# An Observation-based Algorithm for Workflow Matching

Kais Klai[1], Samir Tata[2] and Issam Chebbi[2]

[1] Technical University Eindhoven
[2] GET/INT France

**Abstract.** This work is in line with the *CoopFlow* approach dedicated for inter-organizational workflow cooperation that consists of workflow advertisement, workflow interconnection, and workflow cooperation. To support interconnection, we propose in this paper a efficient algorithm for workflow matching[3].

## 1 Introduction

Research on workflow management has focused on inter-organizational issues and much has been achieved so far [4, 1]. Problems to be encountered on this research include mainly autonomy of workflow processing, flexibility, and lack of arbitrary workflow support. To deal with these issues, we have developed the *CoopFlow* approach [3] that consists of three steps: workflow advertisement, interconnection, and cooperation. In fact, for building an inter-organizational workflow, each organization has to advertise, within a common registry, a description of its offered and required activities within their workflows. For workflow interconnection, each organization identifies its partners using a matching mechanism. For matching workflows, we propose in this paper a new algorithm using symbolic observation graphs (*SOG* for short) [2]. The rest of this paper is organized as follows. Section 2 describes informally our novel method of workflow abstraction based on *SOGs*. Section 3 presents a efficient algorithm to workflow matching. Using workflow matching, Section 4 shows how inter-organizational workflow is formed. Conclusion and perspectives are presented in Section 5.

## 2 Workflow Abstraction

An inter-organizational workflow can be considered as the cooperation of several local workflows. Each one has two types of activities (transitions): cooperative activities that interact with other workflows and local activities that perform local actions. In order to set up cooperation, workflows have to be abstracted to preserve privacy, and advertised into a registry to be found and interconnected to partners' workflows. Workflows are reprenseted by Wf-nets [5]: A WF-net is a Petri net that has one *source* place and

[3] This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.000.504.

Klai K., Tata S. and Chebbi I. (2006).
An Observation-based Algorithm for Workflow Matching.
In *Proceedings of the 4th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pages 193-197
DOI: 10.5220/0002500801930197
Copyright © SciTePress

one *sink* place and all its nodes (places or transitions) should be on some path from *source* to *sink*. To define workflow abstraction and matching, we need to introduce some definitions. Let $\sigma$ be a sequence of transitions ($\sigma \in T^*$). The projection of $\sigma$ on a set of transitions $X \subseteq T$ (denoted by $\sigma_{\lfloor X}$) is the sequence obtained by removing from $\sigma$ all transitions that do not belong to $X$. A sequence $\sigma = t_1 t_2 \ldots t_n$ over transitions is said to be accepted if $i$ (resp. $o$) is in set of input (resp. output) places of $t_1$ (resp $t_n$) and $\sigma$ can be executed by the workflow. The language $L(W)$ of a workflow $W$ is the set of all accepted sequences and the projection function is extended to $L$ as follows: $L_{\lfloor X} = \{\sigma_{\lfloor X}, \sigma \in L\}$.

To abstract workflows, we use *SOG* introduced in [2] as an abstraction of the *reachability marking graph* of a given Petri net within a model checking approach. The building of the *SOG* is guided by the set of the cooperative transitions. Such activities are called *observed*, since they interact with other workflows, while the other transitions are *unobserved*. Then, the *SOG* is defined as a deterministic graph where each node is a set of markings linked by unobserved sequences of transitions and each arc is labeled with an observed transition. Nodes of the *SOG* are called *meta-states* and may be represented and managed efficiently by using Ordered Binary Decision Diagram (OBDD techniques) [6].

The *SOG* technique is suitable for abstracting workflows for many reasons: First, the *SOG* allows one to represent the language of the workflow projected on the cooperative transitions i.e. the local behaviors are hidden. The second reason is that such an abstraction is suitable for checking whether two workflows represented by their *SOG* can be interconnected (see section 3). Finally, the reduced size of the *SOG* (in general) could be an advantage when one plans to store and manage a big number of workflows abstractions in a same registry. For sake of space, we do not give the *SOG* building algorithm, we refer the reader to [2] for more details about the *SOG* technique.

## 3 An Algorithm for Workflow Matching

Given a Wf-net $W_1$ and a registry of potential partners for $W_1$, we discuss in this section the selection criteria allowing to choose of a Wf-net $W_2$ in the registry as partner of $W_1$. Such criteria are based on the observable behavior of $W_1$, i.e. its behavior on the cooperative transitions, which must match with the observable behavior of $W_2$. Each cooperative transition $t$ of Wf-net $W$ is represented by a tuple $t = \langle name, type, msg \rangle$ s.t.(1) the $name$ attribute of $t$ is the label associated to $t$, (2) the $type$ attribute of $t$ is a boolean variable and it says whether $t$ is supposed to receive a message ($t.type = 1$), or to send a message ($t.type = 0$), and (3) the $msg$ attribute of $t$ represents the semantic description of the message (using a common ontology) $t$ has to send or to receive.

In order to check whether there exists a correspondence between two cooperative transitions $t_1$ and $t_2$ belonging to two different Wf-nets, we need to compare these transitions with respect to their attributes. Two attributes are taken in account: $type$ and $msg$. For instance, if $t_1$ is a reception transition then $t_2$ must be a sending transition and both transitions have to match on the semantic of the exchanged message. We denote by $t_1.msg \equiv t_2.msg$ the fact that messages of $t_1$ and $t_2$ deal with the same data type and semantics. Now, if $t_1.type = \neg(t_2.type)$ and $t_1.msg \equiv t_2.msg$, then we say that $t_1$ matches with $t_2$ (and vice versa) and denote this relation by $t_1 \sim t_2$.

The following hypothesis is important for the remaining part of the paper. It says that, within the same Wf-net $W_1$, if a cooperative transition occur in a Wf-net more than once then these occurrences are executed in an exclusive way. In this case we denote by $\{t\}$ the set of occurrences of a cooperative transition $t$ in a Wf-net. Let $\langle W_1, m_1\rangle$ be a marked Wf-net and let $T_1$ be its set of cooperative transitions. Then $\forall t_1 \in T_1$, $\forall \sigma = \alpha t_1 \alpha' t_1$, where $\alpha$ and $\alpha' \in T_1^*$, then $\sigma \notin L(W_1, m_1)$ (H).

To define formally the fact that a Wf-net $W_1$ can cooperate with a given Wf-net $W_2$, we need to introduce a renaming procedure $\mathcal{L}_{W_1}$ and define a *cooperation candidate property*.

Let $W_1$ and $W_2$ be two Wf-nets and let $T_1$ and $T_2$ be their sets of cooperative transitions. The renaming procedure $\mathcal{L}_{W_1}$ associated to $W_1$ is defined as follows:
$\mathcal{L}_{W_1}(W_2) = \forall t_2 \in T_2$ if $\exists t_1 \in T_1$ s.t. $t_1 \sim t_2$ then $t_2.name := t_1.name$.

Let $\langle W_1, m_1\rangle$ and $\langle W_2, m_2\rangle$ be two marked Wf-nets: $\langle W_2, m_2\rangle$ is said to be a candidate for cooperation with $\langle W_1, m_1\rangle$ iff $L_{\lfloor T_1}(\langle W_1, m_1\rangle) \subseteq L_{\lfloor T_2}(\langle \mathcal{L}_{W_1}(W_2), m_2\rangle)$.

To check the above inclusion of projected language, we use the *SOG* of $W_1$ and $W_2$. Actually, the Wf-net $W_2$ would be an effective candidate to cooperate with $W_1$ if the language induced by the *SOG* of $W_1$ is included in that induced by *SOG* of $\mathcal{L}_{W_1}(W_2)$. The inclusion test Algorithm 1 works on the fly i.e. the building of the synchronized product between the involved *SOGs* can be stopped at any moment as soon as the inclusion is proved unsatisfied. When the synchronized product is entirely built, one deduce that the inclusion holds. The parameters of this algorithm are the *SOG*s $SoG_1 = \langle s_0, S_1, E_1\rangle$ and $SoG_2 = \langle s'_0, S'_1, E'_1\rangle$ of $(W_1, m_1)$ and $(\mathcal{L}_{W_1}(W_2), m_2)$ respectively. $s_0$ (resp. $s'_0$) is the initial meta-state of $SoG_1$ (resp. $SoG_2$), $S_1$ (resp. $S_2$) its set of meta-states and $E_1$ (resp. $E_2$) its set of arcs. The data structures used by

**Algorithm 1** ($L(SoG_1 = \langle s_0, S_1, E_1\rangle) \subseteq L(SoG_2 = \langle s'_0, S_2, E_2\rangle)$)?

```
State $s_1$, $s_2$, $s'_1$, $s'_2$;
Set of transition $f_1$, $f_2$;
stack $st$;
$s_1 = s_0$; $s_2 = s'_0$;
$f_1 = Out(s_0)$, $f_2 = Out(s'_0)$;
if $f_1 \neq \emptyset$ and $f_2 \neq \emptyset$ then
    if ($Names(f_1) \not\subseteq Names(f_2)$) then
        return false;
    end if
end if
$Synch = \{\langle s_1, s_2\rangle\}$;
$st$.Push($\langle s_1, s_2, f_1\rangle$);
repeat
$st$.Pop($\langle s_1, s_2, f_1\rangle$);
for $t \in f_1$ do
    $s'_1 = Img(s_1, t)$; $s'_2 = Img(s_2, t)$
    if $\langle s'_1, s'_2\rangle \notin Synch$ then
        $f_1 = Out(s'_1)$; $f_2 = Out(s'_2)$;
        if $f_1 \neq \emptyset$ and $f_2 \neq \emptyset$ then
            if ($Names(f_1) \not\subseteq Names(f_2)$) then
                return false;
            end if
            $Synch = Synch \cup \{\langle s'_1, s'_2\rangle\}$;
            $st$.Push($\langle s'_1, s'_2, f_1\rangle$);
        end if
    end if
end for
until $st == \emptyset$;
return true;
```

Algorithm 1 are a table *Synch* and a stack *st*. *Synch* is used to store the states of the synchronized product non completely treated. An item of *st* is a tuple $\langle s_1, s_2, f_1\rangle$ composed of a reachable meta-state of $(W_1, m_1)$, a reachable meta-state of $(\mathcal{L}_{W_1}(W_2), m_2)$ and a set of cooperative transitions enabled from both nodes. Moreover, three functions are used *Out()*, *Img()* and *Names()*. *Out()* is applied to a node of the *SOG* and return the set of transitions labeling its output edges. *Img()* is applied to a state $s_1$ and a transition $t$ (enabled in this node) and returns the reached state. *Names()* is applied to a set of transitions $f$ and returns the set of transitions' names.

## 4 Workflow Interconnection

The interconnection of two workflows $W_1$ and $W_2$ satisfying the cooperation candidate property is performed by completing $W_1$ (resp. $W_2$) by an interface which connect its cooperative transitions to those of $W_2$ (resp. $W_1$) via some buffer places.

Let $W_1 = \langle P_1, T_1, Pre_1, Post_1\rangle$ and $W_2 = \langle P_2, T_2, Pre_2, Post_2\rangle$ be two Wf-nets. Let $C_1$ and $C_2$ be the cooperative transitions of $W_1$ and $W_2$ respectively. Then the interface workflow is represented by a Petri net $Int_{12} = \langle B, T, Pre, Post\rangle$ defined as follows: (1) $B$ is a set of buffers. For all subsets $\{t_1\}$ and $\{t_2\}$ of occurrences of transitions $t_1$ and $t_2$ in $W_1$ and $W_2$ resp., such that $t_1 \sim t_2$, there exists an associated buffer place $b \in B$, (2) $T = C_1 \cup C_2$ and (3) for each place $b$ and associated occurrences subsets $\{t_1\}$ and $\{t_2\}$ of cooperative transitions: if $t_1.type = 1$ (resp. $t_1.type = 0$) then $Pre(t_1) = b$ and $Post(t_2) = b$ (resp. $Post(t_1) = b$ and $Pre(t_2) = b$).

Now one can use the interface workflow in order to interconnect the associated Wf-nets $W_1$ and $W_2$. This interconnection is simply performed by composing these three Petri nets by fusion of shared transitions (the cooperative transitions). The obtained net can be simply transformed, so that it satisfies the Wf-net properties, by adding a new source place (resp. a sink place) and connect it to the two existing source places (resp. sink places) via a new transition.

## 5 Conclusion and Perspectives

In line with the *CoopFlow* approach that consists of three steps: workflow advertisement, interconnection, and cooperation, we showed, in this paper, how to abstract the behavior of workflows and presented an efficient algorithm for matching and interconnecting Wf-net partners. Currently, we are implementing algorithms for workflow abstraction and interconnection in order to be integrated into the *CoopFlow* framework.

## References

1. A. Van Dijk. Contracting workflows and protocol patterns. In *Proceedings BPM*, Eindhoven, The Netherlands, June 2003.
2. S. Haddad, J-M. Ilié, and K. Klai. Design and evaluation of a symbolic and abstraction-based model checker. In Farn Wang, editor, *ATVA*, volume 3299 of *LNCS*. Springer, 2004.
3. I. Chebbi, S. Dustdar, and S. Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data and Knowledge Engineering Journal*, 56:2, 2006.

4. W.-M.-P. van der Aalst and M. Weske. The p2p approach to interorganizational workflows. *13th Int. Conf. on Advanced Information Systems Engineering*. Springer-Verlag, 2001.
5. W.-M.-P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.
6. I. Wegener. *Branching programs and binary decision diagrams: theory and applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.