Model Fragmentation for Distributed Workflow Execution: A Petri Net Approach

Wei Tan and Yushun Fan

Department of Automation
Tsinghua University, 100084 Beijing, P.R. China
tanwei@mails.tsinghua.edu.cn, fanyus@tsinghua.edu.cn

Abstract. Workflow is the key technology for business process automation, while distributed workflow is the solution to deal with the decentralized nature of workflow applications and the performance requirements of the whole system. In this paper the architecture of distributed workflow execution is given, and the centralized model called CWF-net, which is based on colored Petri net, is presented. Based on the centralized model, a novel model fragmentation algorithm is proposed. This algorithm partitioned the centralized model into fragments by duplicating the places shared by transitions which are executed in different sites. The behavioral equivalence between the CWF-net and resulted fragments are guaranteed by the extended firing rules. Then the correctness of the fragmentation algorithm is discussed, the correctness criteria comprises completeness of the fragmentation, completeness of each fragment, and the behavioral equivalence after fragmentation. Finally the future research work is pointed out.

1 Introduction

Workflow is the key technology for the coordination of various business processes, such as loan approval and customer order processing [1]. By setting up the process model and enacting it in the workflow server, it can help to streamline the business process, deliver tasks and documents among users, and monitor the overall performance of the process.

Traditional workflow systems are often built upon the client/server architecture, in which a single workflow server will be responsible for the operation of the overall process. But this sort of centralized systems may bring about many disadvantages. First of all, in the internet age, the process itself may be distributed among geographically dispersed business partners, therefore the workflow applications are inherently distributed. Secondly, the reliability of the centralized system cannot be guaranteed since there can be a single point of failure. Last but not the least, the performance of the centralized system may be drastically degraded when there are too many process instances to handle.

To solve the problem that centralized workflow systems cannot overcome, many distributed workflow systems are designed from different approaches. The Exotica system [2] developed by IBM Almaden Research center proposes a completely distributed architecture, in which the information among servers is transferred by the persistent message queue. And by this means, the reliability of the system is highly enhanced.

The Mentor Project [3] of the University of Saarland developed a traceable and scalable workflow architecture. A formal model called state chart is used for workflow specification, and a model partitioning method is proposed by mapping a centralized state chart to distributed ones, each comprises a single state. A theorem has showed that this mapping is a homomorphism between state charts, which means that the transformation preserves the behavior of the original specification.

The Dartflow [4] project has shown the use of the mobile agents in distributed workflow execution. In Dartflow, the workflow model is fragmented dynamically, i.e., when one task is completed, the remaining model, which has not been executed, is partitioned, and the partitions are carried by mobile agents and sent to different sites which are responsible for them.

Workflow specification, or process model provides the information of the process, and it's the basis to computationally support business processes automation. In distributed workflow execution paradigm, the whole process must be partitioned into fragments before execution, and each fragment is designated to a workflow server at which it is to be executed. However, the research works in this field mainly focus on the design of the system architecture and the implementation based on specific communication mechanism. As far as we know, little attention has been paid to the formal method of model fragmentation. Mentor project proposes an approach for model partition, but it just separates states from a state chart, and does not consider which ones are to be executed at the same server.

In this paper we propose a Petri net based approach for model fragmentation. First we discuss the distributed workflow execution architecture, through which we explain how a workflow process is handled by multiple servers. Our centralized process model is based on the well known colored Petri net, and we present the algorithm to partition the centralized model into fragments. The partition is done by duplicating the places connecting different transitions which are not to be executed at the same server. An example is also given to illustrate the algorithm. Another contribution of this paper is that the correctness of our fragmentation algorithm is analyzed in three aspects, i.e., completeness of the fragmentation, completeness of each fragment and the behavioral equivalence after fragmentation. Finally the future work is presented and a conclusion is drawn.

2 Distributed Workflow Execution Architecture

In traditional workflow system, there is a single workflow server to take charge of the operation of the overall process, so the workflow engine must communicate with each task agent, deliver necessary information and retrieve the outcome of a task, as it shows in Fig. 1 (a).

While in the distributed paradigm, there are many workflow *servers* (also called *sites*, for the reason that the servers are often geographically dispersed), and each task is designated to be executed in one of them. After each task is labeled with a server identifier, the process model is separated into several fragments, each coordinated by one server. For example, in Fig. 1 (b), first the centralized model is given. Then *task 1* to *task 8* are labeled with the server name of which they are designated to (*task 1* and *task 2* are designated to *server 1*, *task 4* is designated to *server 2*, *task 3* and *task 5* are designated to *server 3*, *task 6, 7* and *8* are designated to *server 4*). After labeling, it's natural to see that the model is divided into four fragments, i.e., f_1, f_2, f_3 and f_4, and each fragment consists of one or more adjacent tasks which are designated to the same

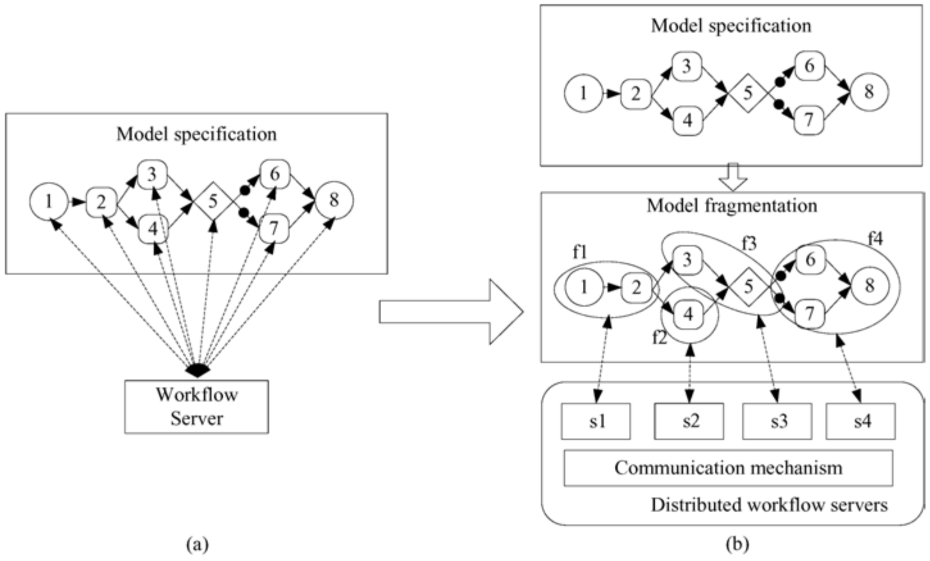

Fig. 1. Architecture for Centralized and Distributed Workflow Execution

server. Note that servers must communicate when the execution is transferred from one sever to others.

3 Centralized Workflow Model

The workflow model used in this paper is an extension of the WF-net [5] proposed by Van der Aalst. WF-net is a special class of Petri net, which prevails in workflow modeling field because of its graphic nature and theoretical foundation. The WF-net has shown its advantages in workflow model verification [5], whereas it's hard to be used to model a real business process because it cannot depict the relevant data and the precondition for tasks. In the meantime, it's well known that colored Petri net [6] can be used in modeling data and condition for tasks, so we combine the WF-net and the colored Petri net for the modeling of workflow in this paper. Below is the definition of CWF-net, which is used for centralized workflow modeling.

Definition 1 (CWF-Net). *A colored Petri net CPN (Σ, P, T, A, N, C, G, E, I) is a colored WF-net (CWF-net) if and only if:*

(i) (P, T, A, N) forms a WF-net, i.e.,

a) CPN has two special places: i and o. Place i is a source place: $^\bullet i=\Phi$. Place o is a sink place: $o^\bullet=\Phi$;

b) If we add a transition tn to CPN which connects place o with i (i.e., $^\bullet tn=\{o\}$ $tn^\bullet=\{i\}$), then the resulting Petri net is strongly connected.

(ii) $\Sigma=\{ID\times DataList\}$, $ID=\{id|id$ is the identifier of process instances$\}$
$DataList=\{d_1,d_2,\ldots,d_n|d_1,d_2,\ldots,d_n$ are workflow relevant data items$\}$

(iii) $\forall p\in P$, $|C(p)_{MS}|=1$; $\forall a\in A$, $E(a)=I(i)$, i is the source place.

From the definition above we know that a CWF-net is a kind of colored Petri net with the structure characteristics of a WF-net. The color sets for all places are the same, and the token identifies the process instance as well as the relevant data items. There is at most one token in each place, each arc expression represents the initial token in source place, and the guard functions on transitions are used to model the conditional dependencies between tasks. Fig. 2 is an example of CWF-net, which represents the centralized model in Fig. 1. Note that the guard functions at t_6 and t_7 are mutually exclusive so that only one transition is enabled when p_6 holds one token.

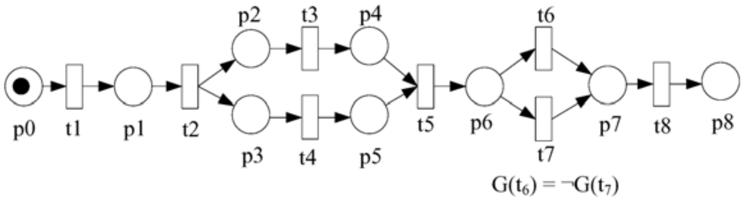

Fig. 2. An Example of a CWF-net

We call a colored WF-net (Σ, P, T, F, C, G, E, I) sound if and only if (P, T, A, N) is a sound WF-net, in this paper it's assumed that before fragmentation the centralized model has been verified for its correctness, therefore we assume that the CWF-net is sound.

4 Fragmentation Algorithm

There are two kinds of fragmentation scheme, the static one and the dynamic one. In this paper we only consider the static fragmentation method, that is, the whole model is fragmented when the workflow process is ready to instantiate.

The basic idea about static fragmentation is that when the centralized model is given and each task is labeled with the server (site) id, the model is fragmented. The method for decomposition (or fragmentation) is inspired by the idea that if the adjacent activities are planned to execute on the same site, then they can be put into a single fragment and sent to the site they are designated to. In a CWF-net, activities are represented by transitions and connected by places. When we decompose the whole process model into smaller parts that can be executed at different sites, we must assure that each resulted fragment is started and ended by places, just as the original CWF-net does.

For the sake of simplicity, we define some functions which will be used throughout this paper.

Definition 2 (Some Functions). *In a CWF-net (P, T, A, N)*

$\forall t \in T$, function site(t) returns the id of the site at which task t is executed.

$\forall T' \subseteq T$, $site(T') = \{s \mid s = site(t), t \in T'\}$

$\forall p \in P$ function dup(p, k) returns a set of duplicated places of p, i.e., $\{p_1, p_2, \ldots, p_k\}$

$\forall p_d \in dup(p, k) = \{p_1, p_2, \ldots, p_k\}$, $dups(p_d) = \{p_1, p_2, \ldots, p_k\}$

For example, in Fig. 2, $site(t_1) = s_1$, $site(t_8) = s_4$; $dup(p_3, 2) = \{p_{31}, p_{32}\}$; $dups(p_{31}) = dups(p_{32}) = p_3$.

Fragmentation Algorithm

Algorithm 1 shows our method for model fragmentation. Through Algorithm 1, a CWF-net is divided into a set of colored Petri nets, and each represents a fragment that can be executed in one site. When we impose Algorithm 1 to the CWF-net in Fig. 2, four fragments are obtained, as it shows in Fig. 3. Note that in Fig. 3, the fragments are encircled by dashed rectangles and duplicated places are grouped by ellipses.

Algorithm 1 (Model decomposition)

```
Given a CWF-net $(\Sigma, P, T, A, N, C, G, E, I)$
For each $p_i$ in P
    If $(k=|site({}^{\bullet}p_i \cup p_i^{\bullet})|>1)$
        Let $\{s_1, s_2, \ldots, s_k\} = site({}^{\bullet}p_i \cup p_i^{\bullet})$
        Let $\{ p_{i1}, p_{i2}, \ldots, p_{ik} \} = dup(p_i, k)$
        For each $t_j$ in ${}^{\bullet}p_i$
            $s_m = site(t_j)$
            $t_j^{\bullet} = \{t_j^{\bullet}\} \backslash \{p_i\} \cup \{p_{im}\}$
        End For
        For each $t_j$ in $p_i^{\bullet}$
            $s_n = site(t_j)$
            ${}^{\bullet}t_j = \{{}^{\bullet}t_j\} \backslash \{p_i\} \cup \{ p_{in} \}$
        End for
    End If
End For
```

Firing Rules in Fragments

From Fig. 3 we know that in the fragments we obtained, places are classified into two categories, one is the kind of places which exist in the original CWF-net, for example, p_0 and p_1 in fragment f_1, we call this kind of places *common places*; the other is the places generated by the *dup* function, we call them *duplicated places*, for example, p_{21} and p_{31} in fragment f_1. For places of different types, we have different firing rules.

1. Local firing rule: for the common places, the firing rule is the same as ordinary Petri nets.
2. Global firing rule: for the duplicated places, we suppose that if a duplicated place p_d possesses one token, then this token is shared among all duplicated places generated by the same place in original WF-net, i.e., *dups*(p_d). And once any duplicated place p_d fires a transition then this token is removed and cannot be used by any other duplicated places in *dups*(p_d).

For example, in Fig. 3 place p_0 and p_1 are common places, so the firing rule is the same as ordinary Petri nets. After the firing of t_1 and t_2, duplicated places p_{21} and p_{31} hold one token respectively. Because p_{21} and p_{31} are duplicated places, according to the global firing rule, the token in p_{21} is shared with p_{23}, so transition t_3 is enabled. With the same reason, p_{31} shares the token it holds with p_{32}, so transition t_4 is enabled. By firing t_3 and t_4, fragment f_2 and f_3 become active and the process is executed continuously.

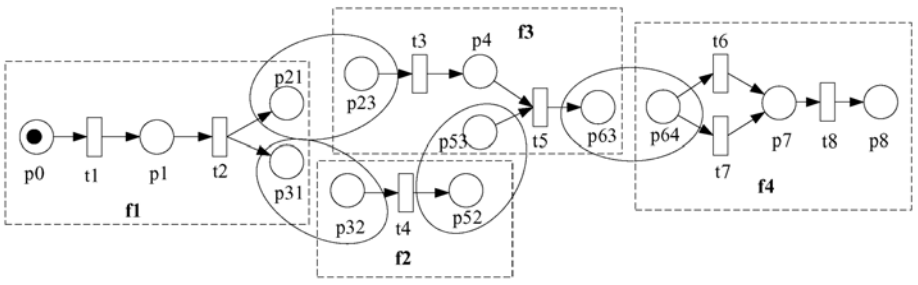

Fig. 3. Results after Fragmentation

5 Correctness of Fragmentation Algorithm

In this section we come to the correctness issue of the fragmentation algorithm. Generally the following three aspects are taken into account, i.e., completeness of the fragmentation, completeness of each fragment and the behavioral equivalence after fragmentation. We are going to discuss the three aspects respectively.

5.1 Completeness of the Fragmentation

The completeness of the fragmentation concerns whether all the fragments can be put together to rebuild the original model.

Through Algorithm 1, a CWF-net is partitioned by duplicating the places which are shared by different fragments. For brevity, we only concern the structure of the fragment, hence we can denote a CWF-net as *PN (P, T, F)*, in which *P* is the set of places, *T* is the set of transitions, and *F* is the set of arcs.

Suppose that by applying Algorithm 1, PN is partitioned into m fragments, i.e., $\{f_1, f_2, \ldots, f_m\}$, and $f_i = (P_i, T_i, F_i)$ for $1 \le i \le m$. We know that if we define $p = \cup dup(p, k)$ for $p \in P$, then we have

$$T_1, T_2, \ldots, T_m \subseteq T, \quad T_1 \cup T_2 \cup \ldots \cup T_m = T;$$
$$P_i = {}^\bullet T_i \cup T_i^\bullet; \qquad F_i = F \cap ((P_i \times T_i) \cup (T_i \times P_i))$$

This kind of partition of a Petri net is called union decomposition in [7], and from the equation above we know that no information about the CWF-net is lost after fragmentation.

5.2 Completeness of Each Fragment

The completeness of each fragment concerns whether each fragment has sufficient information to execute. From Algorithm 1 we know that each fragment is started by one or more places, which denotes the pre-conditions for tasks; and ended by one or more places, which denotes the post-conditions for tasks. In case of conflict, which task to be executed is determined by the local and global firing rule, together with the guard functions on transitions. So we can conclude that each fragment has sufficient information to execute.

5.3 Behavioral Equivalence Between CWF-Net and Fragments

The behavioral equivalence between the CWF-net and the fragments concerns whether the fragments have the same behavioral characteristics with the original CWF-net. Because the fragmentation algorithm imposes union decomposition on the CWF-net, and with the extended firing rules we proposed (i.e., the local firing rule and the global firing rule), the reachability graphs of the fragments and the CWF-net are the same, which means they are behavioral equivalent.

6 Conclusions

In this paper a formal model fragmentation method for distributed workflow execution is discussed. Given the architecture for distributed workflow execution, we partitioned the centralized CWF-net into fragments by duplicating the places shared by tasks executed at different sites, and the behavioral equivalence between the CWF-net and resulted fragments are guaranteed by the extended firing rules.

The correctness of the fragmentation algorithm is ensured because no information is lost after fragmentation, each fragment has sufficient context to execute, and the fragments are behavioral equivalent to the original CWF-net.

Future research issues include the dynamic fragmentation method and the fragmentation policies. This paper only deals with static fragmentation method, i.e., in the situation that the execution site of each task is designated before the process is going to initiate. But when the process is enacting among different sites, some sites may become very busy, or even unavailable, so it's reasonable to believe that designating execution sites for tasks and do fragmentation dynamically will increase the flexibility and performance of the system.

Another issue is the fragmentation policies, that is, the biases we take when we group adjacent tasks into fragments. There are many factors influencing the fragmentation policy. For example, sometimes the tasks belong to the same organizational unit or located at the same place should be put into one fragment for management convenience. And in the data-intensive processes, tasks which exchange large volume of data should be put into one fragment to reduce the data transfer amount among sites. And sometimes which tasks are to be put into one fragment is decided by the process user. All these policies must be studied in detail to improve the availability and performance of the workflow system.

References

1. Georgakopoulos, D., Hornick, M., Sheth A.: An Overview of Workflow Management: from Process Modeling to Workflow Automation Infrastructure. Distributed and Parallel Databases 3 (1995) 119-153
2. Mohan, C., Alonso, G., Guenthoer, R., Kamath, M., Reinwald, B.: An Overview of the Exotica Research Project on Workflow Management Systems. In: Proc. 6th International Workshop on High Performance Transaction Systems, Asilomar, CA (1995)
3. Muth, P., Wodtke, D., Weissenfels, J., et al.: From centralized workflow specification to distributed workflow execution. Journal of Intelligent Information Systems 10 (1998) 159-184

4. T, Cai, P, Gloor, S, Nog.: DartFlow: A Workflow Management System on the Web using Transportable Agents. Technical Report of Dartmouth College, Hanover, USA (1996)
5. WMP Van der Aalst. The application of Petri nets to workflow management. Journal of Circuits Systems and Computers 8 (1998) 21-66
6. Kurt Jensen.: Coloured Petri nets: basic concepts, analysis methods, and practical use. Springer-Verlag, Berlin Heidelberg New York (1992)
7. Wang, P.L,, Zhao Y.J., Ye Z.B.: Union decomposition of Petri net. Control theory and applications 18 (2001) 116-118 (in Chinese with English abstract)