATE \rightarrow 2^{tempexpr}$ assigns true values of a temporal property to each ate in the process instance:

1. (ATE,T,position ate_i) $\models expr$ iff $expr \in \nu(ate_i)$.
2. (ATE,T,position ate_i) $\models \neg tempexpr$ iff $expr \notin \nu(ate_i)$.
3. (ATE,T,position ate_i) $\models tempexpr \wedge tempexpr_1$ iff $ate_i \models tempexpr$ and $ate_i \models tempexpr_1$.
4. (ATE,T,position ate_i) $\models tempexpr \; || \; tempexpr_1$ iff $ate_i \models tempexpr$ or $ate_i \models tempexpr_1$.
5. (ATE,T, position ate_i) $\models$ NEXT $(tempexpr)$ iff *for* (ATE, T) : $ate_i, ate_{i+1}, ...$ $ate_{i+1} \models tempexpr$.
6. (ATE,T, position ate_i) $\models$ ALWAYS $(tempexpr)$ if *for* (ATE, T) : $ate_i, ate_{i+1}, ...$ for all $j \geq i$ $ate_j \models tempexpr$.
7. (ATE,T, position ate_i) $\models$ IN_FUTURE $(tempexpr)$ iff *for* (ATE, T) : $ate_i, ate_{i+1}, ...$ for some $j \geq i$ $ate_j \models tempexpr$.
8. (ATE,T, position ate_i) $\models ((tempexpr)$ EXISTS_UNTIL $(temexpr_1))$ iff *for* (ATE, T) : $ate_i, ate_{i+1}, ...$ for some $j \geq i$ $ate_j \models tempexpr_1$ and for some k $k < j$ $ate_k \models tempexpr$.
9. (ATE,T, position ate_i) $\models ((tempexpr)$ ALWAYS_UNTIL $(temexpr_1))$ iff *for* (ATE, T) : $ate_i, ate_{i+1}, ...$ for some $j \geq i$ $ate_j \models tempexpr_1$ and for all k $k < j$ $ate_k \models tempexpr$.

3.3 Properties of a log

The language for specification of a log properties has to be completed by the specification of the scope of a property. A property of a log can cover one, several or all process instances of a process and one, several or all workflows etc.:

$logproperty$::=$< LOG >$
$< WORKFLOW >$
$< INSTANCE >< tempexpr >$;
$< LOG >$::=FOR-ALL-LOGS |
EXISTS-LOG |
FOR-LOG $< lname >$;
$< WORKFLOW >$::= FOR-ALL-WORKFLOWS |
EXISTS-WORKFLOW |
FOR-WORKFLOW $< wname >$;
$< INSTANCE >$::= FOR-ALL-INSTANCES |
EXISTS-INSTANCE |
FOR-INSTANCE $< pname >$.
$< lname >, < wname >, < pname >$::=$< string >$.

4 EXAMPLES OF PROPERTIES OF A PROCESS INSTANCE

Sometimes it is not trivial to specify a property. To ensure correctness of analysis companies should have specialists responsible for specification of properties and documentation of their business semantics. Those properties can be saved under recognizable names in order to be used by personnel doing everyday monitoring of logs. We show here some examples of possible properties.

4.1 Security properties

Security of editing. When creating own files, we can set up restrictions for other network users. For example, only for two persons *Rob* and *Ana* we can allow editing of information.

FOR-LOG ”$access$”
FOR-WORKFLOW ”$access\ to\ document\ X$”
FOR-INSTANCE ”$the\ 22d\ of\ November$”
$ALWAYS(\neg(WE = \text{"}edit\ document\ X\text{"}) \; ||$
$(WE = \text{"}edit\ document\ X\text{"} \;\&$
$(O = \text{"}Rob\text{"} \; || \; O = \text{"}Ana\text{"})))$.

Security for electronic voting is another example of security properties. The uniqueness property declares that no voter should be able to vote more than one time.

FOR-LOG ”$voting$”
FOR-WORKFLOW ”$voting\ June\ 2003$”
FOR-INSTANCE ”$the\ 10th\ of\ June$”
(ALWAYS ($\neg$(
$(WE = \text{"}voting\text{"} \;\&\; v := O \;\&\; v = v_1)$
EXISTS_UNTIL
$(WE = \text{"}voting\text{"} \;\&\; v_1 := O))))$.

4.2 Four eyes principle

The four eyes principle (FourEyes, 2004) dictates that at least two different persons must witness certain activities. This helps to protect an organization from dishonest individuals and unintended mistakes. Our property indicates that the first workflow element is ”authentication” and the second workflow element is

”authentication” but the originators of these events are different:

FOR-LOG $"organization"$
FOR-WORKFLOW $"logon"$
FOR-INSTANCE $"network\ logon$ 18.09.2004”
IN_FUTURE $((WE = "authentication"\ \&$
$or_1 := O)\ \&$
NEXT $(WE = "authentication"\ \&$
$or_2 := O)\ \&\ (or_1\ != or_2))$.

The property uses operator NEXT nested under the operator IN_FUTURE. Variables or_1 and or_2 save the values of the event originators to compare them: $(or_1\ != or_2)$.

4.3 Separation of functions

There are business activities which can't be fulfilled by the same person. For example, an application for a business trip ($ET = "start"$) can be initiated by any member of a department. A permission for a business trip ($ET = "complete"$) is usually given by the chief of the department. However, if the chief applies for a trip, then the permission for his trip can't be given by himself. The property below specifies that the permission to a travel of the chief of the department should not be given by himself.

FOR-LOG $organization$
FOR-PROCESS $business\ trip$
FOR-INSTANCE 18.10.2004
$((WE = "business\ trip"\ \&$
$ET = "start"\ \&\ O = "H.\ de\ Jong")$
EXISTS_UNTIL
$(WE = "business\ trip"\ \&$
$ET = "complete"\ \&\ O\ != "H.\ de\ Jong"))$.

4.4 Deadlines

Field $timestamp\ TS$ allows us to represent deadlines as properties. Let us imagine that the activity A of the process-instance shown in Figure 1 should be completed till 18.00:

FOR-LOG $"pn - ex - 15.xml"$
FOR-WORKFLOW $"main - process"$
FOR-INSTANCE $"experiment"$
IN_FUTURE $(WE = "A"\ \&$
$ET = "complete"\ \&\ HOURS(TS) \leq 18.00)$.
This deadline-property holds for the process instance in Figure 1.

Relative deadlines. For example, we may demand that an activity should be completed in ten hours from the start:

FOR-LOG $"pn - ex - 15.xml"$
FOR-WORKFLOW $"main - process"$
FOR-INSTANCE $"experiment"$
$((WE = "A"\ \&\ ET = "start"\ \&$
$t_1 := HOURS(TS))$
EXISTS_UNTIL
$(WE = "A"\ \&\ ET = "complete"\ \&$
$t_2 := Hours(TS)\ )\ \&\ t_2 - t_1 \leq 10))$.
The property does not hold for the process instance in Figure 1.

5 IMPLEMENTATION OF THE METHODOLOGY OF THE PROPERTY DRIVEN WORKFLOW MINING

To implement the property driven workflow mining we have solved the following tasks: property constructing, property parsing and property checking.

Property constructing has been implemented as a wizard-tool which helps analysts to choose the scope of property-oriented mining and formulate a property. The wizard shows the set of the temporal operators, the lists of fields and logical connectors (Figure. 2) which allow constructing properties.

Property parsing has been implemented using the Java Compiler Compiler (JavaCC. A parser/scanner generator for java, 2004) in the grammar presented in Section 3. There have been developed parsers for temporal expressions (with nesting), for expressions, and for simple logical expressions to build property checkers on their basis.

Property checking of expressions is performed as interpretation of expressions during parsing. An expression $expr$ is evaluated for one record at a specified $position$ in a process instance.

The temporal property checking could not be done during parsing because of the parser backtracking problem. So, each kind of temporal operator is evaluated by a corresponding function.

1. Function $VerifyNext(p, i, expr)$ checks the value of expression $expr$ for the next ate $(i + 1)$ to the current one.
2. Function $VerifyInFuture(p, i, expr)$ searches an ate of the process instance p where $expr = true$ and then returns value $true$. If such an ate has not been found, the function returns $false$.
3. Function $VerifyAlways(p, i, expr)$ calls the expression checker for every ate of process instance p starting from i. Only if the expression is $true$ for all $ate's$ the function returns value $true$.
4. Function $VerifyExistsUntil(p, i, expr, expr_1)$ searches an ate j of the process instance p where $expr_1 = true$ and then from the ate i until the ate j searches an ate where $expr = true$. Only if both searches are successful, then the function returns value $true$.

5. Function $VerifyAlwaysUntil(p, i, expr, expr_1)$ searches an $ate\ j$ of the process instance p where $expr_1 = true$ and then from the $ate\ i$ until the $ate\ j$ checks if for all the $ate's\ expr = true$. Only if both searches are successful, then the function returns value $true$.

Recursive application of these functions is used to evaluate temporal logical expressions for a process instance.

6 CONCLUSION

Correctness control of workflows using log-mining can not be fulfilled in general: it is always fulfilled with respect to some side-defined criteria. If these criteria come from the workflow design, for example, in form of a standard workflow, then we can control the deviations of the real process from the standard workflow. But the standard workflow can also be developed with mistakes. So, it is always reasonable to specify some correctness criteria that are not based on the design decisions. With respect to these criteria both the standard designed process and the real logged process can be analyzed.

In this paper we have proposed formulating those criteria as workflow properties in a declarative PLTL based language. We have developed the language for the standard log-format and implemented a version of the language for a fixed log structure as a component-prototype which can extend any process management system or any process mining tool. Construction of properties and property checks are supported by the component. The properties formulated by specialists are saved under recognizable names in order to be used by personnel doing the everyday monitoring of logs.

We are going to integrate our component into the ProM tool (ProM framework. TU/e, 2004) being under development by the Business Process Modelling group of the Technology Management Faculty at the Technical University Eindhoven. Such an integrating our component into this framework will allow us to find the methodology of property specification and checking which is the most suitable for monitoring of workflow logs. If the format of logs is changed in the future covering the XML-log format, we can develop automatic adaptation of the property language to the structure of a log. For example, introducing new fields into the description of an event (ate) will change the set of elementary formulas in our workflow mining language. This predictable extension of the language will cause the predictable modification of the wizard-tool and the property checking tool.

ACKNOWLEDGEMENT

The author thanks all members of the Business Process Modelling group of the Technology Management Faculty at the Technical University Eindhoven for sharing insights.

REFERENCES

Aalst W.M.P. van der, B.F.van Dongen, J.Herbet, L.Maruster, G.Schimm, A.J.M.M.Weijters (2003). Workflow mining: A survey of issues and approaches. In *Data and Knowlege Engineering*, volume 47, Issue: 2 , pages 151– 162.

Aalst W.M.P. van der, T. Weijters, L. Maruster (2004). Workflow mining: Discovering process models from event logs. In *IEEE Transactions on Knowledge and Data Engineering*, volume 16, Issue: 9 , pages 1128–1142.

Alur R., C. Courcoubetis, D.L. Dill (1993). Model-checking in dense real-time. *Information and Computation*, 104(1):2–34.

Bernard B., M.Bidoit, A.Finkel, F.Laroussinie, A.Petit, L.Petrussi, Ph.Schnoebelen (2001). *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer-Verlag.

Chuang Lin , Yang Qu (2004). Temporal inference of workflow systems based on time petri nets: Quantitative and qualitative analysis. In *International Journal of Intelligent Systems*, volume 19, Issue 5, pages 417 – 442.

Cook J.E., A.L.Wolf (1998). Discovering models of software processes from event based data. In *ACM Transactions on Software Engeneering and Methodology*, volume 7, Issue: 3 , pages 215– 249.

Cook J.E., A.L.Wolf (1999). Software process validation: Quantitatively measuring the correspondence of a process to a model. In *ACM Transactions on Software Engeneering and Methodology*, volume 8, Issue: 2 , pages 147– 176.

Davulcu H., M.Kifer, C.R. Ramakrishman, I.V. Ramakrishman (1998). Logic based modeling and analysis of workflows. In *ACM Symposium on Principles of Database Systems (PODS), Seattle*.

Dongen B.F. van, W.M.P.van der Aalst (2004). Emit: A process mining tool. In *25th International Conference on Applications and Theory of Petri Nets*.

FourEyes (2004). Retrieved January 15, 2005, from http://www.tpis.com.au/products/fe/default.htm.

JavaCC. A parser/scanner generator for java (2004). Retrieved January 15, 2005, from https://javacc.dev.java.net.

Pnueli, A. (1981). The temporal semantics of concurrent programs. In *Theoretical Computer Science*, volume 13, Issue: 1 , pages 46– 60.

ProM framework. TU/e (2004). Retrieved January 15, 2005, from http://is.tm.tue.nl/research/processmining/.

Schimm G. (2002). Process miner - a tool for mining process schemes from event based data. In *Lecture Notes in Computer Science*, volume 2424 , pages 525–528.

	Data	WorkflowModelElement	EventType	Timestamp	Originator
1	Data	A	schedule	1899-12-30T00:04:00.0 00+01:00	Jan
2	Data	A	start	1899-12-30T00:05:00.0 00+01:00	Sara
3	Data	A	suspend	1899-12-30T00:07:00.0 00+01:00	Sara
4	Data	A	resume	1899-12-30T00:09:00.0 00+01:00	Sara
5	Data	A	suspend	1899-12-30T00:14:00.0 00+01:00	Rit
6	Data	A	resume	1899-12-30T00:15:00.0 00+01:00	Rit
7	Data	A	complete	1899-12-30T00:17:00.0 00+01:00	Rit
8	Data	C	schedule	1899-12-30T00:18:00.0 00+01:00	Jan

Figure 1: Example of a process instance.

Figure 2: A screen shot of the wizard-tool for constructing of properties.