An Improvement of Requirement-Based Compliance Checking Algorithm in Service Workflows

Wattana Viriyasitavat
Business Information Technology Division, Department of Statistics, Faculty of Commerce and Accountancy, Chulalongkorn University, Bangkok, Thailand

Andrew Martin
Department of Computer Science, University of Oxford, Oxford, United Kingdom

***Abstract*—**This paper presents an improvement of requirement-oriented compliance checking algorithm to support trust-based decision making in service workflow environments. The proposed algorithm is based on our previous progressive works on (1) Service Workflow Specification language (SWSpec) serving as a formal and uniformed representation of requirements, and (2) the algorithm based on Constrained Truth Table (CTT), specifically developed for compliance checking for the Composite class of SWSpec. However, CTT algorithm practically suffers from high complexitywhich is $O(|S||V|2^{|V|})$, where $|V|$ is the number of services presented in a workflow, and $|S|$ is the size of a SWSpec formula to be checked. In this paper, we improve algorithm CTT by using Exclusive Disjunctive Normal Form (EDNF) as a new data structure that reduces the time complexity in the average case to $O(|S||V|^2)$. Finally, the performance comparison between these two approaches is conducted.

Keywords—Service, Workflow, Compliance Checking

I. Introduction

Service workflowshave received much interest in the past decades. Nowadays, they appear in several forms ([1], [2], and [3]). For example, within an organization, services are used as a building block to streamline and automate business processes to improve efficiency and scalability. In decentralized collaborative environments such as Grids [4], Virtual Organizations (VO), and Cloud Computing, services become a fundamental element for collaborations. Despite their wide range of applications, services still suffer from the lack of an agreed and standard in requirement representation.

Formal methods provide rich specification languages ([5], [6], and [7]), to express such requirements, modeling languages to abstract systems to be verified, and algorithms. To achieve automatic reasoning that is needed to facilitate scalability, dynamicity, and security in large-scale open environments, three essential elements are required: (1) a modeling language in which workflows can be logically abstracted to represent structure of services, tasks, and their relationships, (2) a specification language as a formal and uniformed representation of requirements, and (3) compliance checking algorithms to ascertain that the services satisfy such requirements [8]. All of the three elements have been comprehensively addressed in our previous work. The workflow to be verified is modeled by Service Workflow Net (SWN), with the introduction of control connectives for structure formulation; the requirements are formally represented by SWSpec formulas [9]; and the compliance checking algorithm are developed based on CTT [10]. In this paper, we improve the algorithm CTT by using EDNF as a new data structure that reduces the time complexity from $O(|S||V|2^{|V|})$ to $O(|S||V|^2)$in the average case.

The rest of this paper is organized as follows. Section II presents our previous work on SWN, SWSpec, and algorithm CTT. Section III explains the process of simplifying SWSpec formulas, which will be used for EDNF compliance checking algorithm. In section IV, algorithm EDNF is presented. Then, we conduct the analysis with comparison of performance between CTT and EDNF algorithms in Section VI. Section VII presents some related works, and thenthe last section concludes and discusses potential future research.

II. Background

In this section, the information regarding (1) our workflow modeling, SWN, (2) SWSpec formulas, and (3) CTT algorithm is described shown in Table I, II, and III, respectively. Please refer to [9] and [10] for more information and justifications.

TABLE I
SWN Definition

Def. 1 A Petri Net is a labeled Place/Transition Net, i.e., a 7-tuple $\mathcal{M} = (P, T, R, f, i, o, l)$, where

1) P is a set of places (representing services),
2) T is a set of transitions (representing tasks),
3) $R \subseteq (P \times T) \cup (T \times P)$ represents directed flows,
4) $f: (P \to (\psi \times \{split, join\})) \cup (T \to (\psi \times \{split, join\})$ is a function containing a workflow structure formula (ψ) with either a split or a join type. A formula contains three types of connectives (AND, OR, and XOR).
5) $l: P \to A \cup \{\tau\}$ is a labelling function where A is a set of properties, and τ denotes a null value. It is used for labelling a service with attributes (properties).

Service-join (or service composition) is defined as a set of possible services that can be activated for a task execution.
Service-split (or service separation) identified a set of services that can be triggered after the execution is done.

ISBN: 978-981-07-2683-6

TABLE II
SWSPEC GRAMMARS

Def. 2SWSpec grammars presented in Backus–Naur Form (BNF) form below

Path Formulas	$R ::= S \mid {\sim}R \mid R \wedge R \mid R \vee R \mid \exists_t \odot R \mid \exists_t \lozenge R \mid \exists_t \square R \mid \exists_t[R \Cup R] \mid \exists_t[R \dot{\cup} R] \mid \forall_t \odot R \mid \forall_t \lozenge R \mid \forall_t \square R \mid \forall_t[R \Cup R] \mid \forall_t[R \dot{\cup} R]$
Composite Formulas	$S ::= \mathcal{F}E_tZ \mid \mathcal{P}E_tZ \mid \mathcal{F}A_tZ \mid \mathcal{P}A_tZ \mid S \,\&\, S \mid S \,\|\, S$ (Quantifier)
	$Z ::= (s,t,o,A) \mid Z \sqcap Z \mid Z \sqcup Z \mid Z \boxplus Z$(Property)
Direction Formulas	$W ::= \mathcal{H}R \mid \mathcal{B}R \mid {\sim}W \mid (W \wedge W) \mid (W \vee W)$

Path operators

$\odot r_1$	*Next*	It allows requirements to be specified that along a selected path the immediate connected service must satisfy.
$\lozenge\, r_1$	*Future*	It allows requirements to be specified that one service (property) must be present along a selected path.
$\square r_1$	*Global*	It allows requirements to be specified that all services (globally) along a path must satisfy.
$r_1 \Cup r_2$	*Strong Until*	It allows requirements to be specified that r_1 must hold until r_2. It also demands r_2 to hold in the future.
$r_1 \dot{\cup} r_2$	*Weak Until*	It is just like the Strong until except r_2 is not required to hold in the future.
$\wedge, \vee, \sim$	And, Or, Not	These operators are similar to CTL
$\exists_t$	*For Some Path*	There must be some paths among a set of connected services through a task t.
$\forall_t$	*For All Path*	It allows requirements to be specified for all paths through a task t.

Composite operators

$\mathcal{F}$	*Forward*	It addresses the properties of target services in service-split type through a task t indicated by E_t or A_t.
$\mathcal{P}$	*Previous*	It addresses the properties of target services service-join type through a task t indicated by E_t or A_t.
E_t	*Composite For Some*	It indicates that at least one services in service-join or service split type through a task t must be satisfied.
A_t	*Composite For All*	It indicates that all services in service-join or service split type through a task t must be satisfied.
$\sqcap$	*Strong Composite Conjunction*	It allows requirements to be specified in service-join or service split type. It must be preceded by the same Composite quantifier operator through a task t indicated by E_t or A_t.
$\sqcup$	*Strong Composite Disjunction*	It indicates that in one or both of the properties in service-join or service split type It must be preceded by the same Composite quantifier operator through a task t indicated by E_t or A_t .
$\boxplus$	*Composite Exclusive Disjunction*t	It indicates that only one property of services is presented in an execution. These services are restricted to a task indicated by E_t or A_t.
&	*WeakComposite Conjunction*	As weaker than $\sqcap$, It is not restricted to be preceded by the same Composite quantifier operator, for example, $\mathcal{P}E_tS \,\&\mathcal{F}A_tS$.
||	*WeakComposite Disjunction*	As weaker than $\sqcup$, It is not restricted to be preceded by the same Composite quantifier operator, for example, $\mathcal{P}E_tS \,\|\mathcal{F}A_tS$.
ε	*Null*	It represents an empty notion meaning that no property of services is required.
(s,t,o,A)	*Atomic*	This is an extensible elementwhere s, t and o are name, type, and owner. A set of attributes A is used to indicate service properties.
!	*CompositeNegation*	It indicates the negation of an expression.

Direction operators

$\mathcal{F}$	*Forward*	It addresses the properties of target services in service-split type through a task t indicated by E_t or A_t.
$\mathcal{P}$	*Previous*	It addresses the properties of target services service-join type through a task t indicated by E_t or A_t.
$\mathcal{H}$	*Henceforth*	It allows requirements to be specified in the forward direction from the preceding to succeeding along a workflow path.
$\mathcal{B}$	*Backward*	It allows requirements to be specified in the backward direction from a succeeding service to a preceding along a workflow path
$\wedge, \vee, \sim$	*And, Or, Not*	They are similar to the definitions in propositional logic.

Def. 3Satisfiability Relations: Let ω be a SWSpec formula and $\mathcal{M}$ be an SWN:
1) $\mathcal{M} \vDash \omega$ is *satisfied*, where ω is satisfied $\mathcal{M}$,
2) $\mathcal{M} \vdash \omega$ is *partially satisfied*, where$\mathcal{M} \vDash \omega$ and $\mathcal{M}' \subseteq \mathcal{M}$;
3) $\mathcal{M} \nvDash \omega$ is *unsatisfied*, whereω does not satisfy of $\mathcal{M}$.

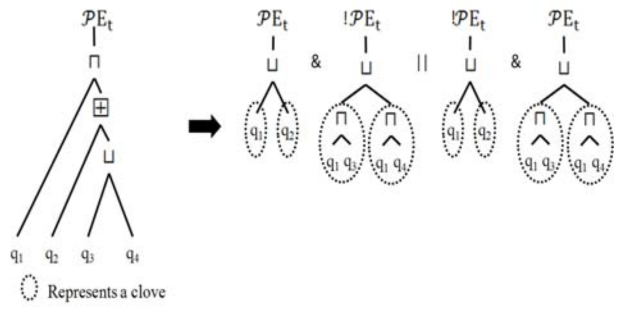


Figure 1. The Clove Tree of a Formula $\mathcal{P}E_t(q_1 \sqcap (q_2 \boxplus (q_3 \sqcup q_4))$ and its Transformation with the Notion of Cloves of $\mathcal{P}E_t(q_1 \sqcap q_2)\ \&\ !\,\mathcal{P}E_t((q_1 \sqcap q_3) \sqcup (q_1 \sqcap q_4))\ \|\ !\,\mathcal{P}E_t(q_1 \sqcap q_2)\ \&\mathcal{P}E_t((q_1 \sqcap q_3) \sqcup (q_1 \sqcap q_4))$

III. PREPROCESSING SWSPEC FORMULAS

For simplicity, any SWSpec formula is translated into a simpler form. It can be pre-processed until the property part of a Composite formula includes only $\sqcup$and $\sqcap$ in order. The notions of cloves and clove trees (from [10]) below represent the transformed formula (see Figure 1).

Def. 4 (Clove): Given a Composite formula, a clove is defined as a set of atomic propositions linked by $\sqcap$ operators, or a single atomic proposition if no such operator is involved.

Def. 5 (Clove Tree): A clove tree is the representation of a Composite formula with the quantifier part as a root, the second level is $\sqcup$ operator, and the leaves are cloves.

The formula $\mathcal{P}E_t(q_1 \sqcap (q_2 \boxplus (q_3 \sqcup q_4))$in Figure 1 can be interpreted as follows (see algebraic properties in Table IV).

- *Initial form*
  $\mathcal{P}E_t(q_1 \sqcap (q_2 \boxplus (q_3 \sqcup q_4))$
- *Applying Distributive Property*
  $\equiv \mathcal{P}E_t((q_1 \sqcap q_2) \boxplus (q_1 \sqcap (q_3 \sqcup q_4))$
- *Applying $\boxplus$ Elimination*
  $\equiv \mathcal{P}E_t(q_1 \sqcap q_2)\ \&\ !\mathcal{P}E_t(q_1 \sqcap (q_3 \sqcup q_4))\ ||$
  $!\mathcal{P}E_t(q_1 \sqcap q_2)\ \&\mathcal{P}E_t(q_1 \sqcap (q_3 \sqcup q_4))$
- *Applying Distributive Property*
  $\equiv \mathcal{P}E_t(q_1 \sqcap q_2)\ \&\ !\mathcal{P}E_t((q_1 \sqcap q_3) \sqcup (q_1 \sqcap q_4))\ ||$
  $!\mathcal{P}E_t(q_1 \sqcap q_2)\ \&\mathcal{P}E_t((q_1 \sqcap q_3) \sqcup (q_1 \sqcap q_4))$

This transformation makes the algorithm simpler because the absence of $\boxplus$ allows us to circumvent the check between OR and $\boxplus$ that can be done indirectly by $\sqcap$ and $\sqcup$.

TABLE III
REDUCTION RULES AND ALGORITHM CTTSAT

Reduction Rules
Suppose that ϕ and ω are two sub-SWSpec formulas
Rule 1: For ϕ AND ω, ϕ and ω must both be true or false.
Rule 2: For ϕ XOR ω, ϕ and ω cannot both be true.
Rule 3: The results of the evaluation cannot be false.

Algorithm CTT

```
1:  FunctionCTTSAT( CTT, V)
        // CTT =a constrained truth table object
        // CTT.R = a set of rows
        // V= a set of workflow variables
2:  Begin
3:    For each $r_i \in$ CTT.R
4:      For each variable $v \in V$ in that row
5:        If$A_t$ is presented
6:        If for all $v \in V$ marked with 1 $\models$ any clove
7:        $r_i$ = satisfied;
8:        End if
9:        End if
10:       If$E_t$ is presented,
11:         If some $v \in V$ marked with 1 $\models$ any clove
12:           $r_i$ = satisfied;.
13:         End if
14:       End if
15:     End for
16:   End for
17:   If for all $r_i$ = satisfied
18:     Returnsatisfied;
19:   Else If some $r_i$ = satisfied
20:     Returnpartially satisfied;
21:   Else
22:     Returnunsatisfied;
23:   End if
24: End Function
```

TABLE IV
PERFORMANCE COMPARISON

Name	*Initial Form*	*Transformed Form*
Distributive Property	$Z_1 \sqcap (Z_2 \boxplus Z_3)$	$(Z_1 \sqcap Z_2) \boxplus (Z_1 \sqcap Z_3)$
	$Z_1 \sqcap (Z_2 \sqcup Z_3)$	$(Z_1 \sqcap Z_2) \sqcup (Z_1 \sqcap Z_3)$
$\boxplus$ *Elimination*	$E_t(Z_1 \boxplus Z_2)$	$(E_tZ_1\&!E_tZ_2)\ ||\ (!E_tZ_1\&E_tZ_2)$
	$\Delta' A_t(Z_1 \boxplus Z_2)$	$(A_tZ_1\&!E_tZ_2)||(!E_tZ_1\&A_tZ_2)$

(see the complete properties in [10])

IV. EDNF

Normal form is an alternative choice in representing Boolean functions in a more concise. A formula with the same number of variables is much more compact comparing to CTT. For this reason, effective compliance checking can be developed. Terms in EDNF are all variables that are connected by AND connectives, and XOR connectives are used to connect between terms. If all terms are true, the result is *satisfied*. If some are true, the result is *partially satisfied*; otherwise, it is *unsatisfied*.

To circumvent the check between OR and $\boxplus$as mentioned earlier, any workflow formula is presented by the combination of AND and XOR, while OR can be transformed as follows:

$$\text{A OR B} = \text{A XOR B XOR (A AND B)}$$

Def. 6 (EDNF): An SWN formula is EDNF if it is an exclusive disjunction of terms where each term is a conjunction of literals.

A. *Algorithm EDNFSAT*

Assume that all SWN formulas are presented in the form of EDNF. The complexity of compliance checking depends on (1) the number of the occurrence of workflow variables (services), (2) the number of connectives, and (3) reasoning algorithms. One of the most efficient algorithms employs a binary tree data structure to represent a workflow formula. Leaf nodes are workflow variables while the upper nodes represent workflow connectives. The graphical explanation of the complexity calculation is illustrated in Figure 2. The operations of this algorithm can be understood by the following steps (the pseudo code for EDNFSat is presented in Table V).

1) Each leaf node is marked with *satisfied*, *unsatisfied*, or $unknown_Q$, if it satisfies, does not satisfies, or partly satisfies with any clove in a clove tree. For instance, if a node contains a property q_2 and there is a clove $cl_i = (q_1 \sqcap q_2)$ which is part of a clove tree of a Composite formula $\mathcal{P}A_t(q_1 \sqcap q_2)$, we mark the node with $unknown_Q$ where the subscripted $Q = \{(q_2, cl_i)\}$. Note that the *unknown* marking occurs only when $\sqcap$ is presented in a clove. The set Q indicates a set of partly satisfied properties. For example, $Q = \{(q_1, cl_1), (q_2, cl_2), \ldots\}$.
2) For each upper AND node traversing up towards the top AND node in an SWN tree formula, if E_t is presented in the clove tree,
   a) if at least one lower node is marked with *satisfied*, we mark the upper AND node with *satisfied*,
   b) if one node is $unknown_Q$ and another is marked with *unsatisfied*, we mark the upper AND node with $unknown_Q$,
   c) if two lower nodes with $unknown_Q$ marking are combined which results in satisfying any clove, we mark the upper AND node with *satisfied*. If not, it is marked with $unknown_Q$;
   d) otherwise, we mark the upper AND node with *unsatisfied*, and repeat until traversing to the top AND node.
   e) Go to step 4.
3) For each upper AND node traversing up towards the top AND node in an SWN tree formula, if A_t is presented in the clove tree,
   a) We mark the node with *satisfied*, if the lower nodes are the combination of (1) both marked with *satisfied*,, or (2) *satisfied* and $unknown_Q$,

TABLE V
ALGORITHM EDNFSAT

```
1:Function EDNFSAT(SWND)  // SWND is a EDNF tree representing
     workflow formulas in the form of clove tree
2:Q = a set of properties of unknown status to satisfy any clove;
3:V = SWND.V  // a set of workflow variables;
4:M = all presented connectives;
5:Begin
6:     For each $v_i \in V$,
7:        If $v_i \vDash$ any clove
8:           $v_i$ = satisfied;
9:        Else if $v_i$ partly complies with any clove
10:          $v_i = unknown_Q$;
11:       Else $v_i$ = unstisfied;
12:       End if
13:    End for
14:    For each $m_i \in M$ and $m_i$ = AND  //Assume that $m_i$ is chosen in order
          from low-to-high layer of the SWND
15:       If $A_t$ is presented
16:          If ($m_i$.left = satisfied AND $m_i$.right = satisfied) OR
               ($m_i$.left = $unknown_Q$ AND ($m_i$.right = satisfied) OR
               ($m_i$.left = satisfied AND $m_i$.right = $unknown_Q$)
                 $m_i$ = satisfied;
17:          Else If ($m_i$.left = $unknown_Q$ AND $m_i$.right = $unknown_Q$)
18:             $m_i$ = satisfied;
19:          Else $m_i$ = unsatisfied;
20:       End if
21:End if
22:       If $m_i$ is TOP AND NODE
23:          If $m_i = unknown_Q$
24:             $m_i$ = unsatisfied;
25:          End if
26:       End if
27:       If $E_t$ is presented
28:          If $m_i$.left = satisfied OR $m_i$.right = satisfied
29:             $m_i$ = satisfied;
30:          Else If ($m_i$.left = $unknown_Q$ AND $m_i$.right = $unknown_Q$)
31:             $m_i$ = satisfied;
32:          Else If ($m_i$.left = $unknown_Q$ OR $m_i$.right = $unknown_Q$)
33:             $m_i = unknown_Q$;
34:          Else $m_i$ = unsatisfied;
35:          End if
36:          If $m_i$ is TOP AND NODE
37:             If $m_i = unknown_Q$
38:                $m_i$ = unsatisfied;
39:             End if
40:          End if
41:       End if
42:    End for
43:    For each $m_i \in M$ and $m_i$ = XOR  ////Assume $m_i$ is chosen in order
          from low-to-high layer of the SWND
44:       If $m_i$.left = satisfied AND $m_i$.right = satisfied
45:          $m_i$ = satisfied;
46:       Else If ($m_i$.left = unsatisfied & $m_i$.right = unsatisfied)
47:          $m_i$ = unsatisfied;
48:       Else $m_i$ = partially satisfied;
49:       End if
50:End for
51: End function
```

b) if both lower nodes with *unknown$_Q$* status are combined to satisfy a clove, we mark the node with *satisfied*, if not, it is marked with *unknown$_Q$*.
c) if an *unsatisfied* mark is presented, we immediately mark the top AND node with *unsatisfied*, and go to step 4.
d) Repeat until traversing to the top AND node.

4) If the top AND node is *unknown$_Q$*, we remark it with *unsatisfied*.
5) In an upper node representing XOR connective,
   a) if all lower nodes are marked with *satisfied*, it is also marked with *satisfied*,
   b) if one of its lower nodes is marked with *satisfied* and another with *unsatisfied* or *partially satisfied*, it will be marked with *partially satisfied*;
   c) otherwise it is marked with *unsatisfied*.
6) Repeat step 5 until reaching the root node where we can determine the final result.

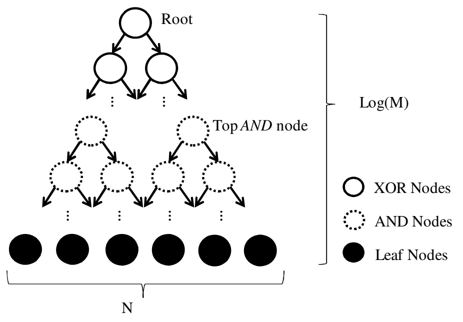


Figure. 2. Graphical Representation of Efficiency Complexity Calculation

B. Analysis of EDNFSAT

Assume again the checking operation between a clove and a leaf node occupies one time unit, the best efficiency evaluation of this form is $O(|C||T||V||M|)$where $|C|$, $|T|$, $|V|$, and $|M|$ are the number of cloves, clove trees, workflow variable, and workflow connectives respectively. In a concise form,$|C||T|$ can be reduced to $|S|$, representing a size of SWSpec tree, such that the time complexity is presented as $O(|S||V||M|)$. In the worst case scenario, the maximum number of occurrence M of workflow variables V can be calculated by the following equation.

$$N = \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n}$$

According to Taylor's approximation,

$$(1+x)^a = \binom{a}{a}x^a + \binom{a}{a-1}x^{a-1} + \cdots + \binom{a}{0}x^0$$

If $x = 1$ we have

$$\binom{a}{a} + \binom{a}{a-1} + \cdots + \binom{a}{0} = 2^a$$

such that,

$$N = \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n} = 2^n - 1$$

As a result, the computational complexity of this form is $O\left(|S||V|2^{|V|}\right)$. However, it is important to look at the average occurrence of V that can be computed as the following equation.

$$average = E(n \times \binom{n}{n} + (n-1) \times \binom{n}{n-1} + \cdots + 0 \times \binom{n}{0})$$

$$\text{average} = \frac{n}{2}$$

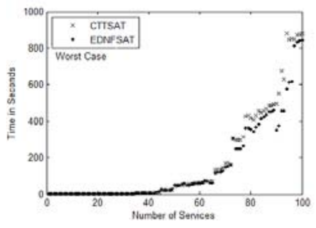
Therefore, the average time complexity of EDNFSat is $O(|S||V|^2)$. The performance comparison between EDNFSAT and CTTSAT is presented in Table VI.

TABLE VI
PERFORMANCE COMPARISON

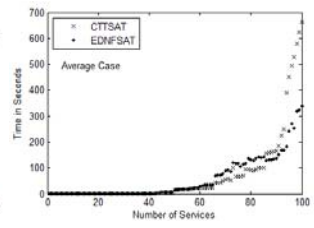
Models	*Checking Time*	*Average*
Constrained Truth Table	$O(\lvert S\rvert\lvert V\rvert 2^{\lvert V\rvert})$	$O(\lvert S\rvert\lvert V\rvert 2^{\lvert V\rvert})$
EDNF	$O(\lvert S\rvert\lvert V\rvert 2^{\lvert V\rvert})$	$O(\lvert S\rvert\lvert V\rvert^2)$

V. PERFORMANCE EVALUATION

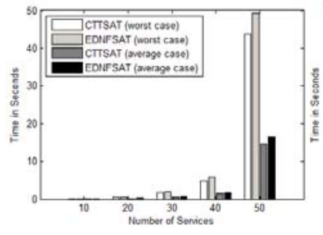
To confirm the applicability, we have developed a prototype to validate our framework. All functions are written in MATLAB to demonstrate the proof of concept and performance comparison between two approaches. The system runs on a Windows 7, Intel® Core™ i5-2435M CPU @ 2.40 GHz, 4 GB RAM, 64-bit Operating System.We design the experiment to evaluate time performance when 10, 50, and 100 services are involved. The result in Figure 3(a) shows that the performance between algorithm EDNFSAT and CTTSAT in the worst case scenario is similar. However, EDNFSATruns faster in the average case (see Figure 3(b) andFigure 3(c) and (d) for the comparison in bar graph). This corresponds to the theoretical evaluation in Table VI.



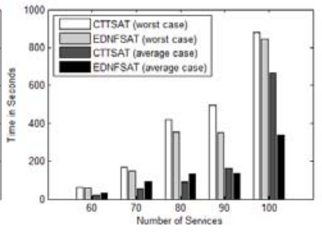
(a)



(b)



(c)



(d)

Figure. 3.Performance Evaluation between CTTSAT and EDNFSAT

VI. RELATED WORKS

After Model checking is first introduced [11], it has been extended to cover wider domains beyond the specific systems modeled by Finite State Machine. It has spread across many areas ranging from verification between business processes and contacts [12], policy-based compliance checking for trust [13], logic-based verification [14] to hardware and software component testing at very low level. In our essence, we intend to apply the concept of model checking for compliance checking in the service workflow domain and requirements specification.

VII. CONCLUSION

This paper presents algorithm EDNFSAT for compliance checking between SWSpec formulas and service workflow. It improves the existing algorithm CTTSAT that specifically deals with Composite class of SWSpec. We conduct the experimentto compare between these two approaches. The primary advantage of this algorithm is that in average case, time complexity for checking operation is reduced into polynomial. In practice, the checking process can be locally computed; each involved service is only to verify if its own requirements. Furthermore, since SWSpec formulas are independent from each other, using parallel computing will significantly improve the overall performance.For future work, we plan to develop the tracer to indicate the conflict points, if any, in both SWSpec and SWN, and to provide the counter example of this conflict.

REFERENCES

[1] L. Xu. Enterprise Systems: State-of-the-Art and Future Trends, IEEE Transactions on Industrial Informatics. vol.7, no.4, pp.630-640, 2011.

[2] W. Viriyasitavat, and A. Martin, A Survey of Trust in Workflows and Relevant Contexts, In Communications Surveys Tutorials, IEEE, vol. pp, no. 99, pp. 1 −30, 2011, DOI (10.1109/SURV.2011.072811.00081).

[3] W. Viriyasitavat, and A. Martin, Formal Trust Specification in Service Workflows, In Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on, pp. 703 −710, 2010.

[4] T. Fei, Z. Dongming, H. Yefa, and Z. Zude. Resource Service Composition and Its Optimal-Selection Based on Particle Swarm Optimization in Manufacturing Grid System, IEEE Transactions on Industrial Informatics. vol.4, no.4, pp.315-327 2008.

[5] W. Viriyasitavat, Modeling Delegation in Requirements-Driven Trust Framework, In IEEE Congress on Services- I, pp. 522-529, 2009.

[6] W. Viriyasitavat, and A. Martin, Formalizing Trust Requirements and Specification in Service Workflow Environment, In Runtong Zhang, Jos Cordeiro, Xuewei Li, Zhenji Zhang and Juliang Zhang, editors, ICEIS, vol. 3, pp. 196–206. SciTePress. 2011.

[7] W. Viriyasitavat, and A. Martin, In the Relation of Workflow and Trust Characteristics, and Requirements in Service Workflows, In Abd Manaf, Azizah, Akram Zeki, Mazdak Zamani, Suriayati Chuprat and Eyas El−Qawasmeh, editors, Informatics Engineering and Information Science, vol. 251 of Communications in Computer and Information Science, pp. 492−506, Springer Berlin Heidelberg. 2011.

[8] E.M. Clarke, B.H. Schlingloff, Model Checking, Chapter 21 in Handbook of Automated Reasoning, Elsevier Science Publishers, 2000.

[9] W. Viriyasitavat, A. Martin, and L. Xu, SWSpec: The Requirements Specification Language in Service Workflow Environments, In Industrial Informatics, IEEE Transactions on. vol. PP, no. 99, pp. 1. 2012, DOI (10.1109/TII.2011.2182519).

[10] L. D. Xu, W. Viriyasitavat, P. Ruchikachorn, and A. Martin, Using Propositional Logic for Requirements Verification of Service Workflow, In Industrial Informatics, IEEE Transactions on. vol. PP, no. 99, pp 1, 2012, DOI (10.1109/TII.2012.2187908)

[11] E. M. Clarke, E. A. Emerson, Synthesis of synchronization skeletons for branching time temporal logic, in ‘Pro. Workshop on Logic of Programs’, Vol. 131 of LNCS, Springer, Yorktown Heights, NY, 1981.

[12] G. Governatori, Z. Milosevic, S. Sadiq, Compliance checking between business processes and business contracts. In the 10th Intl. Enterprise Distributed Object Computing Conf (EDOC 2006). IEEE Press, pp. 221-232, 2006.

[13] M. Blaze M, J. Feigenbaum, A. D. Keromytis, KeyNote: Trust Management for Public-Key Infrastructures, in Security Protocols Intl. Workshop, Cambridge England, 1998.

[14] S. Kerrigan, K.H. Law, Logic-based RegulationCompliance-Assistance.” The 9th Int’l Conf. on ArtificialIntelligence and Law, Scotland, UK, pp. 126-135, 2003.