Available online at www.sciencedirect.com
ScienceDirect
Computers in Industry 58 (2007) 381–391
COMPUTERS IN INDUSTRY
www.elsevier.com/locate/compind

# Dynamic workflow model fragmentation for distributed execution

Wei Tan *, Yushun Fan

*Department of Automation, Tsinghua University, 100084 Beijing, PR China*

Received 18 February 2006; accepted 14 July 2006
Available online 30 August 2006

**Abstract**

Workflow fragments are partitions of workflow model, and workflow model fragmentation is to partition a workflow model into fragments, which can be manipulated by multiple workflow servers. In this paper a novel dynamic workflow model fragmentation algorithm is proposed. Based on the well-known Petri net formalism, this algorithm partitioned the centralized process model into fragments step by step while the process is executed. The fragments created can migrate to proper servers, where tasks are performed and new fragments are created and forwarded to other servers to be executed in succession. The advantages of the proposed dynamic model fragmentation method include the enhanced scalability by outsourcing the business functionalities, the increased flexibility by designating execution sites on-the-fly, the avoidance of redundant information transfer by judging their pre-conditions before forwarding fragments, etc. An industrial case is given to validate the proposed approach. Later some discussions are made on the correctness of the algorithm and the structural properties of the workflow model. Finally the future research perspectives are pointed out.


*Keywords:* Distributed workflow; Dynamic workflow model fragmentation; Petri net

## 1. Introduction

Workflow management 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, a workflow system 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 on the client/server architecture, in which a single workflow server takes the responsibility for the operation of the whole process. Meanwhile, this sort of centralized systems may bring about many disadvantages. First of all, with an increasing need of relocating entire business functions to either self-owned or third-party service providers, business process outsourcing (BPO) has been the trend in management as well as IT field. When an company is leveraging technology vendors to provide and manage some of its enterprise applications, its business process may be distributed among geographically dispersed business partners; therefore the involved 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.

The aim of distributed workflow execution is to separate one integrated workflow model into small partitions and allot them to different servers to be executed. To solve the difficulties that centralized workflow system cannot overcome, many distributed workflow systems have been designed from different approaches.

In this paragraph we give a brief introduction to the related work, a detailed comparison of these works with ours is given in Section 6. Replicated servers and server clusters are used to address the required levels of scalability and fault tolerance in commercial workflow systems, which can be seen as a primary and pragmatic solution to distributed workflow execution [10]. The Exotica project [2] proposes a completely distributed architecture in which a set of autonomous nodes cooperate to complete the execution of a process, with persistent message queue as its information transmission technique. METEOR [3]

* Corresponding author. Tel.: +86 10 6277 6211; fax: +86 10 6278 9650.
*E-mail addresses:* tanwei@mails.tsinghua.edu.cn (W. Tan), fanyus@tsinghua.edu.cn (Y. Fan).

0166-3615/$ – see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.compind.2006.07.004

and Mentor [4] project are developed with similar approaches. Aalst introduces his well-known WF-net model to the inter-organizational paradigm [5]. In [6], an agent-based workflow management system is proposed. The work in [7–10] has shown the use of the mobile agents in distributed workflow execution, the mobile agents which carry parts of the process information can migrate from host to host to execute the workflow tasks. In [11,12], innovative approaches to support decentralized process enactment with the Peer to Peer (P2P) technology are presented.

However, the research work in this field mainly focuses on the design of the system architecture and the implementation technique based on specific communication mechanisms. As far as we have known, little attention has been paid to the formal method of workflow model fragmentation.

Workflow model is the basis for workflow execution. In distributed workflow execution paradigm, the whole process is to be executed at multiple sites instead of a single one. Therefore, the workflow model must be partitioned into small parts and transferred to their designated sites. We call these small parts of a workflow model *fragments*, which carry adequate information, so that they can be manipulated by any given workflow engine. *Workflow model fragmentation* is to partition a workflow model into fragments. We emphasize that model fragmentation is the basis for distributed workflow execution.

In this paper we propose a Petri net-based approach for dynamic fragmentation of a workflow model. Our approach is based on the well-known Petri net formalism. We partition the centralized process model into fragments step by step while the process is executed. The fragments created can migrate to proper servers, where tasks are performed and new fragments are created and forwarded to other servers to be executed in succession. The advantages of the proposed dynamic model fragmentation method include the enhanced scalability by outsourcing the business functionalities, the increased flexibility by designating execution sites on-the-fly, the avoidance of redundant information transfer by judging their pre-conditions before forwarding fragments, etc.

This paper is organized as follows. In Section 2, the problem to be solved in this paper is formulated. The workflow model, the centralized and distributed architecture, and some specifications of workflow fragment are introduced here. In Section 3, the dynamic fragmentation algorithm, i.e., the algorithm to create fragments during process execution is presented. In Section 4, a real case is given to illustrate the advantage of the proposed approach. In Section 5, some discussions are made. Section 6 summaries the related work and compared their approaches with ours. Section 7 concludes the paper and gives some research perspectives.

## 2. Problem formulation

### 2.1. Centralized workflow model

A centralized workflow model is a pre-requisite for distributed workflow execution. In this paper we adopt WF-net [13] proposed by Van der Aalst, as the centralized workflow model. WF-net is a special class of Petri net, which prevails in workflow modeling field because of its graphic nature and theoretical foundation. We do not use high-level Petri nets (colored Petri nets [14], for example) because in this paper we mainly focus on the issue of structural partition. At the same time, we acknowledge the need for using colored Petri net when data or resource issue is further considered, and for workflow modeling with colored Petri nets, one can refer to [15].

In this paper, a WF-net is denoted as a tuple $(P, T, A)$, in which $P$ is the set of places, $T$ is the set of transitions, and $A$ is the set of arcs. We assume that the centralized workflow model is a well-structured and acyclic WF-net, because it is reasonable to assume that in the distributed workflow paradigm, the centralized model is well structured and contains no loop. Well-structured property of a WF-net implies the balance of AND/OR-splits and AND/OR-joins, i.e., alternative flows created via an OR-split should also be joined by an OR-join; parallel flows created via an AND-split should also be synchronized by an AND-join. The definition of well-structured WF-net can be found in [16]. Acyclic property of a WF-net means that the workflow model contains no recursive flows. Meanwhile, we also mention how to deal with cyclic models in Section 4. For the properties of WF-net, one can refer to [13,16], and for the basic definitions of Petri net, one can refer to [17].

### 2.2. Centralized and distributed workflow execution

In traditional centralized workflow management system, there is one central workflow server takes charge of the operation of the overall process, so the workflow engine must communicate with each task performer, deliver necessary information and retrieve the outcome of each task (see Fig. 1(a)).

With the need of distributed workflow execution, many approaches have been proposed, ranging from server clusters to radically distributed architectures [10]. In this paper we present a novel view on this problem. We classify the distributed workflow execution paradigms into two categories according to the model fragmentation method used, i.e., the static paradigm and the dynamic one. In the static fragmentation paradigm [18], before the process is initiating, each task in the workflow model is designated to one workflow server (site) at which it is going to be executed. By this means the process model is naturally separated into several fragments. For example, in the workflow process in Fig. 1(b), tasks $t_1$ and $t_2$ are designated to *server* 1, $t_4$ is designated to *server* 2, $t_3$ and $t_5$ are designated to *server* 3, and the rest are designated to *server* 4. Thus, the workflow model is naturally divided into four fragments, i.e., $f_1$, $f_2$, $f_3$ and $f_4$.

In the static paradigm, the execution site of each task must be determined before the initiating of a process. Obviously it lacks flexibility.

Another paradigm is the dynamic one. It is stimulated by the idea that a workflow process instance can migrate to one server, executing the immediate tasks, partitioning the remaining part, and forwarding the remainder to the next servers. Generally speaking, the model is fragmented step by step with the execution of the process.

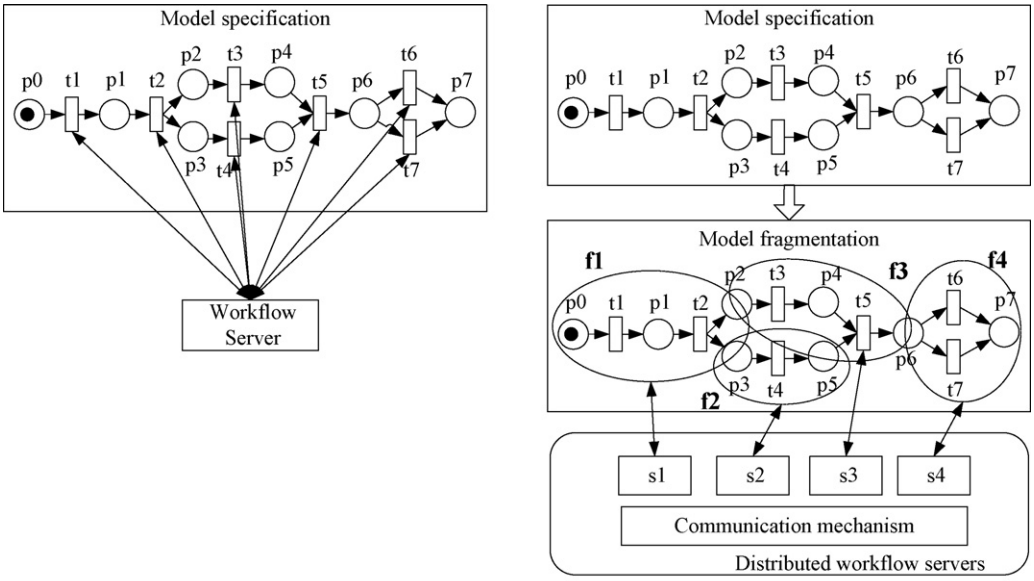



Fig. 1. Architecture for (a) centralized and (b) distributed workflow execution.

Fig. 2 illustrates how a workflow model is dynamically fragmented. Fig. 2(a) shows the original workflow model, denoted as fragment $F_1$. When the first task of $F_1$ (i.e., $t_1$) is completed at some site, the remaining part of $F_1$ forms a new fragment $F_2$ (see Fig. 2(b)). The first task of $F_2$ (i.e., $t_2$) is an AND-split transition, so when $t_2$ is completed, two parallel fragments are generated (i.e., $F_3$ and $F_4$ in Fig. 2(c)). When $t_4$ is completed, the token held by $p_5$ in $F_4$ is transferred to $p_5$ in $F_5$, and then $F_4$ can be neglected. Now let us turn to fragment $F_3$. After $t_3$ is completed, the remaining part forms fragment $F_5$ (see Fig. 2(d)). When $t_5$ in $F_5$ is completed, the remaining part again forms two fragments (i.e., $F_6$ and $F_7$ in Fig. 2(e)). This time the two fragments will not be executed in parallel since $p_6$ is an OR-split place so only one subsequent task can be executed.

During the execution, the fragments can be carried by mobile agents, moving from one site to another, so $F_i$ ($1 \leq i \leq 7$) can be executed at different sites.

This sort of fragmentation paradigm has many advantages. First of all, when the process is enacting among different sites, the pre-designated sites may become busy or even unavailable, so designating executing sites at runtime and do fragmentation dynamically will increase the flexibility and performance of the system. Secondly, the fragments can be forwarded to the execution site by mobile agents, so concurrent tasks can be forwarded to different sites to achieve real parallelism. In addition, in the choice-block, which flow is to be executed can be judged on-the-fly, thus, only the executable flow is forwarded (see $F_6$ and $F_7$ in Fig. 2(e)). Finally, in the data-intensive processes, instead of transferring a large volume of

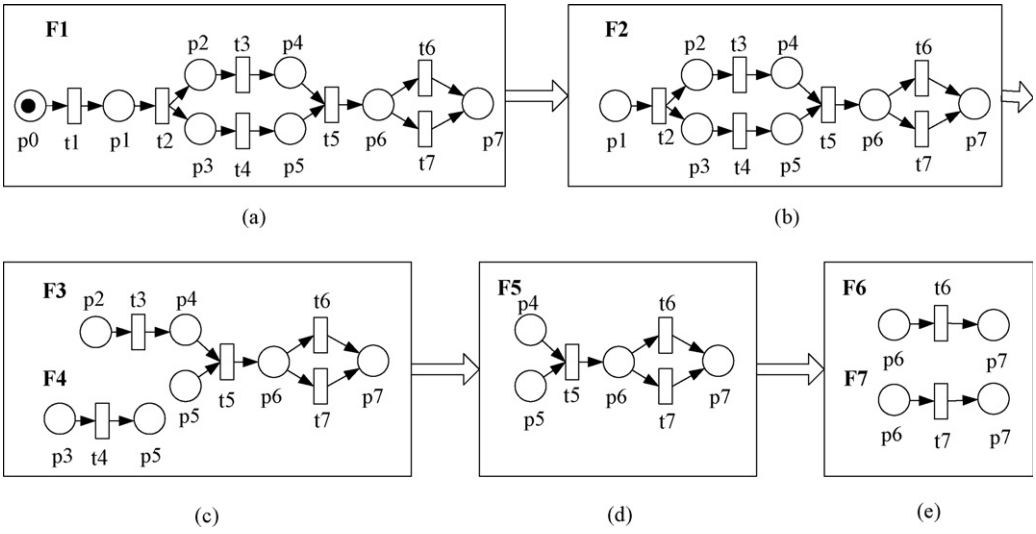



Fig. 2. Process of dynamic fragmentation.

data to the workflow server's site, an agent with process information can travel to the data site, eliminating intermediate traffic and ensure data integrity.

Another issue is that when an AND-split/AND-join block is encountered, two or more parallel fragments will be generated. Since these fragments will all be executed eventually, the information of the tasks after the AND-join transition is not necessarily carried by all parallel fragments. In our dynamic fragmentation algorithm shown in Section 3, we pay special attention to this issue.

### 2.3. Specification of fragment

In this section some definitions which are to be used in the following part of this paper are given.

Informally, a *fragment* is a partition of a workflow model, and it consists of a *source transition*, all the transitions reachable from the source transition, and all the linking places of these transitions. To express all the reachable nodes from one node in a Petri net, we give a formal definition of *reachability*.

**Definition 1.** (*Reachability*) In a Petri net $(P, T, A)$, for $n_1$, $n_j \in P \cup T$, $R(n_1, n_j)$ is true iff there is a path C from $n_1$ to $n_j$ $\langle n_1, n_2, \ldots, n_j\rangle$ such that $(n_i, n_{i+1}) \in A$ for $1 \le i \le j-1$, and for any two nodes $n_p$ and $n_q$ in C, $p \ne q \Rightarrow n_p \ne n_q$. We define $R(n_i, n_i)$ = true.

Based on Definition 1, we can give a formal definition of fragment.

**Definition 2.** (*Fragment*) Given a WF-net $W = (P, T, A)$, a fragment $F$ is also a Petri net $(P_f, T_f, A_f)$ such that

(i) $T_f \subseteq T$; $P_f = {}^\bullet T_f \cup T_f^\bullet$; $A_f = A \cap ((P_f \times T_f) \cup (T_f \times P_f))$;
(ii) $F$ has a special transition $t_s$ such that ${}^\bullet({}^\bullet t_s) = \varnothing$;
(iii) $\forall t \in T_f$, $R(t_s, t)$ = true.

A fragment $F$ is also denoted as $(t_s, F)$ to emphasize its source transition. Fig. 2 gives many examples of fragments, for example, $t_3$ is the source transition of $F_3$.

In Definition 2, we assume that in each fragment there is only one source transition. We make this assumption to ensure each fragment has only one immediate task to fulfill, which enhances the execution parallelism. It is clear that if a WF-net is not started by an OR-split, this WF-net is also a fragment. By adding a null task $t_{null}$ before the starting OR-split place, a WF-net with starting OR-split place can be transformed to a fragment, as is shown in Fig. 3.

In fragment $F = (P_f, T_f, A_f)$, we define all the transitions reachable from a given transition $t_{sf}$ and the linking places of these transitions as the Reachable sub-fragment of $t_{sf}$. A formal definition is given below.

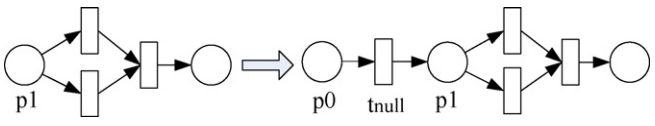



Fig. 3. Model transformation to a fragment.

**Definition 3.** (*Reachable sub-fragment*, *RSF*) For $F = (P_f, T_f, A_f)$ and $t_{sf} \in T_f$, $\mathrm{RSF}(t_{sf}, F) = (P_{rs}, T_{rs}, A_{rs})$ such that

(i) $T_{rs} \subseteq T_f$; $P_{rs} = P_f \cap ({}^\bullet T_{rs} \cup T_{rs}^\bullet)$; $A_{rs} = A_f \cap ((P_{rs} \times T_{rs}) \cup (T_{rs} \times P_{rs}))$;
(ii) $\forall t \in T_f$, if $R(t_{sf}, t)$ = true in $F$, then $t \in T_{rs}$.

From Definition 3, we know that $\mathrm{RSF}(t_{sf}, F)$ is also a Fragment, with $t_{sf}$ as its source transition. The concept of Reachable sub-fragment will be used in Section 3, when we build new fragments upon completing one task. Algorithm 1 gives the method to obtain a Reachable sub-fragment of transition $t_{sf}$ in fragment $F$.

**Algorithm 1.** $\mathrm{RSF}(t_{sf}, F)$: Returns reachable sub-fragment start with transition $t_{sf}$ in fragment $F$. $\mathrm{RSF}(t_{sf}, F) = (t_{sf}, (P_{rs}, T_{rs}, A_{rs}))$; $F = (t_s, (P_f, T_f, A_f))$;

Step 1: Pretreatment
  $T_{rs} = \{t_{sf}\}$; $P_{rs} = \{{}^\bullet t_{sf}\}$; $A_{rs} = P_{rs} \times T_{rs}$
Step 2: Calculate $\mathrm{RSF}(t_{sf}, F)$
  Let $\{p_1, p_2, \ldots, p_k\} = t_{sf}^\bullet$ $(k \ge 1)$
  $P_{rs} = P_{rs} \cup t_{sf}^\bullet$
  $A_{rs} = A_{rs} \cup \{(t_{sf}, p_1), (t_{sf}, p_2), \ldots, (t_{sf}, p_k)\}$
  If $(t_{sf}^\bullet)^\bullet = \varnothing$
    Return $(t_{sf}, (P_{rs}, T_{rs}, A_{rs}))$
  Else
    For each $p_i$ in $\{p_1, p_2, \ldots, p_k\}$
      If $p_i^\bullet \ne \varnothing$
        $\{t_{i1}, t_{i2}, \ldots, t_{ij}\} = p_i^\bullet$ $(j \ge 1)$
        $\mathrm{RSF}(t_{sf}, F) = (P_{rs}, T_{rs}, A_{rs}) \cup \mathrm{RSF}(t_{i1}, F) \cup \mathrm{RSF}(t_{i2}, F) \cdots \cup \mathrm{RSF}(t_{ij}, F)$
      End If
    End For
  End If
Step 3: Return $\mathrm{RSF}(t_{sf}, F)$

## 3. Dynamic workflow model fragmentation method

In [18], we deal with static fragmentation method for distributed workflow execution. In this section we come to the dynamic model fragmentation method.

### 3.1. Issue of information redundancy

As we have mentioned in Section 2.2, when an AND-split/AND-join block is encountered, two or more parallel fragments will be generated. Since all these fragments will be executed, the information of the tasks following the AND-join transition is not necessarily carried by all parallel fragments. For example, in Fig. 2(b) and (c), fragment $F_2$ is started by an AND-split $t_2$, and the corresponding AND-join task is $t_5$. When task $t_2$ is completed, two fragments will be generated. If we generate two fragments by function RSF, i.e., $F_3 = \mathrm{RSF}(t_3, F_2)$ and $F_4 = \mathrm{RSF}(t_4, F_2)$ (see Fig. 4), we find that the process information behind AND-join transition $t_5$ is carried by both $F_3$ and $F_4$. Consider that $F_3$ and $F_4$ are to be executed in parallel and to be merged in $t_5$, only one copy of the process information after $t_5$ needs to be kept. Therefore, in Fig. 2(c), the transitions and places behind AND-join transition $t_5$ is truncated in one of the following fragments $F_4$.

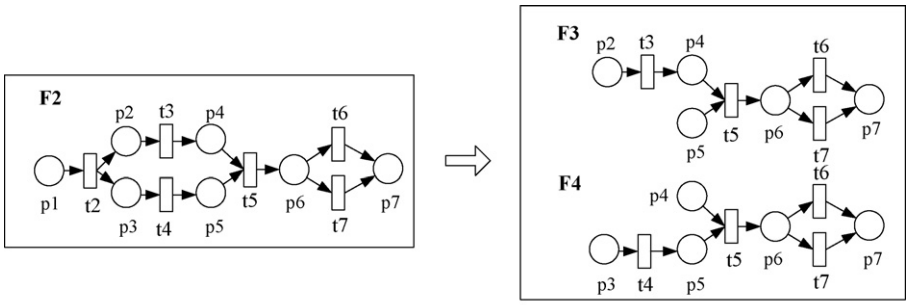



Fig. 4. An example to illustrate information redundancy.

To solve this problem, we introduce the concept of transition restricted reachable sub-fragment (TRRSF), to truncate fragments with AND-join transitions. A formal definition is given below.

**Definition 4.** (*Transition restricted reachable sub-fragment, TRRSF*) For $F = (P_f, T_f, A_f)$, $t \in T_f$, $T_r = \{t_1, t_2, \ldots, t_k\} \subset T_f$ $(k \geq 0)$ and $t \notin T_r$, $\mathrm{TRRSF}(t, F, T_r) = (P_{rrs}, T_{rrs}, A_{rrs})$ such that

(i) $T_{rrs} \subseteq T_f$; $P_{rrs} = P_f \cap ({}^{\bullet}T_{rrs} \cup T_{rrs}{}^{\bullet})$; $A_{rrs} = A_f \cap ((P_{rrs} \times T_{rrs}) \cup (T_{rrs} \times P_{rrs}))$;
(ii) $\forall t_s \in T_f$ if $R(t, t_s) = \text{true}$ and $R(t_1, t_s) = R(t_2, t_s) = \cdots = R(t_k, t_s) = \text{false}$ in $F$, then $t_s \in T_{rrs}$.

For example, in Fig. 4, when we truncate $\mathrm{RSF}(t_4, F_2)$ with $t_5$, we get $F_4 = \mathrm{TRRSF}(t_4, F_2, \{t_5\})$ (see Fig. 2(c)).

Algorithm 2 gives the formal method to obtain a Transition restricted reachable sub-fragment of transition $t$ in fragment $F$, restricted by transitions in set $T_r$.

**Algorithm 2.** $(P_{rrs}, T_{rrs}, A_{rrs}) = \mathrm{TRRSF}(t, F, T_r)$: Returns the transition restricted reachable sub-fragment of $F$, started from transition $t$, restricted by transitions in $T_r$.

Step 1: Pretreatment
  $T_{rrs} = \{t\}$; $P_{rrs} = \{{}^{\bullet}t\}$; $A_{rrs} = P_{rrs} \times T_{rrs}$
Step 2: Calculating $\mathrm{TRRSF}(t, F, T_r)$
  Let $\{p_1, p_2, \ldots, p_k\} = t^{\bullet}$ $(k \geq 1)$
  $P_{rrs} = P_{rrs} \cup t^{\bullet}$
  $A_{rrs} = A_{rrs} \cup \{(t, p_1), (t, p_2), \ldots, (t, p_k)\}$
  If $({}^{\bullet}t^{\bullet}) \subseteq T_r$
    Return $(P_{rrs}, T_{rrs}, A_{rrs})$
  Else
  For each $p_i$ in $\{p_1, p_2, \ldots, p_k\}$
    If $\{t_{i1}, t_{i2}, \ldots, t_{ij}\} = p_i^{\bullet} - (p_i^{\bullet} \cap T_r) \neq \varnothing$
      $\mathrm{TRRSF}(t, F, T_r) = (P_{rrs}, T_{rrs}, A_{rrs}) \cup \mathrm{TRRSF}(t_{i1}, F, T_r) \cup \mathrm{TRRSF}(t_{i2}, F, T_r) \cdots \cup \mathrm{TRRSF}(t_{ij}, F, T_r)$
    End If
  End For
  End If
Step 3: Return $\mathrm{TRRSF}(t, F, T_r)$

### 3.2. Algorithms for dynamic model fragmentation

Here we give a brief explanation of the control flow of Algorithm 3 which partitions the workflow model into fragments dynamically. First let us discuss the situation that the source transition $t_s$ of $F$ has only one output place $p$. If $p$ has only one output transition, we can just cut off $t_s$ from $F$ and receive a subsequent fragment (see Fig. 2(a)). Else if $p$ has multiple output transitions, then each transition is the source transition of a new fragment (see Fig. 2(d) and (e)), multiple fragments are obtained although only one of them can really be enabled and executed since they are mutual exclusive.

Another situation, which is more difficult to tackle, is the case that when $t_s$ has multiple output places. In this situation each output place forms at least one new fragment, and these fragments are to be executed in parallel. So we take some measure to avoid information redundancy when multiple fragments are generated, i.e. we introduce the idea of Transition restricted reachable sub-fragment, and when we do fragmentation, the set $T_r$ is updated constantly to prohibit the unnecessary spanning of sub-fragments (see Fig. 2(b) and (c)).

Algorithms 4–6 are used by Algorithm 3. Algorithm 4 is to find the join transition of an AND-split transition, and Algorithm 5 is to find the split transition of an AND-join transition. Algorithm 6 is used to update $T_r$. By using Algorithm 6, if a fragment $F$ is started with an AND-join/AND-split block of which the split transition is $t_s$ and the join transition is $t_j$, then in a newly generated fragment $F_1$, if transition $t$ is between $t_s$ and $t_j$, and partially joins some of the transitions split at $t_s$, then $t$ is added to the set of restricted transitions $T_r$.

**Algorithm 3.** Dynamic model fragmentation

Input: Fragment $(t_s, F)$
Output: A list of fragments, denoted as F_LIST
If $(|t_s^{\bullet}| = 1)$
  Let $p = t_s^{\bullet}$
  If $(|p^{\bullet}| = 1)$// $p$ has only one output transition
    Let $t_{next} = p^{\bullet}$
    Add $(t_{next}, \mathrm{RSF}(t_{next}, F))$ to F_LIST
  Else//$p$ has multiple output transitions
    Let $\{t_1, t_2, \ldots, t_k\} = p$
    For each $t_i$ in $p$
      Add $(t_i, \mathrm{RSF}(t_i, F))$ to F_LIST
    End For
  End If
End If
Else//$t_s$ has multiple output places, i.e., $t_s$ is a AND-split
  Let $\{p_1, p_2, \ldots, p_k\} = t_s^{\bullet}$
  Let $t_{join} = \mathrm{JoinTrans}(t_s)$
  $T_r = \{t_{join}\}$
  For each $p_i$ in $t_s$
    If $(|p_i^{\bullet}| = 1)$

```
            Let t_{i next} = p_i^•
            If (i = 1)
                Add F_{i next} = (t_{i next}, RSF(t_{i next}, F)) to F_LIST
                Update (T_r, F_{i next}, F)
            Else
                Add F_{i next} = (t_{i next}, TRRSF (p_i^•, F, T_r)) to F_LIST
                Update (T_r, F_{i next}, F)
            End If
    Else //p has multiple output Transitions
        Let {t_{i1}, t_{i2}, ..., t_{iki}} = p_i^•
        For each t_{ij} in p_i^•
            If (i = 1)
                Add F_{ij next} = (t_{ij}, RSF (t_{ij}, F)) to F_LIST
            Else
            Add F_{ij next} = (t_{ij}, TRRSF (t_{ij}, F, T_r)) to F_LIST
            End If
        End For
        Update (T_r, F_{i1 next}, F)
    End If
  End For
End If
```

**Algorithm 4.** JoinTrans ($t_s$)—Returns the corresponding AND-join transition of an AND-split transition

```
Let {p_1, p_2, ..., p_k} = t_s^•(k ≥ 2)
Let Q be an empty queue
Select one transition from p_1^•, denote as t_1
Add t_1 to Q
While Q! = NULL
    Pop one transition from the head of Q, denote as t_h
    If R(p_2, t_h) = true AND R(p_3, t_h) = true AND ··· AND R(p_k, t_h) = true
      Return t_h
    Else
      Push (t_h^•)^• to the tail of Q
    End If
End While
```

**Algorithm 5.** SplitTrans ($t_j$)—returns the corresponding AND-split transition of an AND-join transition

```
Let {p_1, p_2, ..., p_k} = •t_j (k ≥ 2)
Let Q be an empty queue
Select one transition from •p_1, denote as t_1
Add t_1 to Q
While Q! = NULL
    Pop one transition from the head of Q, denote as t_h
    If R(t_h, p_2) = true AND R(t_h, p_3) = true AND ··· AND R(t_h, p_k) = true
        Return t_h
    Else
        Push •(•t_h) to the tail of Q
    End If
  End While
```

**Algorithm 6.** Update ($T_r$, $F_1$, $F$)

Fragment $F$ is started by an AND-split/AND-join block, in which the split and join transition is denoted as $t_s$ and $t_j$, respectively. $F_1$ is a sub-fragment of $F$.

```
Let (P_1, T_1, A_1) = F_1
For each t ∈ T_1
    If R(t, t_j) = true in F AND t ≠ t_s, t ≠ t_j AND |•t| > 1 AND SplitTrans
(t) = t_s in F
        T_r = T_r ∪ {t}
    End If
End For
```

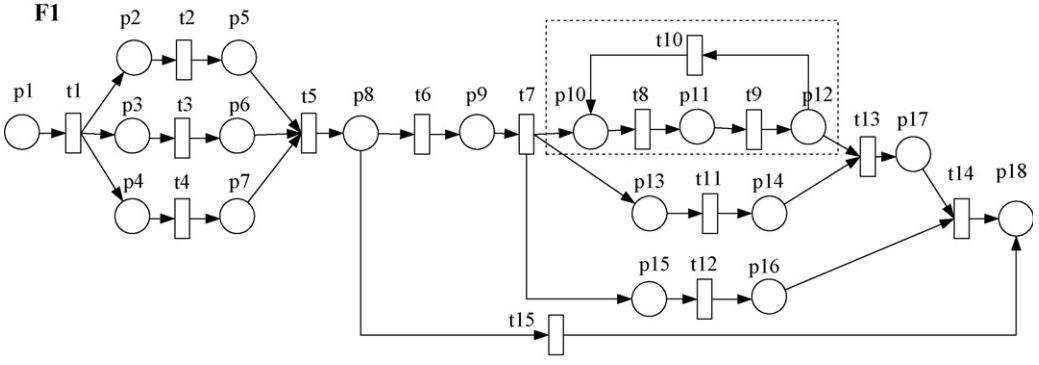
## 4. Case study

In this section we give a real case on how we use the proposed approach to implement a distributed workflow management system in a bike manufacturing company. For the integrity of the content, we concentrate on model fragmentation method, omitting some of the implementation details.

### 4.1. Background

XBike is a company offering bike customization services. As a small and medium sized enterprise (SME), XBike outsources some of its business functions (e.g., production and logistics) because of a will to concentrate on its core competence, i.e., the ability to design and deliver various customized bikes.

Although later we still use the term *department* to indicate the performers of the business, we emphasize that actually the stock, the production and the logistics department are independent partners collaborate with XBike based on contracts.




t1: Check and complete order
t2: Calculate price
t3: Check stock
t4: Check technical feasibility
t5: Check order feasibility
t6: Send order confirmation letter
t7: Generate Worksheet
t8: Produce bike
t9: Check Quality
t10: Generate quality report
t11: Produce package
t12: Arrange transportation
t13: Package
t14: Delivery & Payment
t15: Send order modification letter

Fig. 5. The bike customization process.

Fig. 5 illustrates the bike customization process of XBike. When a customer wants to place an order, he visits XBike's website and fills in the related information, including the customer information, bike information (type, brakes, pedals, tires, etc.) and some extra-specifications. Then a bike customization process starts. First, the sales department checks the order and completes possible missing fields. Afterwards three tasks are performed in parallel: the financial department calculates the price, the stock department checks the stock, and the design department checks the technical feasibility. After all these three steps completes, the system decides whether this order is feasible or not. If it is feasible, an order confirmation letter is sent to the customer, a worksheet containing product, package and delivery information is generated, and the bike will be produced by the production department. If the bike passes quality check, it is packaged and delivered to the customer by the logistics department; if it fails to pass quality check, a quality check report is generated and redirected to the task *produce bike* (this task may be complex and nested, however, we do not cover the details here). While if the order is not feasible due to some reason, an order modification letter with suggestions will be sent to the customer and the process terminates.

### 4.2. System implementation

In this project, we have made some modifications to our formerly developed central workflow management system—the CIMFlow system [19], as the cross-enterprise workflow management system.

Our architecture is built based on the P2P scheme, which means that the components in each site are identical. The main components of each workflow suite are shown as follows.

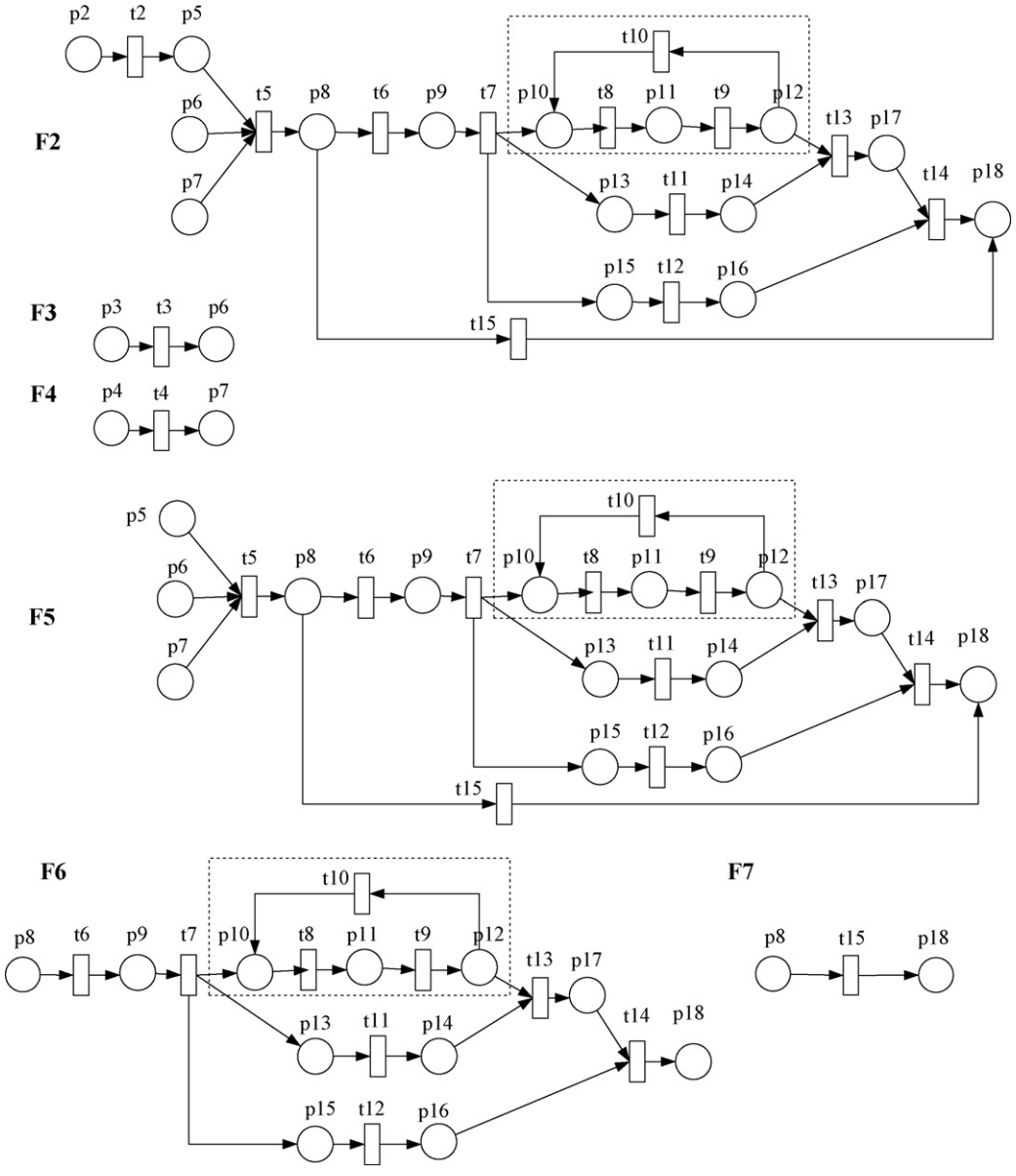



Fig. 6. Model fragmentation of the bike customization process.

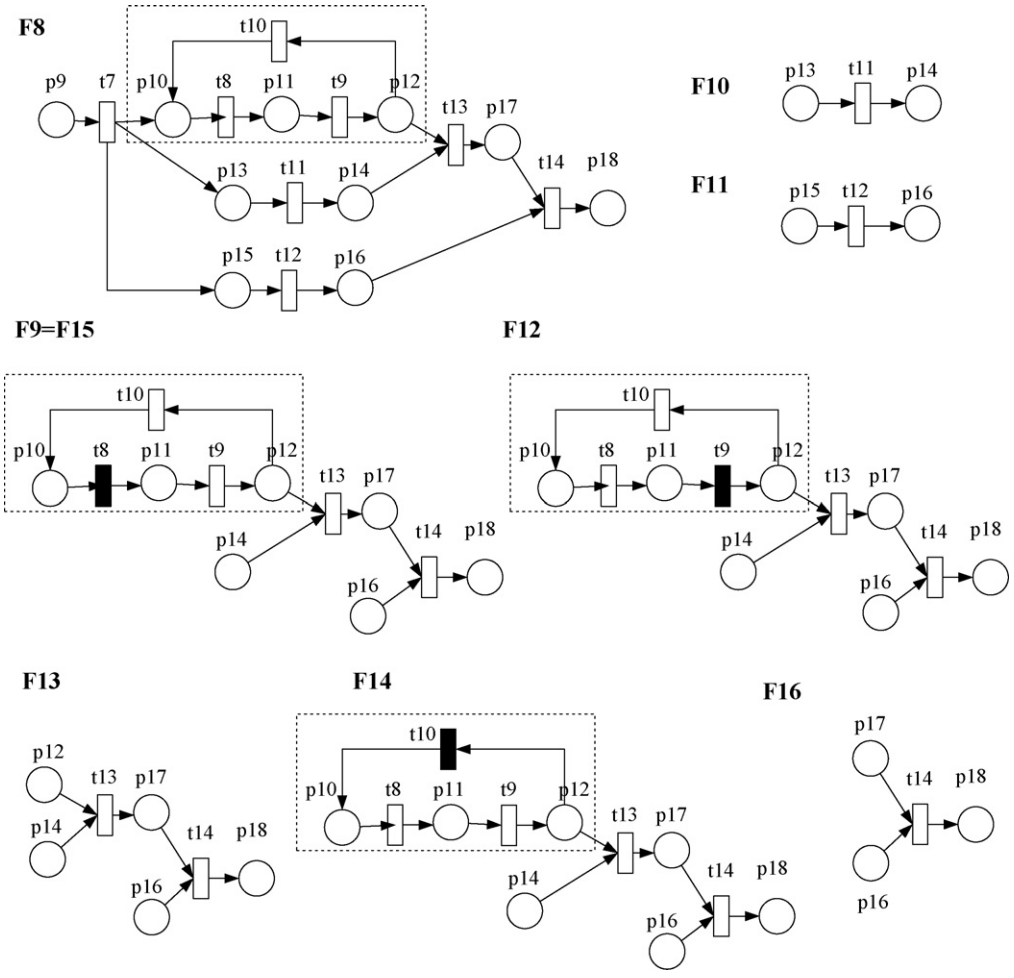



Fig. 6. (*Continued*).

#### 4.2.1. Process modeler

*Process modeler* allows the users at current site to establish workflow model. For example, the Sales Department of XBike can set up an order-processing model.

#### 4.2.2. Task allocator

*Task allocator* receives the task allocation request and allocates tasks to specific sites at which they will be performed. For example, the task *Tech. Check* is suggested to be allocated to one of the several sites at Design department, while the task *Send confirmation letter* is mandatory to be allocated to the site at which the email server locates.

#### 4.2.3. Task monitor

*Task monitor* keeps a log on the allocated sites for each task, their performance and other issues (for example, the charge for consuming the service). Task monitor is of great importance in the collaborative environment in virtual enterprises since it provides feedback mechanism for business process monitoring and improvement.

#### 4.2.4. Task client

*Task client* maintains a task list for each site, and workflow users at this site can access this task list and manipulate tasks in it.

#### 4.2.5. Fragment pool

Fragment pool keeps all the information about the fragments allocated to this site.

#### 4.2.6. Fragment manager

*Fragment manager* serves as the *engine* to drive the workflow process. When a new fragment is put into the *fragment pool*, the *fragment manager* checks whether the first task of this fragment can be executed at once. If the condition is valid (for example, in the AND-join case, all the preceding tasks have finished), the *fragment manager* put this task to the task list to be executed. When this task completes, new succeeding fragments will be generated and sent to the fragment pools at their designated sites. By this means the execution is propagated until all the tasks have been accomplished.

### 4.3. Dynamic model fragmentation

Now let us consider how the process in Fig. 5 is fragmented and performed among the multiple workflow servers (sites) located at different departments (or partners), and we will also show how the proposed fragmentation method have increased the flexibility and productivity of the order processing business. The fragments are presented in Fig. 6.

When an order from customer comes, a workflow fragment $F_1$ is created (see Fig. 5). $F_1$ is sent to one of the workflow servers in the sales department to be executed. When task $t_1$ is completed, three fragments are generated. First, fragment $F_2$ is generated by RSF($t_2$, $F_1$). In $F_2$, task $t_5$ is the AND-join transition of $t_1$, then $t_5$ is added to $T_r$, i.e., $T_r = \{t_5\}$. So other two fragments (i.e., $F_3$ and $F_4$) are generated by function TRRSF($t_3$, $F_1$, $T_r$) and TRRSF($t_4$, $F_1$, $T_r$), respectively. $F_2$, $F_3$ and $F_4$ are sent to the financial, stock and design department respectively, where they will be handled by three workflow servers and thus, real parallelism is achieved. Note that $F_3$ is handled by a different company to which the production business it outsourced.

We know that $t_2$, $t_3$ and $t_4$ are all data-intensive tasks, i.e., when performing these tasks, a large volume of data (the financial, stock and technical data) is needed. In traditional mode, a central workflow server must retrieve these data from a remote site and manipulate them locally. While in the dynamic fragmentation mode, we just forward the fragment to the data site to be executed, by this means unnecessary data transfer is avoided and data security is guaranteed.

When $t_2$, $t_3$ and $t_4$ have all been completed, fragment $F_5$ is generated. When task $t_5$ in $F_5$ is completed, $F_6$ and $F_7$ are generated. Since $F_6$ and $F_7$ are mutual exclusive, only one of them will eventually be executed according to whether the order is feasible or not. So when $t_5$ is completed, only feasible fragment will be forwarded.

If the order is feasible, $F_6$ is generated. When task $t_6$ is completed (order confirmation letter is sent to the customer), fragment $F_8$ is generated. When $t_7$ (generate worksheet) is completed, three tasks are going to be executed in parallel, i.e., $t_8$ (produce bike), $t_{11}$ (produce package) and $t_{12}$ (arrange transportation). Three fragments (i.e., $F_9$, $F_{10}$ and $F_{11}$) will be created. $F_9$ and $F_{10}$ are sent to the production department, where the bike is produced and packaged. At the same time, $F_{11}$ is sent to the logistics department, where transportation is arranged.

Fragment $F_9$ contains a loop which is annotated by a dashed rectangle. (Strictly speaking, $F_9$ is not a fragment since no tasks can be seen as the source transition. In this case we can extend Definition 2 and define $t_8$ as the source transition. Similarly, $F_{12}$, $F_{14}$ and $F_{15}$ can all be seen as fragments.) In this case, when the process is executed inside this loop, no tasks should be deleted since they may be executed again ($t_8$ will be executed again if the result of $t_9$ is QUALITY CHECK NOT PASSED). Therefore, we should not directly use our algorithm here. When $t_8$ in $F_9$ completes, we simply keep the original fragment and denote the next task $t_9$ as the source task of this fragment (the task filled with black in $F_{12}$). When $t_9$ completes, if quality check passes, fragment $F_{13}$ is created, else we again keep the original fragment and denote the next task $t_{10}$ as the source task of this fragment (the task filled with black in $F_{14}$). When $t_{10}$ in $F_{14}$ completes, we again keep the original fragment and denote the next task $t_8$ as the source task of this fragment (the task filled with black in $F_{15}$). When $t_{13}$ in $F_{13}$ completes, fragment $F_{16}$ which contains the last task is created, then the bike is delivered to the customer by the logistics department.

This case manifests all the advantages of the approach we proposed. Moreover, we stress that our algorithm can support arbitrary complex workflow models, which can be nested. And as is shown in this case, we can deal with workflow model with loop structure by simply adding notations in fragments.

Through this real case, we see that the distributed workflow management system and the dynamic model fragmentation method have helped XBike coordinate its business processes which span over several partners. Feedback from XBike has shown that this approach has helped to increase the flexibility of process execution, reduce data transfer and enhance data integrity.

## 5. Discussions

### 5.1. Proof on correctness

The correctness issue of the fragmentation algorithm covers the following three aspects: completeness of the fragmentation, completeness of each fragment and the behavioral equivalence after fragmentation. We are going to discuss these three aspects respectively.

The completeness of the fragmentation concerns whether all the fragments can be put together to rebuild the original model. Given $F = (t_s, (P, T, A))$, let us suppose that by applying Algorithm 3, $F$ is partitioned into $m$ fragments, i.e., $\{F_1, F_2, \ldots, F_m\}$, and $f_i = (P_i, T_i, A_i)$ for $1 \leq i \leq m$. Then we get

$$T_1, T_2, \ldots, T_m \subseteq T, T_1 \cup T_2 \cup \cdots \cup T_m \cup \{t_s\} = T;$$
$$P_i = {}^\bullet T_i \cup T_i^\bullet; \quad A_i = A \cap ((P_i \times T_i) \cup (T_i \times P_i))$$

We know that no information about the original workflow net is lost after fragmentation.

The completeness of each fragment concerns whether each fragment has sufficient information to execute. From Algorithm 3 we know that each fragment is started by places which denote the pre-conditions for tasks, and ended by places which denote the post-conditions for tasks. So each fragment has sufficient information to execute. (Note that in this paper we only concern the structure perspective of the process model, in real business processes data and resource perspective should also be addressed.)

The behavioral equivalence concerns whether the fragments generated by Algorithm 3 have the same behavioral characteristics with the original fragment.

Consider a fragment $F = (P, T, A)$ with source transition $t_s$. From the definition of fragment and source transition we know that if every source place of $F$ holds one token respectively, only $t_s$ is enabled. After $t_s$ is completed, $F$ can be further fragmented into one or more fragments, denoted as $\{F_1, F_2, \ldots, F_m\}$. For $\forall t \in T/\{t_s\}$, $t$ must exist in at least one fragment in $\{F_1, F_2, \ldots, F_m\}$. Suppose that $t$ exists in $\{F_{i1}, F_{i2}, \ldots, F_{ik}\}$ for $\forall F_{ij}$ in $\{F_{i1}, F_{i2}, \ldots, F_{ik}\}$,

$$t^\bullet \ (\text{in every } F_{ij} \text{ in} \{F_{i1}, F_{i2}, \ldots, F_{ik}\}) = t^\bullet \ (\text{in } F)$$

$${}^\bullet t \ (\text{in every } F_{ij} \text{ in } \{F_{i1}, F_{i2}, \ldots, F_{ik}\}) = {}^\bullet t \ (\text{in } F)$$

That is to say, for any subsequent task of $t_s$ in $F$, the fragmentation keeps their pre-condition and post-condition, which means that the fragmentation method preserves their behavioral characteristic.

### 5.2. Discussions on workflow model

As we have stressed earlier, our approach only consider well-structured and acyclic workflow nets. These assumptions have been required by the necessity of starting from a simplified model, yet covering important and typical features required, to undertake an interesting and relevant topic that has not been given much attention in the literature so far.

Our approach resembles the approach proposed in [9] in many aspects. In [9], the author defined a decentralized workflow model, known as self-describing workflows, which allows each task execution agents to receive a self-describing workflow, executes its task, and prepares and forwards self-describing workflows to the next agents. The workflow model is based on direct graph, and there are no restrictions on the model structure (i.e., it may be acyclic and non well-structured), and the fragmentation is done by a simple combination of the reachable successive tasks.

With the assumption we imposed on the workflow model, our work differentiates theirs in the following aspects. First, our model is based on Petri net, so it is relatively easy to analyze the structural and behavioral properties of the workflow model in the future.

Secondly, in [9] usually there will be redundant information between fragments. For example, in Fig. 4, when $t_2$ in $F_2$ completes, by the approach proposed in [9], two fragments ($F_3$ and $F_4$ in Fig. 4) are generated, the process information after $t_5$ is redundant since it only needs to be kept in one fragment ($F_3$ or $F_4$). However, with our approach, redundant information is avoided so the fragments are more compact (see $F_3$ and $F_4$ in Fig. 2(c)). What is more, when redundant information exists, sophisticated measure must be taken to prevent redundant execution. While in our approach, we assume the workflow model to be well-structured and acyclic, and we introduce the concept of Transition restricted reachable sub-fragment by which we guarantee that no redundant fragment will be generated, therefore there will be no redundant execution.

## 6. Related work

Several approaches and architectures have been proposed to support distributed workflow execution. In Section 5.2, we have compared our work with [9], and in this section we will introduce other relevant researches. Here we focus on the difference between our perspective and theirs, and the advantages of our approaches.

### 6.1. Exotica

The Exotica system [2] proposes a completely distributed architecture, in which the information among servers is transferred by the persistent message queue. By this means, the reliability of the system is highly enhanced. METEOR [3] project proposes a similar approach. However, these approaches mainly concentrate on system architecture and implementation technique based on specific communication mechanisms, little attention has been paid to the model fragmentation issue.

### 6.2. Inter-organizational workflow

Aalst extends his WF-net model to the inter-organizational paradigm [5], in which the global workflow model is made up of several local ones. By transferring an inter-organizational workflow into a WF-net, the correctness issue can be solved easily. Our work is also based on WF-net, yet we focus on the dynamic model fragmentation method, which is not covered in [5].

### 6.3. Mentor

The Mentor Project [4] of the University of Saarland developed a traceable and scalable workflow architecture. A formal model known as state chart is utilized for workflow specification, and a model partitioning method is proposed by mapping a centralized state chart to distributed ones. But the workflow model in Mentor is statically partitioned, so it lacks the advantages we have with our approach.

### 6.4. Dartflow

The Dartflow [7] project has shown the use of the mobile agents in distributed workflow execution. In Dartflow, the workflow model is fragmented dynamically, and the partitions are carried by mobile agents and sent to different sites which are responsible for them. But their work focuses on the system architecture, and the model fragmentation method is not well established in [7].

## 7. Conclusion

In this paper a formal model fragmentation method for distributed workflow execution is proposed. We present a novel method to stepwisely partition the centralized workflow model into fragments, and these fragments can migrate to servers to be executed, further fragmented and forwarded. The advantages of the proposed dynamic fragmentation method include the increased flexibility by designating execution sites on the fly, the avoidance of redundant information transfer, etc. Moreover, we have validated the approach we propose in a cross-organizational workflow management system of a bike manufacturing company. This case has shown that our approach can handle distributed workflow execution in dynamic environment, with considerable flexibility and performance.

We make the restriction on the workflow model that it must be well-structured and acyclic, future research might develop more elaborated algorithms which are able to deal with more expressive modeling features. However, we still stress that this restriction on the model can bring about many conveniences in

workflow execution (for example, no redundant part will exists in fragments).

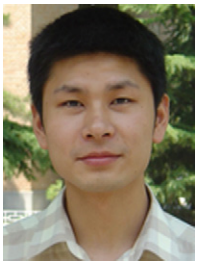
Under some circumstances, doing fragmentation whenever one task is completed can be over elaborated. Suppose that $N$ tasks are going to be executed sequentially at the same site, we have no reason to do fragmentation for $N-1$ times. One way to deal with this issue is to combine the static and dynamic fragmentation method. For instance, we can first sort the tasks which can be executed at the same site into one group. When we do fragmentation, the tasks belong to the same group are regarded as one atomic task, and we only partition the model when the entire group of tasks (i.e., the *atomic task*) is completed. Further on we will device more powerful method to reduce the fragmentation overhead.

## Acknowledgements

This research is supported by Hi-Tech Research & Development Program of China (863) under grant 2003AA414032 and the National Natural Science Foundation of China under grant 60274046.

## References

[1] D. Georgakopoulos, M. Hornick, A. Sheth, An overview of workflow management: from process modeling to workflow automation infrastructure, Distributed and Parallel Databases 3 (2) (1995) 119–153.
[2] C. Mohan, G. Alonso, R. Guenthoer, M. Kamath, B. Reinwald, An overview of the exotica research project on workflow management systems, in: Proceedings of the 6th International Workshop on High Performance Transaction Systems, Asilomar, 1995.
[3] J. Miller, A. Sheth, K. Kochut, X. Wang, CORBA-based run-time architectures for workflow management systems, Journal of Database Management 7 (1) (1996) 16–27 (special issue on multidatabases).
[4] P. Muth, D. Wodtke, J. Weissenfels, A.K. Dittrich, G. Weikum, From centralized workflow specification to distributed workflow execution, Journal of Intelligent Information Systems 10 (2) (1998) 159–184.
[5] W.M.P. Van der Aalst, Loosely coupled interorganizational workflows: modeling and analyzing workflows crossing organizational boundaries, Information & Management 37 (2) (2000) 67–75.
[6] Y. Yan, Z. Maamar, W. Weiming Shen, Integration of workflow and agent technology for business process management, in: Proceedings of CSCW in Design 2001, London, Ontario, Canada, (2001), pp. 420–426.
[7] T.P. Cai, A.S. Gloor, DartFlow: a workflow management system on the web using transportable agents, Technical Report of Dartmouth College, Computer Science, Hanover, NH, 1996.
[8] J.M. Vidal, P. Buhler, C. Stahlet, Multiagent systems with workflows, IEEE Internet Computing 8 (1) (2004) 76–82.
[9] V.S.A. Atluri, S.A. Chun, P. Mazzoleni, A Chinese wall security model for decentralized workflow systems, in: Proceedings of the 8th ACM conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, 2001.
[10] R.S. Silva, J. Wainer, E.R.M. Madeira, A fully distributed architecture for large scale workflow enactment, International Journal of Cooperative Information Systems 12 (4) (2003) 411–440.
[11] G.J. Fakas, B. Karakostas, A peer to peer (P2P) architecture for dynamic workflow management, Information and Software Technology 46 (6) (2004) 423–431.
[12] J. Yan, Y. Yang, G.K. Raikundalia, Enacting business processes in a decentralised environment with p2p-based workflow support, Proceedings of Advances in Web-Age Information Management, Lecture Notes in Computer Science 2762, 2003, pp. 290–297.
[13] W.M.P. Van der Aalst, The application of Petri nets to workflow management, Journal of Circuits Systems and Computers 8 (1) (1998) 21–66.
[14] K. Jensen, Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use, Springer-Verlag, Berlin, 1992.
[15] D.S. Liu, J.M. Wang, S.C.F. Chan, J.G. Sun, L. Zhang, Modeling workflow processes with colored Petri nets, Computers in Industry 49 (3) (2002) 267–281.
[16] W.M.P. Van der Aalst, A.H.M. ter Hofstede, Verification of workflow task structures: a Petri net-based approach, Information Systems 25 (1) (2000) 43–69.
[17] T. Murata, Petri nets: properties, analysis and applications, Proceedings of the IEEE 77 (4) (1989) 541–580.
[18] W. Tan, Y. Fan, Model fragmentation for distributed workflow execution: a Petri net approach, Proceedings of the 5th IEEE International Symposium and School on Advance Distributed Systems (ISSADS 2005), Guadalajara, Mexico; Lecture Notes in Computer Science vol. 3563, Springer-Verlag, Berlin, 2005. pp. 207-214.
[19] H. Luo, Y. Fan, CIMflow: a workflow management system based on integration platform environment, in: Proceedings of the 7th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'99), Barcelona, Spain, 1999.



**Wei Tan** received his B.S. degree in Automation from Tsinghua University, Beijing, China, in 2002. He is currently a Ph.D. candidate in control theory and engineering at the Department of Automation in Tsinghua University. His research interests include business process management, distributed workflow technology, CIMS, Petri net, etc.

**Yushun Fan** received his B.S. degree in automatic control from Beijing University of Aeronautics and Astronautics, Beijing, China, in 1984, and his M.S. and Ph.D. degrees in control theory and application from Tsinghua University, Beijing, in 1987 and 1990, respectively. He is currently Professor of the Department of Automation, Vice Director of the System Integration Institute, and Director of the Networking Manufacturing Laboratory, Tsinghua University. His research interest includes enterprise modeling methods and optimization analysis, business process re-engineering, workflow management, system integration and integrated platform, object-oriented technologies and flexible software systems, Petri nets modeling and analysis, workshop management and control. He authored nine books in enterprise modeling, workflow technology, intelligent agent, and object oriented complex system analysis, computer integrated manufacturing, respectively, and published more than 250 research papers in journals and conferences. He is a member of the IFAC Advanced Manufacturing Technology Committee. From September 1993 to March 1994, he was a Visiting Scientist at the University Bochum, Germany, supported by Federal Ministry for Research and Technology. From April 1994 to July 1995, he was a Visiting Scientist, supported by Alexander von Humboldt Stiftung, at Fraunhofer Institute for Production System and Design Technology (FhG/IPK), Germany. Dr. Fan served on the Program Committees of the 1992 International Symposium on CIM, Beijing, China, 1997 IEEE International Conference on Factory Automation and Emerging Technology, Los Angeles, CA, and 2002 International Workshop on Emergent Technologies in Engineering Cooperative Information Systems, Beijing, China.