Component and Data Distribution in a Distributed Workflow Management System[†]

San-Yih Hwang
Department of Information Management
National Sun Yat-Sen University
Kaohsiung, Taiwan 80424

Chi-Ten Yang
Software Engineering Laboratory
Institute for Information Industry
Taipei, 106 Taiwan

Abstract

Designing a distributed workflow management system (WFMS) has become a topic with tremendous research interests in recent years. Unlike a traditional client-server WFMS that executes only at a single site, a distributed WFMS allows the execution of a workflow instance to be governed by more than one servers so as to achieve availability and efficiency. A consensus has generally been reached that the component-based approach is a promising one to designing a distributed WFMS. The component-based approach divides the functionality of a WFMS into a number of components, each of which follows some well-accepted distributed object standard, e.g., CORBA or DCOM. These components can be distributed transparently to different sites for execution, and cooperatively they achieve the goal of the underlying business process. Following the component-based approach, this paper proposes and discusses several alternatives on how to distribute the components and the data of a distributed WFMS. Specifically, the component distribution problem can be transformed into two combinatorial problems, namely the integer programming problem and the weighted matching problem of a bipartite graph, under different conditions. Regarding the data distribution, we recommend the full replication of workflow definition data and suggest different strategies in placing different types of instance data.

1. Introduction

To respond to the ever-increasing uncertainty and unprecedented volatility of the global business environment, many organizations have adopted process-oriented approaches to organizational change. Unlike the functional approaches traditionally employed by most organizations, the process-oriented approaches aim to integrate various kinds of resources in the organization, including human, existing applications, and tasks previously performed by humans, to achieve the goals of business processes. A workflow management system (WFMS) is a kind of computer software that intends to meet the requirements imposed by process-oriented approaches.

Since the emergence of workflow related terminologies in mid 1980s, more than 250 products have existed in the market that claim to support (some degree of) workflow management [6]. With the advent of internet/intranet, workflow management systems have even been seen as an ideal technological infrastructure for supporting the construction of virtual enterprises. In Taiwan, popular workflow management systems include IBM Flowmark, Lotus Notes, and MS Exchange.

The research community has also acknowledged the importance of workflow management and identified some limitations imposed by current workflow management systems [2][14]. A non-exhaustive list includes the following:

1. The process model behind the process definition tools is not powerful enough to express all kinds of workflow applications.
2. The correctness of a process definition is not ensured.
3. The large scale workflow applications are not properly supported.
4. Almost all commercial workflow management systems adopt a client-server architecture with server's functionality being executed at a single site. This causes reliability as well as load balancing problems.
5. Seamless integration to legacy systems is not generally supported.
6. Dynamic change to workflow process definition or workflow process instances is not adequately supported.

[†] This research was sponsored by MOEA and supported by Institute for Information Industry, R.O.C.

In an attempt to deal with the problems 3, 4, and 5, several researchers have adopted the component-based approach to designing a WFMS, and the reports on the implementation of these prototype WFMSs can be widely seen in the literature [3][4][10][12][13]. The component-based approach partitions the functionality of a WFMS into a set of components, each of which is wrapped up according to some standard object technologies, e.g., CORBA or DCOM, to provide transparent distribution of and access to the components. The standard object technology such as CORBA also serves as an ideal infrastructure for integrating legacy systems. By distributing components of a WFMS to various sites, failure to a single site will not impede the operation of all workflow processes.

However, although the component-based approach imposes no restrictions on how the various components are distributed, different strategies of distributing components still make differences in terms of efficiency and load balancing. In this paper, we address the issues on how to distribute the components and the needed data in a computing environment consisting of several local domains connected by computer networks. To the best of our knowledge, these issues have not been addressed in the literature by the workflow research community.

The rest of the paper is organized as follows. Section 2 describes the related workflow concepts adopted in this paper and proposes a distributed WFMS architecture by using the component-based approach . The component-based approach meets the functional requirements of a WFMS by designing a set of objects. These objects collaboratively work together to achieve the functionality of a WFMS. Section 3 discusses how to distribute these objects to facilitate efficiency and load balancing. As a WFMS needs to store both the definition data and instance data in a database, the data management plays an important role in the operation of a WFMS. Section 4 is devoted to the discussion of distributing the data required by a WFMS. Finally, Section 5 provides a summary and points out our future work.

2 Background

A workflow is the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules [16]. The description about the action performed by a participant is called an activity, which forms a logical step in the pertaining workflow process. Activities could be manually performed by humans or automatically conducted by a computer program or hardware. A workflow management system (WFMS) provides facilities to manage workflow processes, enabling users to define, execute, and monitor processes. A reference model proposed by the Workflow Management Coalition (WfMC) is shown in Figure 1 [15].

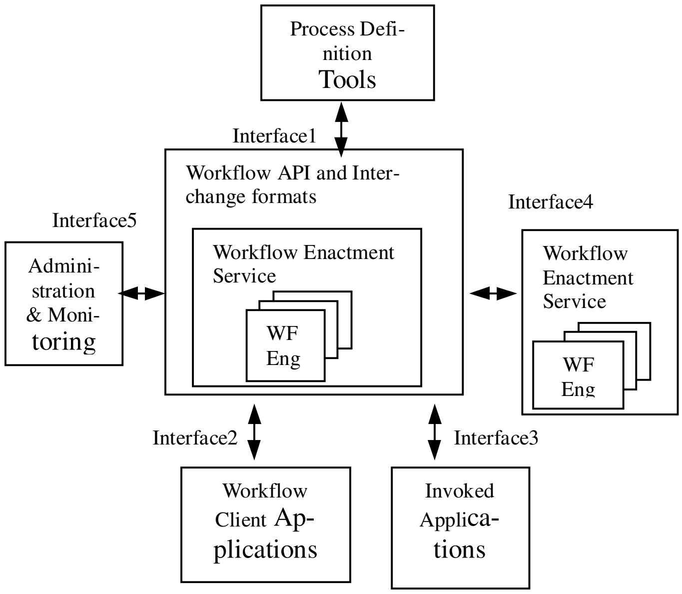


Figure 1. A Reference Model by WfMC

The central part of the model is the workflow enactment service, which provides the workflow repository and functions to meet the external requirements such as defining, activating, and monitoring processes, tracking the status about activities, and interacting with other workflow enactment service.

Several hundred products that exist in the market claim to support workflow management [6]. These systems are designed for office automation with emphasis on circulating documents from one participant to another. Besides, they adopt client-server architecture, in which the entire workflow enactment service is executed on a single platform. Such a centralized architecture suffers from scalability and reliability [11][14]. In recent years, a great deal of research interests has been focused on the design of a distributed WFMS, in which the functionality of the workflow enactment service is performed across various platforms. Several designs have been proposed and prototyped in the literature [3][4][10][11]. Of all the various designs, the component-based approach has been widely adopted. The component-based approach divides the workflow enactment service into software components, which collaboratively perform the workflow functionality. To enable seamless distribution and communication among components, emerging distributed object management technologies, e.g., CORBA and DCOM, are adopted. Besides, the component-based approach also solves the problem of integrating legacy systems by wrapping them up according to the adopted object standard.

The process definition tools usually provide graphical interfaces to ease the specification of workflow processes and data. This specification will be converted to an intermediate format called workflow specification language. A workflow specification language includes syntax for specifying data, activities, and the interdependencies among them. A typical process specification language is described in [17]. It was proposed by WfMC and intends to serve as an interchange language between those from different vendors. Taking an intermediate format about a workflow process as an input, the component-based approach generates code for activity managers, including scheduling components, invocation code, data access routines, and the recovery mechanisms. Each activity manager has the sole responsibility for conducting an activity, which could be manual or automated. For a manual activity, the activity manager may just place a work item on the worklist of each potential participant and start tracking the status about the work item. Once a work item is selected by a participant, the same work item will be removed from the work lists of the other potential participants. For an automated activity, the activity manager starts it by providing the necessary information. Once an activity manager determines that the associated activity is completed, it activates the successor activity managers, if any. This procedure continues until the last activity manager finishes its activity, which marks the end of a workflow process instance.

Note that each module in the component-based architecture, e.g., activity, activity manager, worklist handler, etc., is implemented as an object according to the adopted object standard. The object standard provides an infrastructure for the development of software in a distributed and heterogeneous environment. Thus, activity managers can be transparently distributed to different servers, and the communications between them rely on the facilities provided by the object standard. The architecture is depicted in Figure 2.

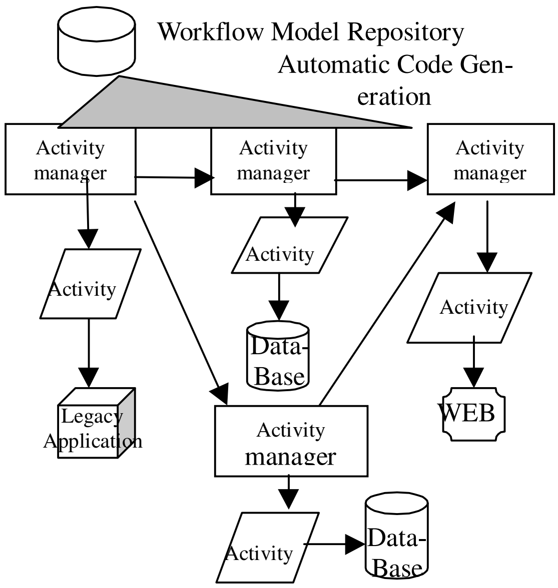


Figure 2. Architecture of a Distributed WFMS

In addition to the code, data also plays a crucial role in workflow applications. Workflow data can be classified into the following two categories:

- Definition data: This is the set of (meta) data specified by process definition tools. It includes the process and activity definitions, the control and data dependencies among activities, the definitions of relevant data whose values determine the control flow of a workflow process, and the organizational data that describes roles in an organization and the relationships among them.
- Instance data: This is the set of data generated when workflow process instances are executed. For each process instance, such information as its status, the constituent activity instances and their status, the values of the relevant data, and other statistical data, e.g., starting and ending time, is recorded.

A nice property with the component-based architecture is that objects can be distributed in any way across different platforms, and the internal structure of each object is hidden from other objects. However, to achieve an efficient workflow execution and to balance the loads incurred at different servers, the way objects and data are distributed should be carefully planned. The focus of this paper are (1) object distribution and (2) data distribution, which will be respectively discussed in detail in the next two sections.

3 Distributing Objects

Let us think of the entire computing environment as a composition of a number of local domains, in each of which a workflow server equipped with a database exists. With the component-based approach, several types of objects, including activity managers, worklist handler, etc., are generated to work cooperatively to meet the objectives of workflow processes. There are two types of activity managers, namely auto activity manager and user activity manager. An auto activity manager is in charge of the execution of an automated activity, whereas a user activity manager is responsible for interacting with persons via their worklist handlers. A web interface could be used by a staff member to browse her work-list. We will discuss in this section how to determine the local domain to which each object should be assigned.

As a matter of fact, for all types of objects except the activity managers, determining their residing local domains is straightforward. For example,

- an application object should be assigned to the local domain where the application is to be executed,
- a monitoring (activation) object should be assigned to the local domains where monitoring (activating) workflow processes can possibly occur, and
- as each participating staff member is associated with a worklist, a worklist handler object should be assigned to the local domain in which the asso-

ciated staff member (usually) locates.

In the component-based architecture, activity managers play a crucial role in ensuring the correct execution of activities. Much functionality, such as scheduling, invocation, data access, and recovery, is implement within an activity manager. Exactly how to assign activity managers to various local domains to achieve best performance is essential and will be the focus of the following subsections.

3.1 Expressing the Activity Manager Assignment Problem as an Integer Program

There are a number of factors that one may consider in distributing activity managers, including:

1. *An activity manager should be assigned to a local domain that is closer to all participants that can potentially conduct the corresponding activity.* For an automated activity, we can just assign the associated activity manager to the local domain where the automated activity is taken place. For a manual activity, however, there could be several candidates that are qualified of performing it. The activity manager has to interact with all the candidates by communicating with their worklist handlers. Worse, these qualified candidates may not reside on the same local domain.
2. *An activity manager should be assigned to a local domain that incurs less communications to its predecessor and successor activity managers.* In a distributed WFMS, an activity manager is activated by its predecessor activity managers and may be responsible for initiating the following activity managers. The local domain assignment has to take the communication overhead between activity managers into account.
3. *Do not assign too many activity managers to a local domain.* The server in each local domain has its capacity. Overloading a server not only hinders workflow execution efficiency but also underutilizes the resources in the entire computing environment.

To better understand the problem, in the following, we consider the above three factors and formally express the activity manager assignment problem as an integer program. Suppose we are required to assign the activity managers for a workflow process with n activities, and there are m local domains in the computing environment. $ActivityCost_i$ is the communication cost incurred in executing the activity manager i due to the interaction with its potential performers. If activity manager i is the immediate predecessor (in the control flow or data flow) of activity manager j, there exists some communication cost between them, which we call *transition cost*, denoted as $TransitionCost_{i,j}$. Let x_i be the local domain to which activity manager i is assigned. We want to minimize

$$\sum_{i=1}^{n} ActivityCost_i + \sum_{i=1}^{n} \sum_{j=1}^{n} TransitionCost_{ij}$$

Subject to

$$ActivityCost_i = Cost(i, x_i), \quad (1 \le i \le n)$$

$$TransitionCost_{ij} = (\alpha * Zero(x_i - x_j) + \beta * (1 - Zero(x_i - x_j)) * Transition(i, j), (1 \le i, j \le n)$$

$$\sum_{i=1}^{n} Zero(1 - x_i) \le Capacity_l, \quad (1 \le l \le m), \; 1 \le x_i \le m \quad (1 \le i \le n)$$

$$\textbf{\textit{where}} \qquad Zero(y) = \begin{cases} \textbf{\textit{0}}, & \textit{if} \quad y = \textbf{\textit{0}} \\ \textbf{\textit{1}}, & \textit{otherwise.} \end{cases}$$

$$Transition(i,j) = \begin{cases} \textbf{\textit{1}}, & \textit{if activity}_i \textit{ has immediate predecessor of activity}_j, \\ \textbf{\textit{0}}, & \textit{otherwise.} \end{cases}$$

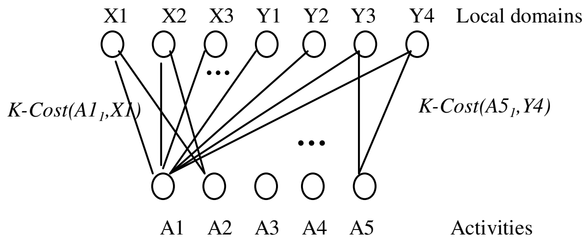
Note that Activity*Cost*$_i$ is determined by *Cost(i, x$_i$)*, the communication overhead needed in executing the activity manager i at the designated local domain x_i. If activity i is the immediate predecessor of activity j, a non-zero transition cost ($TransitionCost_{i,j}$) is incurred. In this case, if activity manger i reside in the same local domain as the activity manger j, $TransitionCost_{i,j}$ is α; otherwise, it is β. Of course, in most cases, β is much bigger than α. To better denote the equation, we introduce a step function *Zero(k)* that returns 1 when k=0 and 0 otherwise. To prevent a local domain from overloading, there is a limit on the number of activity managers that can be assigned to a local domain l, denoted as $Capacity_l$.

It is obvious that solving the above integer program efficiently is almost impossible, as a much simplified form of integer program, *the knapsack problem,* has already been shown to be NP-complete [1]. Many approximation algorithms have been written on books and papers that can be used to get a non-optimal solution in less time [7]. To get an optimal solution efficiently, in the following subsection, we convert the problem to a bipartite matching one by ignoring the transition cost between activity managers. This greatly reduces the computation complexity in solving the activity manager assignment problem.

3.2. Expressing the Activity Manager Assignment Problem as a Bipartite Matching One

The interactions between an activity manager and its (potential) performers could be substantial. Consider a manual activity. The activity manager first has to inform the worklist handlers of all staff members that are capable of performing it. When one staff member decides to take the activity, it then informs the activity manager, which in turn tells the other worklist handlers to drop the work item from their work lists. During the execution of the activity, the participant may communicate with the activity managers several times to suspend, resume, or terminate the activity. By contrast, the interaction between different activity managers is less frequent, especially with sequential/conditional/parallel control/data connectors, which make up most of the inter-activity dependencies in most cases. In this subsection, we ignore the transition costs between activity managers and formulate the problem as a weighted matching one of a bipartite graph, which can be solved much more efficiently.

We follow the notations used in the last subsection. That is, *Cost(a, d)* indicates the communication overhead incurred in executing *a* when its activity manager is assigned to local domain *d*; $Capacity_d$ is the maximum number of activity managers that can be accommodated in the local domain *d*. Let *G=(S, T, A)* be an undirected bipartite graph with *S* and *T* being the set of activity nodes and the set of local domain nodes respectively, and A containing edges between S and T. Note that each activity node in *S* uniquely represents an activity manager, while a local domain *d* is replicated $Capacity_d$ times in *T*. Let *K* be a sufficiently large number[1]. The weight associated with an edge *<a, d>* is defined as *K – Cost(a, d)*. The goal is to find a matching in the bipartite graph for which the sum of weights of the edges is maximum. For example, suppose there are two local domains X and Y in the computing environment and five activity managers for a given workflow process. The capacities of X and Y are 3 and 4 respectively. The corresponding bipartite graph is shown below.



The goal is to find a maximum match in the above bipartite graph. This problem can be solved in polynomial time[2].

4. Distributing Data

Once the distribution of software components have been determined, the next step is decide on where the associated data should locate. Intuitively we should place the needed data close to the relevant component as much as possible. However, a moment of thoughts shows that this is not as simple as it first appeared. Different types of data may have different access and update pattern. In this section, we discuss alternatives for dispatching/replicating various types of data by considering the ways data is accessed.

As mentioned in Section 2, there are two types of data, namely workflow definition data and workflow instance data. In [11], the authors assume the workflow definition data to be fully replicated among all local domains. We share their opinion in that definition data is rarely updated but intensively accessed during activity execution. For example, to execute a manual activity, the activity manager has to first find out the role that is assigned to this activity from the activity definition, then get the identifiers about the persons that fulfill this role from the organizational data. When the workflow definition data is available at each local domain, the scheduling at each activity manager can be greatly speeded up. To update the definition data that is fully replicated, it takes much more time as some transactional facility has to be employed to ensure data consistency. We think it is worthwhile to pay the price as updates to workflow definition data rarely occur.

However, regarding the instance data, it is not obvious whether we should partition it, replicate it, or just keep a central repository. The following two subsections discuss in detail about how to place process (activity) instance data and relevant data respectively.

4.1. Placing Process and Activity Instance Data

Before coming to the discussion of data placement

[1] In this case, *K* must be no less than $Max_d(Capacity_d)$ to make the weight associated with each edge non-negative.

[2] Let |*S*| be the number of activity managers and |*T*| be $\Sigma_d(Capacity_d)$. Let $m = Max(|S|, |T|)$ and $n = Min(|S|, |T|)$. The Hungarian method for solving the bipartite weighted matching problem takes $O(m^2n)$ [9].

about process and activity instances, let us take a look on their schema. Since the data schema is rarely discussed in the literature, we modify the schema suggested by WfMC [18] and propose our own schema for illustration purpose:

Process_Instance_Table

ProcessInstance ID	InitialProcessInstanceID	ProcessName	ProcessStatus

StartTime	EndTime	Activator

Activity_Instance_Table

ProcessInstanceID	ActivityInstanceID	Status	Start Time	End-Time	Role-Name	ParticipantID

Each process instance has an entry in the Process_Instance Table that include values about its id, its status, its starting and ending time, the id of the person who activated it. As a process can be nested, a process instance could be several levels down in the hierarchy from the top. The process instance at the top of the hierarchy activated by the user is called initial process instance, and its id is recorded by every succeeding lower level process instance for future queries. The reason why we keep the initial process instance id in Process_Instance Table will become clear later.

Similarly, each activity instance stores the information in the Activity_Instance Table about its id, its status, its starting and ending time, and the id of the pertaining process instance. Besides, the ids of the role and the user that executed the activity are also recorded if it is a user activity.

The information about an activity instance is recorded by its activity manager. To speed up the processing of activity managers, it is reasonable to horizontally partition Activity_Instance Table such that each local domain has a table to store the information about the activity instances executed locally.

A process instance is created by a user who decides to execute a process.

By using the administrating/monitoring tool, its information may be subsequently modified to suspend, resume, or terminate its execution, or accessed by tracking its status. When the last activity of the process instance is executed, the corresponding activity manager marks the completion of the process instance. During the course of process execution, the information about the process instance remains intact. There are three approaches in placing the process instance data:

1. Keep a single, central repository for storing all process instance data
2. Distribute the process instance data such that the information about a process instance is placed at the local domain where it is created.
3. Fully replicate the process instance data at all local domains.

The first and the second approaches have similar cost in updating process instances because subsequent modification to a process instance data may not occur at the same domain as where it is created. The third approach, by contrast, incur much overhead in terms of update cost. Let us now look into the cost for monitoring a process instance. Suppose a user would like to know the progress of an insurance claim process instance activated by John Chen. The following three steps are needed to process this query:

1. Find out the initial process instance id activated by John Chen. The SQL statement against our schema is

```
SELECT InitialProcessInstanceID
FROM PROCESS_INSTANCE_TABLE
WHERE ProcessName = 'Insurance Claim'
AND Activator = 'John Chen';
```

2. Find out the id of all process instances with the initial process instance id being that returned by Step 1. Let iid be the process instance id returned by Step 1. The corresponding SQL statement is

```
SELECT ProcessInstanceID
FROM PROCESS_INSTANCE_TABLE
WHERE InitialProcessInstanceID = $iid$;
```

3. Find out the ids of the activity instances with status being active and the pertaining process instance ids being in the set of process instance ids returned by Step 2. Let sid be the set of process instance ids returned by Step 2. The following is the SQL statement for Step 3:

```
SELECT ActivityInstanceID
FROM ACTIVITY_INSTANCE_TABLE
WHERE ProcessInstanceID IN $sid$ AND
        Status = 'Active';
```

One may wonder why not combining the three steps by using a single SQL statement that joins three tables. This argument is valid for approach 1 and 3. In this case, the server in each local domain independently executes the above three steps. Since it has been decided that ACTIVITY_INSTANCE_TABLE be horizontally partitioned and distributed to various local domains, when all servers finish execution, the results have to be collected and presented to the user as a whole. On the other hand, if approach 2 is adopted and PROCESS_INSTANCE_TABLE is partitioned and distributed to various local domains, the three steps have to be conducted one after another at each local domain, and after each step, a synchronization is needed to collect and distribute the results before the next step can be started.

We next discuss the reliability problem. The biggest problem with the first approach is the possibility of *single site failure*, by which we mean the crash of the site that stores PROCESS_INSTANCE_TABLE will prevent the operations of workflow process initialization, suspension, resumption, and termination (though the activity execution will not be affected). The chance of failure to the server in the local domain that stores PROCESS_INSTANCE_TABLE can be greatly reduced by adopting some hardware approach, e.g., by employing a fault-tolerant machine, or software approach, e.g., by using a replication server to store a secondary copy at another site. Nevertheless, if the reliability is a major concern, the third approach incorporated with some consistency maintenance mechanism in a partitioned network to handle the failure problems [5] becomes the best choice. The comparison can be summarized in the following table:

Approach	Update Efficiency	Monitoring Efficiency	reliability	Ease of Implementation
1	high	high	low	high
2	high	low	medium	medium
3	low	high	high	low

4.2. Distributing Relevant Data

Relevant data can appear in the conditions of various places, e.g., transitions between activities, indicating whether or not the end of an activity should trigger the start of another, the precondition of an activity under which the activity can be started, and postcondition of an activity, signifying the successful completion of the activity. Since each process instance is an independent thread of execution, it has its own set of relevant data values. The issue we are look at in this section is, for each process instance, where the relevant data should be placed. There are three strategies:

1. Put all the relevant data values in a single site.
2. Deliver the relevant data to the required activity managers by following the data flow specified in the process definition.
3. Fully replicate the relevant data values at all local domains.

The third strategy will not be discussed since it incurs much overhead in storing data. The first strategy is simple. However, it suffers from the reliability problem. Besides, requiring all updates to the relevant data be stored in a fixed and possibly remote database is likely to cause performance problem. Thus, we recommend the use of the second strategy. This strategy can be implemented in a pushing manner. By pushing, we mean that, at the end of an activity, the required relevant data values are sent to the local domain of the activity at the other end of a data flow edge. However, since a process definition may contain many conditional branches, the activity at the destination end of a data flow edge may not be executed afterwards. To eliminate the unnecessary data deliveries,, we suggest the use of another implementation, namely *data pulling*. For pulling, when an activity at the destination end of a control flow edge is initiated, its activity manager requests the local domains in which the source activities of these edges reside to send the required relevant data. If the source and destination activities of a data flow edge happen to reside in the same local domain, no communications are needed.

5. Conclusions

This paper addresses the object and data distribution problems in a distributed WFMS designed by the component-based approach. Regarding the object distribution, we have considered the communication cost for executing an activity, that for activity transition, and the maximum load allowed at a single site. An integer program is formulated accordingly. If the activity transition cost can be ignored, which is valid in most cases, the problem can be further transformed to a weighted matching one of a bipartite graph. The resultant problem can be solved more efficiently. With respect to the data distribution, we considered several alternatives and reached the following conclusions:

1. Workflow definition data, including process and activity definitions, organizational structures, and relevant data definition, should be fully replicated at all sites.
2. Information about activity instances should be distributed to the site on which the corresponding activities are executed.
3. Process instance data can be centralized stored for efficiency reason. However, if reliability is to be a big concern, fully replication of the process instance data can be adopted with the sacrifice of efficiency.
4. Relevant data values are dynamically sent to a site by following the data flow connectors.

The distributed WFMS architecture described in this paper has been designed by using UML methodologies. Due to space limitation, we do not show their detailed design here [8]. The distribution of application data is not discussed in this paper as it is beyond the control of the WFMS. However, if some historical information about the application data is available, its placement can be carefully planned. This is part of our future work.

References

[1] A. Aho, J. Hopcroft, and J. Ullman. Data Structures and Algorithms, Addison Wesley, 1983.

[2] G. Alonso, D. Agrawal, A. Abbadi, and C. Mohan. Functionalities and Limitations of Current Workflow Management Systems, Technical Report, IBM Almaden Research Center, 1997.

[3] A. Dogac, E. Gokkoca, S. Arpinar, P. Koksal, I. Cingil, B. Arpinar, N. Tatbul, P. Karagoz, U. Halici, and M. Altinel, “Design and Implementation of a Distributed Workflow Management System: Metaflow,”

Proc. of NATO Advanced Study Institute on Workflow Management Systems and Interoperability, 1997.

[4] S.Das, Kl Lochut, J. Miller, A.Sheth, D. Worah: “ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for METOR_2”, Technical Report #UGA-CS-TR-97-001, Department of Computer Science, University of Georgia, February 1997.

[5] S. Davidson, H. Garcia-Molina, and D. Skeen. “Consistency in Partitioned Networks,” ACM Computing Survey, Vol. 17, No. 3, 1985.

[6] D. Georgakopoulos, M. Hornick, and A. Sheth,“ An Overview of Workflow Management: from Process Modeling to Workflow Automation Infrastructure,” *Distributed and Parallel Databases*, Vol.3, No.3 , 1995

[7] T. C. Hu, Integer Programming and Network Flows, Addison Wesley Publishing, 1970.

[8] S.-Y. Hwang, Final Report to the Research on Analysis, Design and Feasibility of the Architecture of Distributed Workflow Systems (in Chinese), Project No. 0064, 1998.

[9] E. L. Lawler. Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, 1973.

[10] J. Miller, D. Palaniswami, A. Sheth, K. Kochut and H. Singh, “WebWork: METEOR2’s Web-Based Workflow Management System,” Journal of Intelligent Information Systems, Vol. 10, No. 2, 1998.

[11] C.Mohan, G. Alonso, M.Kamath.., “Failure Handling in Large Scale Workflow Management Systems, Research Report RJ 9913, IBM Almaden Research Center, November 1994.

[12] P. Muth, D. Wodtke, J. Weissenfels, A. Dittrich and G. Weikum, “From Centralized Workflow Specification to Distributed Workflow Execution,” Journal of Intelligent Information Systems, Vol. 10, No. 2, 1998.

[13] R. Schmidt and U Assmann, “CompFlow: A Component-based and Aspect-separated Architecture for Workflow Support,” Proc. of EDBT Workshop on Workflow Management Systems, 1998.

[14] A. Sheth and K. J. Kochut. “Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems,” Proc. of NATO Advanced Study Institute on Workflow Management Systems and Interoperability, 1997.

[15] Workflow Management Coalition, The Workflow Reference Model, 1994.

[16] Workflow Management Coalition Terminology, Workflow Management Coalition, 1996.

[17] Workflow Management Coalition Interface 1: Process Definition Interchange, Workflow Management Coalition, 1996.

[18] Workflow Management Coalition Audit Data Specification, Workflow Management Coalition, 1996.