A Data Storage Mechanism for Peer-to-Peer Based Decentralised Workflow Systems

Jun Yan¹, Yun Yang¹ and Gitesh K. Raikundalia¹,²

¹ CICEC - Centre for Internet Computing and E-Commerce, School of Information Technology
Swinburne University of Technology
PO Box 218, Hawthorn Melbourne, VIC 3122, Australia
Email: {jyan, yyang}@it.swin.edu.au
² School of Computer Science and Mathematics
Victoria University
P.O. Box 14428, Melbourne City, MC 8001, Australia
Email: Gitesh.Raikundalia@vu.edu.au

Abstract This paper proposes an innovative data storage mechanism for decentralised workflow systems. The research is based on a peer-to-peer workflow architecture in which various nodes communicate with each other directly to carry out workflow functions. In this approach, a process is partitioned into separate tasks, and then individual tasks are passed to appropriate nodes. To realise this approach, a data repository independent process notation is provided. Moreover, this paper also designs communication protocols to maintain data in a dynamic environment. The key distinction of this approach is to propose a “know what you should know” policy to reflect the nature of applications.

1 Introduction

To survive in a globally competitive environment, many organisations have adopted Business Process Reengineering (BPR) to achieve dramatic performance improvement [7]. As an enabling technology for BPR, workflow, which is the automation of a business [12], has attracted the attention of many researchers, vendors and users, and experienced tremendous growth for the last decade. A number of conferences and workshops have been held to present research in the workflow area. At the same time, many commercial workflow products have been developed. Moreover, workflow technology is moving towards standardisation. Workflow Management Coalition (WfMC) and Object Management Group (OMG) have published a series of standards to decrease the risk of developing workflow products [11, 12].

Although workflow technology has been heavily addressed, appropriate methods for distributed storage of process data have been largely ignored for years. This is simply because almost all the current workflow management systems adopt a centralised client/server architecture with all the build-time data being stored at a single site. However, while the client/server architecture has faced many challenges, there is a growing trend that the next generation of workflow will be built in a truly distributed manner [6, 15]. These decentralised workflow systems are supported by decentralised architectures, which remove central data repositories and central workflow engines from systems to ensure fault tolerance, enhanced scalability and better human support. Hence, the question of how to store process definition data without a central data repository has arisen and been acknowledged as an extremely important issue. Unfortunately, this problem has not been addressed well so far.

This paper proposes a data storage mechanism to store process definition data during build-time. In this approach, a workflow system is treated as a peer-to-peer (p2p) system with nodes within the system communicating with one another directly. Process definition data is partitioned appropriately and distributed to individual nodes. The rest of this paper is organised as follows. In the next section, a decentralised architecture for workflow support, which removes the servers from workflow systems completely, is described. In section 3, the data storage approach is detailed, including data partition, data distribution and data maintenance. Section 4 is an illustration example, followed by a comparison with major related work in section 5. Finally, the conclusions and authors’ future work are discussed in section 6.

2 Background

In workflow research community, there are some unsolved problems such as bad performance, poor scalability and user restriction [10]. From a system architecture viewpoint, these problems partly, if not
wholly, result from the mismatch of application requirements and conventional approaches. Given the nature of the application environment and the technology involved, workflow applications are inherently distributed [2, 15]. However, because of the benefits like easy auditing and design simplicity, most existing process support tools use centralised client/server software architectures [6, 14]. Therefore, to address the above weakness and have a cost-effective workflow environment, it is essential to design a decentralised architecture for workflow support that does not imply the presence of (1) a centralised data repository, nor (2) a centralised workflow engine for coordination.

On the other hand, p2p, which can be defined simply as the sharing of computer resources and services by direct communication, is driving a major shift in the area of genuinely distributed computing [1]. In a p2p system, while each peer behaves as a client in the client/server model, a peer is also capable of performing server functions. Technically, p2p enables better scalability, eliminates the risk of single-source bottleneck, and provides load balance, by distributing data and control.

Combining the concepts of workflow and p2p, an innovative p2p-based decentralised workflow architecture known as SwinDeW has been presented in [13], which designs a workflow system as a p2p system with each node being treated as a peer. In general, each node functions independently and interacts with other nodes by direct communication to fulfil key workflow functions. In this approach, some key terms are defined as follows:

Def 1: A p2p-based decentralised workflow management system is a workflow management system that adopts p2p technology to carry out workflow functions in a decentralised manner.

Def 2: A node is denoted as a software residing on a physical machine to enable direct communication with other nodes to carry out workflow while, in most cases, a node is associated with a human being.

Def 3: A community, which is characterised by a specific capability, is a cluster of nodes, with this capability, connecting logically.

3 Process Definition Data Storage Mechanism

Based on the SwinDeW architecture, this paper further investigates issues of process definition storage without the help of a centralised data repository. With respect to a philosophy that individual node needs to be concerned with only a part of the process, this section introduces a data storage mechanism which is capable of:

1. partitioning a process into individual tasks;
2. distributing individual tasks to appropriate nodes; and
3. maintaining the process data properly in order to accommodate system environment changes.

3.1 Data Partition

According to the WfMC, a process definition is the representation of a business process [12], which is a network of tasks and their relationships. As shown in Figure 1, in an ideal process model, each task is concerned only with its predecessors and successors. Thus, to extract a task from a process is to separate the task and its context from the process. This paper uses a six-tuple \( T(\text{processid}; \text{taskid}; \text{Cpre}; \text{Cpost}; \text{capability}; \text{resourceset}) \) to represent a data partition, which is denoted as a notation of a task extracted from a workflow process:

- \( \text{processid} \): process identifier.
- \( \text{taskid} \): task identifier.
- \( \text{Cpre} \): pre-condition (\( \text{taskset1}, \text{logic} \)). The relationships between the task and its predecessors.
- \( \text{Cpost} \): post-condition (\( \text{taskset2}, \text{logic} \)). The relationships between the task and its successors.
- \( \text{taskset1} \): \( (\text{taskid1}, \text{input data1}) \mid \text{Cpre} \), \( (\text{taskid2}, \text{input data2}) \mid \text{Cpre}, \ldots) \). Preceding tasks with corresponding input data.
- \( \text{taskset2} \): \( (\text{taskid1}, \text{output data1}, \text{threshold1}) \mid \text{Cpost} \), \( (\text{taskid2}, \text{output data2}, \text{threshold2}) \mid \text{Cpost}, \ldots) \). Successive tasks with corresponding output data.

Here, the threshold indicates the condition when the task can notify the next step.

- \( \text{logic} \): (and, or, xor, selection, none). By combining either an incoming or an outgoing flow with logic, a variety of types of logic operators are produced. The operators produced are and in, and out, or in, or out, xor in, xor out, selection in, selection out, none in, none out. Those logic operators related to the incoming flow will determine the start condition of this task. For example, and in means the task can start only after it has received notifications from all its predecessors. On the other hand, those logic operators related to the outgoing flow sometimes invoke a decision-making process. For example, xor out means the task will determine which way the outgoing flow will go according to the particular condition.
- \( \text{capability} \): The user capability that is necessary to fulfil the task.
- \( \text{resourceset} \): (\( \text{resource1}, \ldots \)). The resources that are necessary to fulfil the task, for instance, external system applications.
By means of data partition, a process is converted into a list of individual tasks for distribution after it is defined by a node known as definition node, the. In addition, with the unique process identifier, it is very easy to unite various tasks to form a complete process.

### 3.2 Data Distribution

After a process is divided into partitions, the data storage issue comes down to the situation of how to distribute a task specification, i.e., to which nodes a specific task should be distributed. Consider a real-world organisation, staff members normally deal with the work in accordance with the roles they play. Therefore, each node in the workflow system merely needs to know, and has to know, the details of the tasks that this node has the capabilities to carry out. Following this philosophy, the data distribution proposes a policy named “know what you should know”, which only distributes appropriate data to appropriate nodes with the matched capability.

SwinDeW uses a loosely-coupled topology. The nodes are clustered together to form virtual communities. In a workflow scenario, the participants with one specific capability are in the same community and capable of discovering each other. A node with multiple capabilities is associated with multiple communities so that it connects all nodes in these communities. In addition, some external means like organisational management are used to assure that no single node is isolated. By these means, a node discovery service is offered eventually. A node can always locate another node with the support of this service.

Based on the description above, definition node N₁ distributes a data partition to all the nodes with matched capability in three steps:
1. N₁ locates one node with required capability, say N₂;
2. then N₁ distributes the data partition to N₂;
3. node N₂ then propagates the data partition within the virtual community to those other nodes with the same capability.

### 3.3 Data Maintenance

All previous discussion is based on an assumption that the workflow management system is static. However, workflow systems are always considered dynamic. Consequently, the main challenge in supporting dynamic data storage is how to maintain process data in accordance with nodes’ dynamic behaviours.

A new node joins a workflow system by joining a virtual community according to its capability, which may be configured by external administrative tools. By announcing an “I am joining with capability XXX” message, a new node establishes connections with the existing nodes with the same capability, and then retrieves all the related process definition data from one of them. If a new node has more than one capability, it will join the various communities separately.

When a node leaves a workflow system, it actually leaves all the virtual communities involved. Normally, a node sends an “I am leaving” message to notify the system about its absence. In the case of exceptions, a node may leave a workflow system quietly without notifying other nodes. To keep the system informed of all, another “I am active” message is propagated by each node periodically. If a node has not been heard from for a period of time, it will be removed automatically and an exception message would be generated to notify the system manager.

There are occasions that a node changes its capabilities; for example, a staff member is trained in further capabilities. Thus, a node may join or leave some communities according to the alteration. At that time, it can update the corresponding process definition by means of the procedures described above.

### 4 An Example

To better illustrate how data storage works with p2p, this section applies the mechanism described in the previous section to a simplified home loan application, which is a typical workflow application. Assume the loan application process comprises six tasks as depicted in Figure 2(a). At the same time, four physical nodes, representing four users, Tim, John, Mary and Lisa, are involved in this workflow. As depicted in Figure 2(b), these four
participants form five virtual communities according to their capabilities. Given the mechanism described, the home loan application process is divided into six separate tasks with relationships among them. For example, the task with 
\[T_{12}\] can be represented as: 
\[
(\text{credit checking report}) \land (\text{credit checking report}) \lor \\
(\text{approved application form}, \text{finished}) \lor (\text{unapproved application form}, \text{finished}) \lor (\text{approval}) \lor (\text{loan approval})
\]
Thus, each user only keeps a part of the process definition. Figure 3 shows process definition data on John’s computer. As he joins virtual communities 1 and 3, he only has knowledge about two tasks, i.e., 
\[T_{12}\] and 
\[T_{45}\], in this process.

After the process is partitioned, process definition data will be passed to different virtual communities. In this case, definition of 
\[T_{12}\], 
\[T_{23}\], 
\[T_{45}\], 
\[T_{56}\], and 
\[T_{67}\] will go to communities 1, 2, 3, 4, and 5, respectively. Thus, each user only keeps a part of the process definition.

Figure 3 shows process definition data on John’s computer. As he joins virtual communities 1 and 3, he only has knowledge about two tasks, i.e., 
\[T_{12}\] and 
\[T_{45}\], in this process.

5 Related Work

Although the research community has acknowledged the importance of a decentralised workflow management system, little attention has been given to the data storage issue. To the best of authors’ knowledge, very little literature mentioned this issue and only few results were published. In short, the main mechanisms used in supporting process data storage in decentralised workflow systems are listed as follows:

1. Retaining a central database. Some partly-decentralised workflow systems still retain a central database in the system. Users still have to access a single node to retrieve the data. For example, Endeavors [8] is derived directly from the web-based client/server model and relies on a single HTTP workflow server to store data. This kind of approach is easy to implement, as the client/server architecture is considered a mature model. However, problems related to the centralised server may still remain.

2. Replicating data at each node. Some approaches simply replicate all data at each node. For example, [9] fully replicates the workflow definition data based on the assumption that definition data is rarely updated but intensively accessed. However, full replication is clearly resource-consuming and error-prone. Also, due to the frequently changing environment and the increasing size of the system, this kind of approach is clearly impractical and unnecessary.

3. Data migration. Another way to store data is migrating data from one place to another either by agents or multiple workflow servers. For example, DartFlow [4] uses transportable agents as the backbone to control execution of process instances. The agents travel from one node to another, carrying essential process information. Normal nodes do not have to keep data at all. [3] and [5] use multiple workflow servers and migrate instance level data from one server to another dynamically to achieve better efficiency. However, this kind of approach always requires complicated communication languages and complex algorithms to control data migration. Moreover, workflow systems developed with this architecture are normally expensive and inflexible, which make it perhaps only feasible for large organisations while unaffordable for SMEs (small and medium enterprises).

4. Data partition and data distribution. The first contribution of such mechanisms is Exotica/FMQM [2]. Exotica/FMQM compiles process definition to determine the information relevant to each node and distribute different information to different nodes. However, Exotica/FMQM has not addressed data distribution and maintenance issues in a p2p-like dynamic environment.

In contrast, the approach presented in this paper proposes a policy of “know what you should know” with the consideration of the fact that each node involved in a process knows neither nothing nor all. The main contribution of this approach is providing an effective mechanism to partition information, distribute information, and maintain information. The process representation is natural and effective. The communication protocols are simple and practical. The mechanism is independent of the data repository and can be realised without any external tool. Moreover, this approach enhances the system flexibility. Individual user’s
behaviour such as joining and leaving will not influence the whole system because the data is shared locally.

6 Conclusions and Future Work

Decentralised workflow research emerged as it demonstrated great advantages in comparison with its centralised counterpart. However, distributed data storage, which is one of the key issues in supporting decentralisation, has been largely overlooked. Most current approaches do not genuinely address this issue as they normally simplify this issue for reasons such as easy implementation. Other solutions are too complex to be effective. As a response to the state-of-the-art, this paper addresses distributed data storage problems for decentralised workflow support and proposes an innovative approach, which:

(1) regards a decentralised workflow management system as a p2p system with each node being an independent peer in virtual communities;
(2) proposes a “know what you should know” policy when storing data and bases distribution on the capability attribute;
(3) designs a data distribution protocol with the support of a node discovery service; and
(4) implements various protocols to maintain data in a dynamic environment.

With respect to the above merits, the appropriate data storage in this approach accords with the distributed nature of the real world application. Thus, the mechanism can better support decentralised process execution. In the future, further research on decentralised workflow support will be conducted. Data storage will be extended to the instance level so that process execution data can be distributed as well. Mechanisms of control distribution, for enabling process instances to be enacted as defined, will be investigated. Issues such as exception handling, dynamic changes to process instances or process definitions, and so on, will also be further explored.

Acknowledgement

This work is partly supported by Swinburne Vice Chancellor’s Strategic Research Initiative Grant 2002-2004.

References