Automated Discovery of Workflow Models for Hospital Data

Laura Maruster[a] Wil van der Aalst[a] Ton Weijters[a]

Antal van den Bosch[b] Walter Daelemans[c]

[a] Eindhoven University of Technology, I&T
[b] Tilburg University, ILK/Computational Linguistics
[c] Antwerpen University, Computational Linguistics

Abstract

Workflow nets, a subclass of Petri nets, are known as attractive models for analysing complex business processes. In a hospital environment, for example, the processes show a complex and dynamic behavior, which is difficult to control; the workflow net which models such a complex process provides a good insight into it, and due to its formal representation offers techniques for improved control. We provide a method of which the main advantage consists in discovering the workflow Petri nets automatically from process logs. We illustrate the functioning of our method on simulated hospital process logs, containing information about which medical actions took place over time. We show that our method is able to discover processes whose underlying models are acyclic and sound WF nets, involving parallel, conditional and sequential constructs. We argue that solutions have to be found for cyclic and free-choice/non-free-choice workflow nets.

1. Introduction

The managing of complex business processes of today's organizations calls for the development of powerful information systems, able to control and support the flow of work. These systems are called *Workflow Management Systems* (WfMS), where a WfMS is generally thought of as "a generic software tool which allows for definition, execution, registration and control of workflows" [1]. Petri nets are attractive as the underlying model language for WfMS: they have a precise mathematical formalism, they provide a graphical image of the investigated processes, they can express all important features of the WfMS, and they can be subject to many analysis techniques [1]. Petri nets used for workflow process definition are called, for short, *workflow nets* (WF nets).

Nevertheless, the process of workflow design takes a lot of time and the resulting models are often incomplete and not realistic. In the hospital domain, for example, some projects were developed to support patient WfMS, built on guidelines [2,3]. Specifying the clinical practice in terms of guidelines, which provides the process logic,

presupposes the presence of expert knowledge. The knowledge extraction process consumes a lot of time and may not reveal exactly the clinical practice. In contrast, hospital processes are highly dynamic and subject to change. Thus, the WfMS has to be flexible enough and able to capture all changes that occurred in a short time frame. The issue of flexible workflows has drawn a lot of attention and many research efforts have been spent in this direction [4,5]. In this sense, a WfMS able (i) to acquire process knowledge automatically and (ii) to incorporate changes quickly, will be more desirable in the hospital domain and other dynamic workflow environments.

In this paper we present a discovery procedure that, given a workflow log for a business process (a "history" which contains information about how events took place, chronologically ordered), builds the associated workflow net. The obtained workflow net can be used for analysing, redesigning and managing the investigated process. The idea of discovering models from process logs was previously investigated in contexts such as software engineering processes and also workflow management. Cook and Wolf propose in [6] three methods for process discovery in case of software engineer processes: a finite state-machine method, a neural network and a Markov approach. Their methods focuses on sequential processes. Also, they provided some specific metrics for detection of concurrent processes, like entropy, event type counts, periodicity and causality [7]. Herbst and Karagiannis used a hidden Markov model in the context of workflow management, in case of sequential processes [8] and concurrent processes [9]. The drawback of Herbst and Karagiannis results is that workflow net constructs like AND/OR splits and joins are not depicted. In the mentioned works, the focus was on identifying the dependency relations between events. Our goal is to detect explicitly (i) flow and (ii) concurrency/choice relations between events. Moreover, and this is our main contribution, we try to discover complete WF nets, not only the dependency relations. WF nets are very useful for analysing the considered process.

Our goal is to develop an automatic discovery and analysis tool for providing insight in real world situations. We take the modeling of logistical processes of medical actions in a hospital as a case study. Medical treatment in a hospital often requires involvement of different specialties. Because of an aging population that shows complex syndromes and the increased specialisation of medical technology, the number of different specialties involved in treatments increases. We will call a patient that requires different specialties for her/his treatment a *medical multidisciplinary patient* (MMP). Especially, we focus on the patient category with *vascular* medical problems, because the managing of vascular patients involves the biggest number of specialties. An efficient coordination of such MMPs may imply restructuring the organization of hospitals into specialty-oriented units, while care for patients is not constrained within the boundaries of one of those units [10]. The problem that arises here is how to build these specialty-oriented units. For this purpose, we need to investigate the underlying patient flow process and to decide on the organizational structure of the hospital. In this paper, we concentrate only on developing a tool for investigating the workflow. In future works, we want to apply this tool to our real hospital MMP data.

The structure of the paper is as follows. Section 2 addresses some basic theoretical aspects of WfMS and WF nets. In Section 3 we present our process discovery method. Section 4 summarizes the experiments done for testing our method. A discussion of the status and shortcomings of the present approach, and future directions of our work are presented in Section 5.

2. Theoretical aspects of Workflow models. WF nets

Because of their good theoretical foundation, Petri nets have been used successfully to model and analyse processes from many domains, like for example, software and business processes. Workflow processes can be modeled by WF nets, which form a subclass of Petri nets [1]. A Petri net is a directed graph with two kinds of nodes, *places* and *transitions*, where *arcs* connect a place to a transition or a transition to a place. Each place can contain zero, one or more tokens. The state of a Petri net is determined by the distribution of tokens over places. A transition can fire if each of its input contains tokens. If the transition fires, i.e. it executes, it takes one token from each input place and puts one token on each output place.

Workflows are *case* oriented, which means that each *activity* executed in the workflow corresponds to a case. In our hospital domain, a *case* corresponds with a patient and an *activity* corresponds with a medical activity. The process definition of a workflow assumes that a partial order exists between activities, which establishes which activities have to be executed in what order. Referring to the Petri net formalism, workflow activities are modeled as transitions and the causal dependencies between activities are modeled as places and arcs.

A WF net has one *source* place (i.e. a place without incoming arcs), that represents the beginning of the case in the workflow, and a *sink* place (i.e. a place without outgoing arcs), which represents the end of the case in the workflow. Moreover, each transition and place in the WF net is on a path from source place to sink place. The existence of one token in the source place will correspond to the situation in which the patient enters for the first time into the hospital and needs to be registered. One token in the sink place corresponds to the situation in which the patient registration to that hospital is ended.

The routing in a workflow assumes four kind of routing constructs: *sequential*, *parallel*, *conditional* and *iterative* routing [1]. *Sequential* routing concerns ordered causal relationships between tasks. For example, if we consider tasks A and B, we have a sequential routing construct when task B is executed only after task A is executed. *Parallel* routing is used when the order of execution is less strict. A parallel routing is modeled by *AND-split* and *AND-join* blocks. An AND-split corresponds to a transition with two or more output places and an AND-join corresponds to a transition with two or more input places. *Conditional* routing allows the modeling of a choice between two or

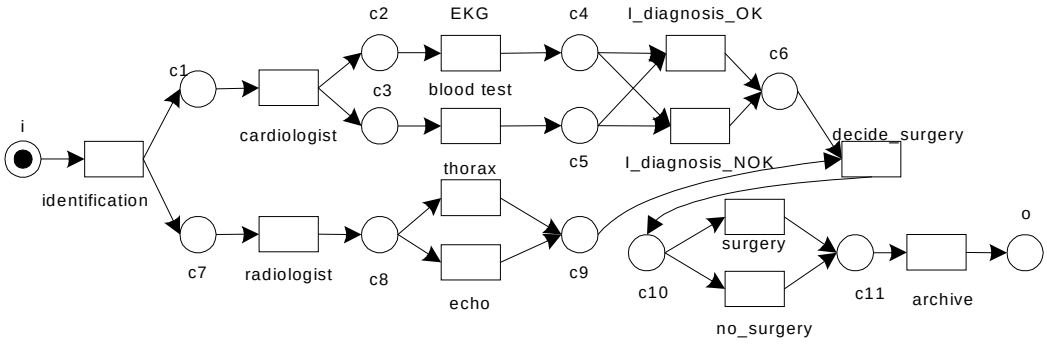

Figure 1. A Petri net for handling a medical complaint.

more alternatives. To express the conditional construct, *OR-split* and *OR-join* blocks are used. An OR-split corresponds to two or more alternative output transitions and an OR-join corresponds to two or more alternative input transitions. Figure 1 illustrates the workflow process definition for handling a medical complaint. In this figure we can identify the following routing constructs: transitions *identification* and *cardiologist* are AND-splits, transitions *I_ diagnosis_OK*, *I_diagnosis_NOK* and *decide_surgery* are AND-joins, places *c4, c5, c8* and *c10* are OR-splits and places *c6, c9* and *c11* are OR-joins.

Our method assumes to discover WF nets which are *sound*. A WF net is *sound* iff: (i) we can always complete a case, (ii) after the completion of an activity, no work is left behind in the workflow and (iii) there are no dead activities, i.e. activities that cannot be achieved [1]. Of a special interest are *free-choice* Petri nets. A Petri net is *free-choice* iff for every two transitions that share the same input place, they match [1].

3. Process discovery method

The present work concentrate on discovering WF models from *workflow logs* that can be represented as WF nets. The workflow log (*WL*) consists of traces of events, related to cases. A hospital workflow log contains medical activities performed to each patient, as they happen over time. In case of our multidisciplinary patients (MMP) problem, *WL*s contain sequences of identifiers. Each identifier corresponds to a department (i.e. cardiology, radiology, vascular laboratory, etc.) visited by the patient at a specific point in time. Formally, we define a workflow log as following:

Definition 1 (Workflow Log *WL*):
Let A be a set of observable actions. A *workflow log* is a set of sequences over A, i.e., $WL \subseteq A^*$, where A^* is the set of all sequences that are composed of zero or more actions of A.

Given the WF net process from Figure 1, an example of a patient trace is given below:

```
identification, cardiologist, blood_test, EKG, radiologist, echo,
I_diag_OK, decide_surgery, archive.
```

Consider now the WF net from Figure 1, which is a simplified model of handling a medical complaint. A patient who enters the hospital is first identified with his/her hospital card. Afterwards, he/she has to visit a cardiologist and radiologist, in any order. After visiting the cardiologist, it is necessary to perform both *EKG* and *blood test*, in any order. Depending on *EKG* and *blood test* results, a first diagnosis is made. The first diagnosis can relate to a vascular disease problem (*I_diagnosis_NOK*) or not (*I_diagnosis_OK*). When the first diagnosis results and either the *thorax* or *echo* results become available, a decision on surgery is made. Depending on this decision, the surgery is performed or not. Finally, this particular patient case is archived.

In this work we address the following research question: given a *WL*, discover the underlying WF net that generates all events in *WL* and distinguishes its routing constructs. Our discovery techniques is based on the following definitions:

Definition 2 *(α, β, first, last)*:

For any sequence $s \in A^*$, with $s = (a_1, a_2, ..., a_n)$, we have:

- $\alpha(s) = \{a_1, a_2, ..., a_n\}$ is the alphabet of *s*,
- $\beta(s) = \{(a_1, a_2), (a_2, a_3), ..., (a_{n-1}, a_n)\}$ is the set of pairs of *s*,
- $first(s) = a_1$,
- $last(s) = a_n$.

The previous definition formalizes the concept of trace of events from A^* (the set of all potential traces). Identifiers a_i represent events in traces. Traces have a first and a last event. Our algorithm focuses on pairs of events from $WL \subseteq A^*$. Definition 3 states the possible relations that can exist between the elements of a pair. Namely, if there is a sequence in *WL*, where event *y* appears after *x* and there is no sequence where *y* appears before *x*, then the pair $(x,y) \in R^{\rightarrow}$. If there is a sequence in *WL*, where event *x* appears after *y* and also *y* appears after *x*, then $(x,y) \in R^{\leftrightarrow}$.

Definition 3 ($R^{\rightarrow}$, $R^{\leftrightarrow}$):

Let *WL* be a workflow log over A. We consider the following relations:

$$R^{\rightarrow} = \{(x, y) \in A \times A \mid \exists s \in WL, (x, y) \in \beta(s) \wedge \forall s \in WL, (y, x) \notin \beta(s)\},$$

$$R^{\leftrightarrow} = \{(x, y) \in A \times A \mid \exists s \in WL, (x, y) \in \beta(s) \wedge \exists s \in WL, (y, x) \in \beta(s)\}.$$

The first step for discovering the WF net is to build the net N_O^{WL}, as it is formally stated in the following definition.

Definition 4 (N_0^{WL}):

Let *WL* be a workflow log, and $R^{\rightarrow}$ and $R^{\leftrightarrow}$ as defined. Then there is

$N_O^{WL} = (P, T, F)$, where

$$T = \bigcup_{s \in WL} \alpha(s),$$

$$P = R^{\rightarrow} \cup \{i, o\},$$

$$F = \{(i,t) \mid \exists s \in WL, t = first(s)\} \cup \{(t,o) \mid \exists s \in WL, t = last(s)\} \cup$$
$$\cup \{((x,y),t) \in R^{\rightarrow} \times T \mid y = t\} \cup \{(t,(x,y)) \in T \times R^{\rightarrow} \mid x = t\}.$$

Considering pairs $(x,y) \in R^{\rightarrow}$ from *WL*, a net N_0^{WL} can be constructed. For all pairs of events $(x,y) \in R^{\rightarrow}$, which have in common the same *x*, an arc will link the node *x* with all nodes *y*. The node *x* will be the starting point and the nodes *y* will be the ending points. For all pairs of events $(x,y) \in R^{\rightarrow}$ which have in common the same *y*, an arc will link all nodes *x* with node *y*. The nodes *x* will be the starting points and the node *y* will be the ending point. For the rest of pairs of events $(x,y) \in R^{\rightarrow}$, an arc will link node *x* with node *y*. The set of all nodes will form *T* (the set of transitions in the net N_0^{WL}). Additionally, on each arc that connects two nodes, a place from set *P* (the set of places in the net N_0^{WL}) will be placed. There are two special places, *i* and *o*; *i* links all events from set *first(s)* and *o* links all events from set *last(s)*. All arcs that connect nodes will form the set *F* in the net N_0^{WL}.

Definition 5 (*merge*):

Let $N=(P,T,F)$ be a Petri net and $X \subseteq P$ a set of places. Then

$merge(N, X) = (P', T, F')$, with

$$P' = (P \setminus X) \cup \{P_X\},$$

$$F' = F \cap (P' \times P') \cup \{(P_X, t) \mid t \in T \wedge \exists p \in X, (p,t) \in F\} \cup \{(t, P_X) \mid t \in T \wedge \exists p \in X, (t,p) \in F\}$$

This formalizes the operation of merging two places into one place. The ingredients of the merging operation are one "source" transition, two arcs that link the source transition with the two places, and two more arcs that link each place with one of the two "destination" transitions. The result of merging is one "source" transition, one arc that links the "source" transition with the merged place, and two arcs that link the merged place with "destination" transitions. The merge operation works analogously in case of two "source" transitions and one "destination" transition.

Definition 6 (N^{WL}):

Let *WL* be a workflow log, N_0^{WL} and $R^{\rightarrow}$ as defined. Take N_0^{WL} and merge all places that have non-concurrent input or output transitions. The resulting net is N^{WL}. Formally:

$$M(N) := \{(P_1, P_2) \in P \times P \mid (P_1 \bullet \cap P_2 \bullet \neq \emptyset \wedge \exists t_1 \in \bullet P_1, \exists t_2 \in \bullet P_2, (t_1, t_2) \notin R^{\leftrightarrow}) \vee$$
$$\vee (P_1, P_2) \in P \times P \mid (\bullet P_1 \cap \bullet P_2 \neq \emptyset \wedge \exists t_1 \in P_1 \bullet, \exists t_2 \in P_2 \bullet, (t_1, t_2) \notin R^{\leftrightarrow})\}$$

for any Petri net $N = (P, T, F)$.

In the above definition, $\bullet P_i$ means the set of input transitions for place P_i, and $P_i \bullet$ means the set o transitions sharing P_i as an input place. The decision to merge places that have non-concurrent input or output transitions is taken if the input transitions are in relation $(t_1,t_2) \notin R^{\leftrightarrow}$, or if the output transitions are in relation $(t_1,t_2) \notin R^{\leftrightarrow}$, too.

For building the WF net N^{WL}, we have to apply the following algorithm:

$N:=(P,T,F)= N_0^{WL}$;
while $M(N) \neq \emptyset$
do $(P_1,P_2) :\in M(N)$
$N:=merge(N,\{P_1,P_2\})$
od
$N^{WL}:=N$.

This algorithm states that first, the net N_0^{WL} is built (see Definition 4). Second, for each pair of transitions $(P_1,P_2) \in M(N)$, the decision to merge the related places is made.

To illustrate our algorithm, we use the WF net presented in Figure 1. Suppose that we have a *WL* corresponding to the WF from Figure 1, and we want to discover the underlying WF net. The basic idea of our approach supposes three main steps:

- Identify the pairs $(x,y) \in R^{\rightarrow}$ and $(x,y) \in R^{\leftrightarrow}$ (Definition 3);

In *WL*, the pairs $(x,y) \in R^{\rightarrow}$ are those pairs of events that always occur in the same order, for example: (*identification, cardiologist*), (*identification, radiologist*), (*cardiologist, EKG*), (*cardiologist, blood_test*), (*I_diagnosis_OK, decide_surgery*), (*I_diagnosis_NOK, decide_surgery*), (*radiologist, thorax*), (*radiologist, echo*), (*thorax, decide_surgery*), (*echo, decide_surgery*), and so on. The pairs $(x,y) \in R^{\leftrightarrow}$ are pairs of events that can happen in any order, like, for example, (*cardiologist, radiologist*), *(EKG, blood_test*), (*cardiologist, thorax*), and so on.

- Using pairs $(x,y) \in R^{\rightarrow}$, build the net N_0^{WL} (Definition 4);

This step assumes to connect all pairs $(x,y) \in R^{\rightarrow}$ and to insert a place between the connected transitions. The result of this step is shown in Figure 2.

- Apply the algorithm for building the WF net N^{WL}, which supposes to merge all places that have non-concurrent input or output transitions (Definition 5 and 6).

In Figure 2, because $(I\_diagnosis\_OK, I\_diagnosis\_NOK) \notin R^{\leftrightarrow}$, we have to merge places 4 with 4'. The same states for 5 with 5', 6 with 6', 8 with 8', 9 with 9', 10 with 10' and 11 with 11'. After merging, we get the WF net from Figure 1.

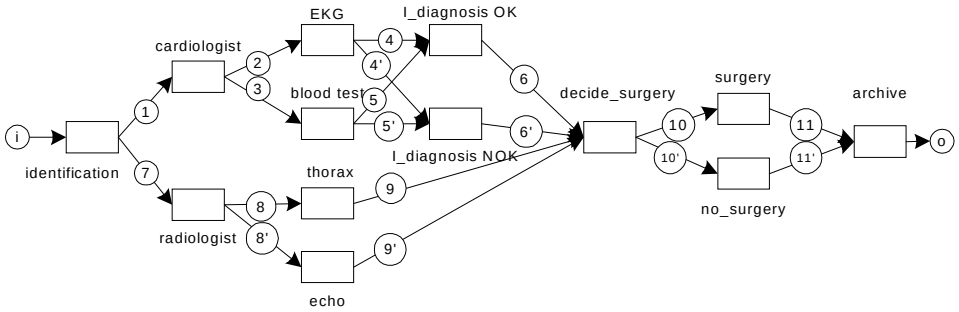

Figure 2. The N_0^{WL} net for handling medical complaints.

4. Experiments

We tested our method, considering five different sound and acyclic WF nets. The WF nets are similar with the example given in Figure 1, i.e. they contain between 10-12 transitions, involving parallel, conditional and sequence routing constructs. For each WF net, we generated random workflow logs with 500 event traces.

In four experiments, the WF net built with our method is equivalent with the initial WF net. In Figure 3 a) is presented the initial WF net of one of our experiment, for which we generated the *WL*. In Figure 3 b) is showed the N_0^{WL} net. In the N_0^{WL} net, after merging places 2 and 2', 7 with 7' and 10 with 10', we obtained the initial WF net. In other three experiments, the considered WF nets have comparable structure and complexity.

However, in the fifth experiment involving a WF net which is actually not free-choice, the method was not able to find the complete WF net.

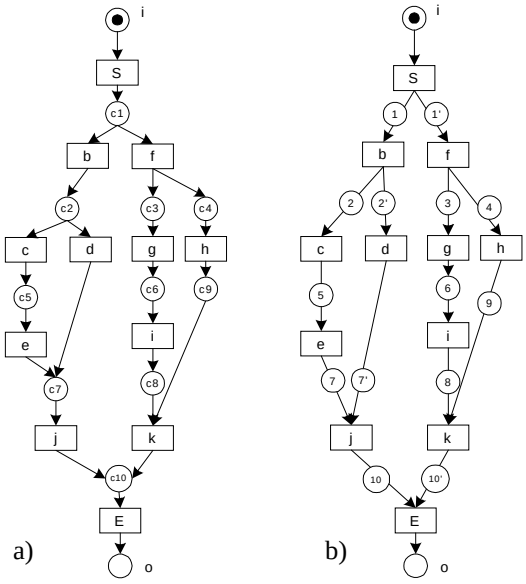

Figure 3. One example of sound and acyclic WF net. After merging places in b), the resulting WF net is identical with the initial WF net from a).

5. Discussion and future work

In this work we presented a method for discovering the underlying WF net from a process workflow log. The experiments done with five different WF nets show that in case of sound and acyclic WF nets, involving parallel, conditional and sequential constructs, our method is able to rebuild them correctly from its workflow log. However, the current technique does not work for all kind of WF nets, as one experiment involving a non-free-choice net showed. We have to experiment more for determining the types of WF nets where our method is applicable and to provide theoretical foundations.

Our research is preliminary; we plan to do future research in several directions. First, we want to extend our method to cyclic WF nets (we investigated the performance of our method in the acyclic case). Especially in the medical domain, follow-up visits to the same specialist happen very often, thus the detection of the iteration in a process is necessary. Second, we want to improve our method such as to be applicable in case of free-choice and also non-free-choice WF nets.

Our final goal is to have a robust tool which is able to discover and further analyse a complex and completely unknown process, namely the logistical flow of medical multidisciplinary patients (MMP). Such a tool coupled with WfMS that offer generic modeling and enactment capabilities can provide an efficient way of analysing and managing today's business organisations.

References

[1] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *J. of Circuits, Systems, and Computers*, 8(1): 21-66, 1998.

[2] L. Dazzi, C. Fassino, R. Sarraco, S. Quaglini, M. Stefanelli – *A patient workflow management system built on guidelines.* Proc. of AMIA 97, 146-150, Nashville, TN, 1997.

[3] S. Andersen – *PATMAN: Patient Workflow Management System*, EU Project, January 1998.

[4] W.M.P. van der Aalst and S. Jablonski, editors. *Flexible Workflow Technology Driving the Networked Economy*, Special Issue of the *International Journal of Computer Systems, Science, and Engineering*, volume 15, number 5, 2000.

[5] M. Klein, C. Dellarocas, and A. Bernstein, editors. *Adaptive Workflow Systems*, special issue of the journal of *Computer Supported Cooperative Work*, volume 9, numbers 3-4, 2000.

[6] J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data, *ACM Transactions on Software Engineering and Methodology*, 7(3):215-249, 1998.

[7] J.E. Cook and A.L. Wolf. Event-Based Detection of Concurrency. In *Proceedings of the Sixth International Symposium on the Foundations of Software Engineering* (FSE-6), Orlando, FL, November 1998, pp. 35-45.

[8] J. Herbst. A Machine Learning Approach to Workflow Management. In 11th European Conference on Machine Learning, volume 1810 *of Lecture Notes in Computer Science*, pages 183-194, Springer, Berlin, Germany, 2000.

[9] J. Herbst. Dealing with Concurrency in Workflow Induction In U. Baake, R. Zobel and M. Al-Akaidi, *European Concurrent Engineering Conf.*, SCS Europe, Ghent, Belgium, 2000.

[10] Geerhard de Vries. Multidisciplinaire Samenwerking rondom Vaatpatiënten. Tussenrapportage, Prismant, May 2001.