SwinDeW – A p2p-based Decentralised Workflow Management System

Jun Yan, Yun Yang, *Member, IEEE,* and Gitesh K. Raikundalia, *Member, IEEE CS*

***Abstract*—Workflow technology undoubtedly has been one of the most important domains of interest over the past decades, from both research and practice perspectives. However, problems such as potential poor performance, lack of reliability, limited scalability, insufficient user support and unsatisfactory system openness are largely ignored. This research reveals that these problems are mainly caused by the mismatch between application nature, i.e., distributed, and system design, i.e., centralised management. Therefore, conventional approaches based on the client-server architecture have not addressed them properly so far.**

The authors abandon the dominating client-server architecture in supporting workflow because of its inherent limitations. Instead, the peer-to-peer infrastructure is used to provide genuinely decentralised workflow support, which removes the centralised data repository and control engine from the system. Consequently, both data and control are distributed so that workflow functions are fulfilled through the direct communication and coordination among the relevant peers. With the support of this approach, performance bottlenecks are likely to be eliminated whilst increased resilience to failure, enhanced scalability, and better user support are likely to be achieved. Moreover, this approach also provides a more open framework for service-oriented workflow over the Internet.

This paper presents the authors' innovative, decentralised workflow system design. The paper also covers the corresponding mechanisms for system functions and the SwinDeW prototype which implements and demonstrates this design and functions.

***Index Terms*—Coordination, Decentralisation, Peer-to-peer, Workflow management**

I. Introduction

Workflows automate business procedures for passing documents, information or tasks from one participant to another, according to a defined set of rules. During the past two decades, the workflow undoubtedly has been one of the most important domains of interest. A number of conferences, workshops and symposia have been organised for researchers to discuss a wide range of topics in the workflow area and numerous contributions have been published (e.g., [1], [12], [13], [17]). At the same time, empirical workflow practice is also thriving. A vast variety of commercial Workflow Management Systems (WfMSs) are available, such as ActionWorkflow (Action Technologies), FlowMark (IBM), Staffware (Staffware), InConcert (TIBCO), FileNet Ensemble (FileNet), and so on. It is observed that organisations are increasingly using workflow systems for supporting their business. This is because the deployment of workflow may result in greater efficiency, better process control, improved worker productivity, enhanced flexibility, and process improvement.

Although workflow research and practice have reached a certain degree of maturity, some limitations of the existing approaches have been recognised. The state-of-the-art in workflow management has so far been determined by functional aspects provided in workflow systems. Issues such as performance, reliability, scalability, user support, and system openness are hardly ever considered in the development of existing workflow systems. Thus, conventional workflow systems have exhibited common weaknesses such as poor performance, lack of reliability, limited scalability, user restriction and unsatisfactory system openness [4], [14], [32]. The authors argue in this paper that these weaknesses mainly result from the architectural limitations of conventional workflow systems. To understand this, it is necessary to identify the gap between workflow features and the existing solutions. On one hand, given the nature of the application environment and the technology involved, workflow applications are inherently distributed, or even decentralised [3], [31]. On the other hand, almost all the current workflow systems use the dominating client-server architecture, which provides centralised coordination and control. The popular adoption of client-server technology for workflow systems is understandable. As a proven technology, client-server technology is able to satisfy functional aspects of workflow like process management and coordination. Moreover, client-server technology also offers benefits such as thin clients, centralised monitoring and auditing, simple synchronisation mechanisms, and ease of design and implementation for workflow systems. These benefits have been naturally exploited to support workflows in traditional application domains like office and banking environments since the advent of the workflow technology. However, because of its inherent feature, i.e., centralised management,

Manuscript received May 6, 2004. This work was supported in part by Swinburne Vice Chancellor's Strategic Research Initiative Grant 2002-2004.

Jun Yan and Yun Yang are with the Faculty of Information and Communication Technologies, Swinburne University of Technology, PO Box 218, Hawthorn, Melbourne, Australia 3122 (phone: 61-3-92148860; fax: 61-3-98190823; e-mail: {jyan, yyang}@swin.edu.au).

Gitesh K. Raikundalia is with the School of Computer Science and Mathematics, Victoria University, PO Box 14428, Melbourne City, Australia 8001 (e-mail: gitesh.raikundalia@vu.edu.au).

client-server technology is not the ideal supporting technology for distributed workflows in some application domains because it encounters some difficulties to satisfy non-functional aspects of workflow like performance and scalability, as detailed in Section II. The mismatch between application nature, i.e., distributed, and system design, i.e., centralised management, may result in serious problems as pointed out above. Current research based on client-server either addresses these problems partially, or inevitably increases the complexity of the already sophisticated workflow systems.

This paper reports distinct research addressing the above problems rudimentally from an architectural perspective. In this paper, an innovative, decentralised workflow system called SwinDeW is proposed, which is based on the peer-to-peer (p2p) infrastructure [2]. Briefly, this new paradigm changes the workflow system design fundamentally by using p2p rather than the client-server architecture, which differentiates it largely from the previous efforts addressing these problems. This research first presents a distinctive workflow system design, which has no need of centralised servers and hence removes the servers—the main causes of the above problems—from the system. Then the corresponding mechanisms supporting both build-time and run-time workflow functions are given. In addition, a prototype is also contributed for demonstration and proof-of-concept purposes.

The rest of the paper is organised as follows. In the next section, the motivations of the research are further illustrated, and the system requirements are analysed. Section III presents the system design which combines workflow technology with p2p. Section IV then explains approaches to providing system functions in a decentralised environment. After that, Section V and Section VI present a JXTA-based prototype implementation and a case study, respectively. In Section VII, major related work is introduced and the pros and cons of the presented approach are discussed. Finally, Section VIII concludes this paper and outlines the authors' future work.

II. Motivations and Requirements Analysis

Workflows are normally managed by a WfMS which consists of software components to create, direct and monitor the execution of workflows. A WfMS provides support in two key areas: build-time and run-time functions. Build-time functions deal with the modelling, representation and storage of processes. Run-time functions handle the execution of a process instance, including the procedure of process instantiation, work assignment and task navigation. These functions always involve a large number of physically-dispersed participants. In a conventional client-server based workflow system, these dispersed participants interact with a centralised data repository and a centralised workflow engine with assistance of client applications, requesting data and commands, respectively. Almost all the workflow functions thus are carried out on the server side in a centralised manner. Unfortunately, such an approach has encountered many problems as follows [3]–[5], [12], [23], [26]:

- The client-server architecture provides centralised workflow coordination while the computing potential at the client side is barely used. Workflow systems based on such an architecture are heavy-weight. In application domains where many workflow instances need to be executed in parallel, the centralised server (workflow server) may be overloaded with heavy computation and intense communication when the system load increases, thereby becoming a potential bottleneck. Thus, system performance can be degraded seriously in such domains.
- Systems built on top of the client-server architecture are normally vulnerable to server failures. The centralised server is normally viewed as a single point of failure in the system. The malfunction of the server may bring the whole system down. Again, this deficiency is more evident in application domains where the workflow server is required to manage many workflow instances. Although the primary-secondary server approach, for example, may improve reliability, it always requires sophisticated implementation.
- Limited scalability of the client-server architecture prevents WfMSs based on it from coping with the ever-changing workflow environment. It also raises difficulties in system configuration, as any change to the system, e.g., joining of new participants, requires modifying and updating the centralised workflow server, which is very inconvenient and inefficient. Thus, such WfMSs are especially inapplicable in application domains where workflow participants are required to join and leave frequently.
- From a user support point of view, workflow participants are largely restricted, rather than well supported. An essential and critical element of any workflow system is empowering participants to maintain autonomy and control. However, workflow participants in a client-server based workflow system are solely controlled by centralised services. A serious problem that occurs is that a vast number of participants who work on the "thin client side" may not be able to demonstrate their control, decision-making and problem-solving abilities.
- The client-server architecture is not suitable for service-oriented workflow [23] which involves service integration and service composition for flexible and short-running workflows. This is because the client-server architecture is too closed to facilitate external (Web) services available on the Internet. Thus, it is better to have an open model which allows external services to be used.

Based on the above discussion, the authors argue that a centralised workflow system is not ideal for supporting distributed workflows, especially in some application domains. The distributed nature of workflows needs to be reflected better by introducing a decentralised workflow

framework. To design a decentralised workflow system, the centralised server is expected to be ruled out while the services provided by the centralised database and workflow engine should be retained. To achieve this, a decentralised workflow system should:

- adopt a loosely-coupled structure, which does not imply the presence of a centralised data repository for data storage, or a centralised workflow engine for coordination;
- store data which are traditionally stored in a centralised database in a decentralised way so that they can be accessed by relevant sites in the system;
- migrate services which are traditionally performed by a workflow engine to other machines in the system;
- locate the service providers in a way that the traffic (requests and responses) could be guided to appropriate machines automatically;
- enable a service seeker to communicate with a service provider directly, and vice versa; and
- accommodate service-oriented applications which is the trend for application systems including WfMSs.

There are fundamental differences between a client-server based workflow system and a decentralised workflow system in terms of the system architecture and communication model. To meet the above requirements, a workflow system needs to be restructured and redesigned carefully. The corresponding approaches for supporting both build-time and run-time functions need to be re-examined.

III. System Design

As analysed in Section II, on one hand, the client-server architecture has placed some limitations on reflecting workflow's distributed nature in some application domains. Thus, the decentralised workflow system deserves careful investigation. On the other hand, there is a revolution underway that represents a different computing model. This revolution is being sparked by the technology known as *peer-to-peer (p2p) computing*, which is an extension and complement to the traditional client-server model and is more suitable to facilitate contemporary service-oriented environments. p2p can be simply defined as the sharing of computer resources and services by direct exchange. Technically, p2p computing eliminates the risk of single–source bottleneck and failure, enables better scalability and provides load balance in large organisations. Besides these, much of the wide appeal of p2p is due to social factors [2]. Since every peer in a p2p system is autonomous, ordinary participants can always play more active roles and enjoy the ability to bypass centralised control. In addition, p2p computing technology provides a collaborative, open framework for service-oriented applications such as Web services. It is observed that p2p is driving a major shift in the area of genuinely distributed computing. Success stories such as Gnutella (http://gnutella.wego.com/), Freenet (http://freenet.sourceforge.com/) and ICQ (http://www.icq.com/) have shown the practical applicability and public demand for such systems.

Very recently, p2p-based workflow systems have been recognised as one of the most strategic future directions for workflow research [19]. Combining concepts from workflow and p2p, the authors have designed *Swin*burne *De*centralised *W*orkflow (*SwinDeW*) as a special p2p system, which provides workflow management support in a truly decentralised way. SwinDeW adopts a flat, flexible and loosely-coupled structure with an intentional absence of both a centralised data repository for data storage and a centralised control engine for coordination [26]. Fig. 1 illustrates SwinDeW at a very high level. In brief, the system is defined as four layers. The top layer is the application layer, which defines application-related functions to fulfil workflows. *W*ork*f*low *P*articipant *S*oftware (WfPS) is an application which provides interfaces to interact with a workflow participant and other workflow participant software, requesting services and responding to requests. Core services of the workflow system are provided at the service layer, which include the peer management service, the process definition service, the process enactment

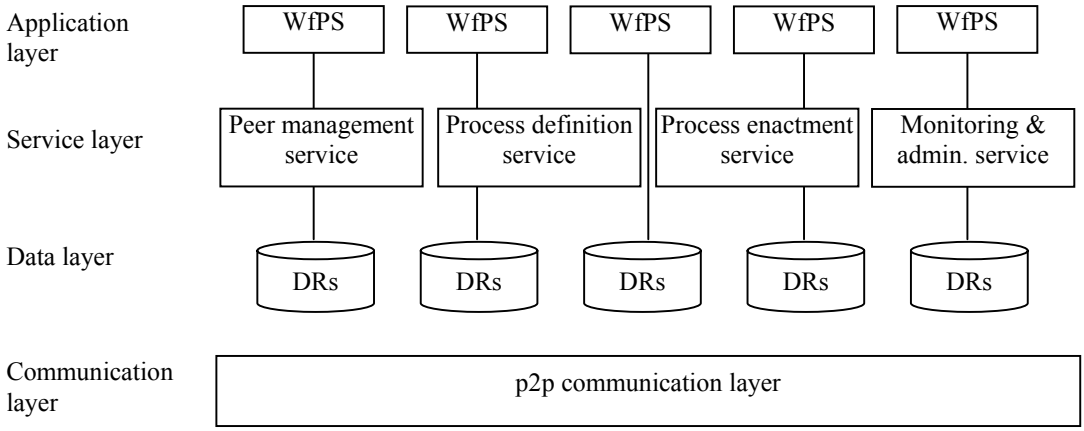

Fig. 1. Decentralised system architecture of SwinDeW

service, and the monitoring & administration service. The peer management service configures and manages peers in the system, as discussed later in this section. The process definition service provides support for workflow build-time functions, as detailed in Section IV.A. The process enactment service provides services for workflow run-time functions, as detailed in Sections IV.B and IV.C. The data layer consists of distributed data repositories (DRs) which store workflow-related information. Finally, the monitoring & administration service provides supervisory capabilities and status monitoring, which are beyond the scope of this paper and will be addressed elsewhere. At the bottom, the communication layer provides support like data transfer and routing to facilitate the direct communication amongst workflow participants. Since this paper focuses on the problems of workflow instead of p2p, the details of communication protocols and routing mechanisms are beyond the scope of this paper.

A WfPS and a set of DRs in Fig. 1 form the basic working entity in SwinDeW, which is known as a *peer*. Each peer resides on a physical machine, enabling direct communication with other peers through the communication layer in Fig. 1, in order to carry out the workflow. In most cases, a peer is a self-managing entity which is associated with and operates on behalf of a workflow participant. Normally, a peer is capable of performing some functions independently. For example, a peer could facilitate the associated participant to carry out an individual task independently. At the same time, peers are able to offer certain abilities to other peers and collaborate with one another to provide the core services defined at the service layer in Fig. 1. Again, this collaboration is achieved through direct communication between peers. For instance, a peer could pass documents, artefacts, and control information to another peer directly in a way that is governed by rules, in order to provide the process enactment service. In this regard, a peer is very similar to a service.

From the functional perspective, the WfPS of a peer consists of three software components—*user component*, *task component* and *flow component*. At the same time, the set of DRs of a peer includes four data repositories—*peer repository*, *resource and tool repository*, *task repository*, and *process repository*. These software components and data repositories represent a number of workflow features and facets in a precise and comprehensive way:

- Workflow participant software components:
  - A user component of a peer is a "bridge" between the associated workflow participant and the workflow environment. On one hand, a user component provides procedures and operations to the associated participant by delivering essential information of the workflow. On the other hand, a user component represents the roles of the associated participant in the system in terms of capabilities.
  - A task component of a peer is in charge of the execution of tasks conducted by the associated participant. In detail, a task component determines the time when a particular task starts and monitors task execution, administrates resources for the tasks, invokes tools when necessary, and terminates the thread after the work is done.
  - A flow component of a peer helps to fit an individual task into the workflow. The main purpose of a flow component is to deal with data dependency and control dependency among tasks by handling incoming and outgoing messages. In this way, the workflow execution can be coordinated step-by-step as pre-defined. A flow component manages a process repository which stores the process definition distributed to this peer.
- Data repositories:

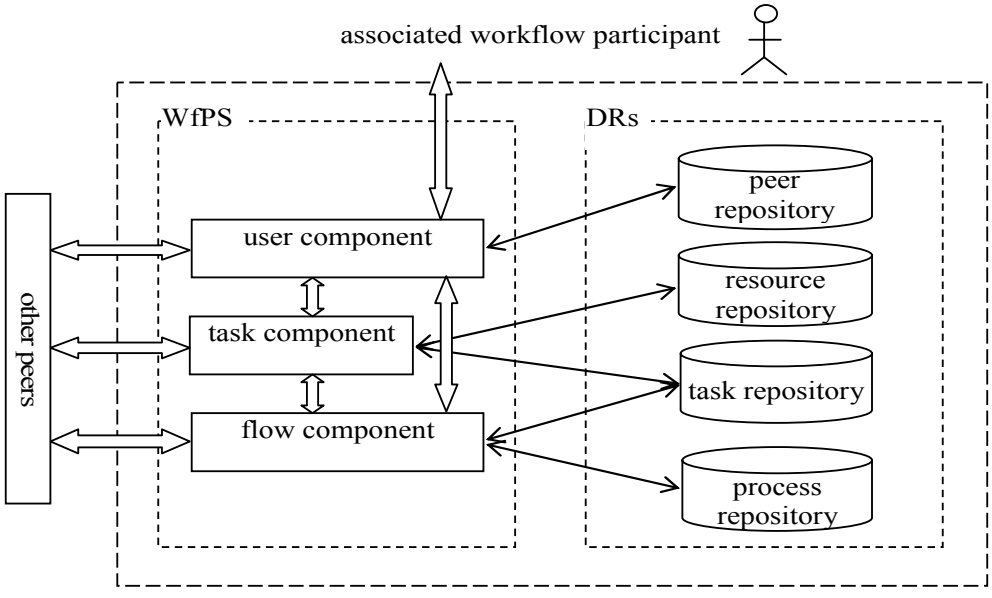

Fig. 2. Structure of a peer in SwinDeW

- A peer repository stores an organisational model which represents organisational entities and their relationships. This repository represents a user's view of the completely-defined organisational model. In SwinDeW, a peer repository is managed by a user component and realised in a directory.
- A resource and tool repository stores part of the resource model, which represents non-human resources such as machines, external hardware, tools, and so on. This repository is managed by the task component, as a task instance may require resources to support process execution.
- A task repository stores a set of active task instances, which represent the work allocated to the associated workflow participant in the context of process instances. The execution of these task instances is co-governed by the task component and flow component.
- A process repository, which stores a partial process definition distributed to this peer, is managed by the flow component. The process definition stored in a peer's process repository is actually a definition of a set of tasks, which provides the templates for the work to be done. The mechanism for process definition distribution is addressed in the next section.

Fig. 2 depicts the internal structure of a peer in SwinDeW and the interactions among the workflow participant software components and data repositories as described above. Compared with the client-server architecture, it is clear that a peer in SwinDeW plays the role as a client as well as a server. A peer plays part of the role as a server in two areas. First, a peer helps to manage part of data such as organisational information and process definition. Second, the three components interact with one another internally and externally to provide run-time coordination support for execution of workflow instances.

The other issue that needs to be addressed is how these peers are organised so that a peer can locate other cooperative peers when necessary. SwinDeW utilises a capability-based addressing mechanism by introducing a term called *virtual community* to group peers logically. A capability in SwinDeW is an object encapsulating rules with a role in workflow processes, which include the responsibility of this role, usage scenarios of this role, application-related constraints of each scenario (input, allowable operations, output, etc.), and so on. Capabilities are normally constructed by management manually based on domain knowledge. Before various peers are started up, their capabilities are normally configured by management in accordance with peers' role profiles. A capability's description is internal only to peers that have this capability for purposes like training. However, for system operation, each capability has a unique name which is referred to in process representation and in peer coordination. Based on the concept of capability, a virtual community which is characterised by a certain capability, denotes a cluster of peers associated with participants who demonstrate this capability. Virtual communities are also created, sometimes manually, at the system initialisation stage when peers are started up. Later on, virtual communities are maintained automatically in a dynamic environment. New peers search for and join corresponding communities automatically when they join the system, as detailed in Section IV.A. Given this logical gathering of peers, peers form an overlay network layered on top of the existing p2p network infrastructure.

Peers with the same capability are gathered because the associated participants normally have similar interests. Naturally, peers in the same community know about each other. If a participant has more than one capability, the associated peer is involved in multiple virtual communities. In this case, these multiple communities are interconnected through the shared member. In addition, some external means such as organisational management and organisational configuration can be used to ensure that no community in the system is isolated although this situation is normally rare. In summary, any two peers in the same community are able to locate each other. Two peers in two different communities can locate each other if there is a peer involved in both communities. Even if there is an absence of a particular peer in two communities, two peers in two different communities can still find a path towards each other via a set of third party peers. Therefore, an automatic peer discovery service can be designed, which is a key service of the peer management service and is used to locate collaborative peers whenever a peer needs to communicate with others. Generally speaking, the peer discovery service is based on capability-based addressing. In detail, each discovery request carries a capability attribute and requests the location of at least one available peer with this capability. First, a peer will try to resolve a discovery request locally by querying its own peer repository. If the request can be satisfied, that is, the requested peer is a known peer stored in the local peer repository, it is done. Otherwise, the request is routed to the peers in its peer repository, known as neighbour peers, for resolution. In turn, neighbour peers may further route the request to their own neighbour peers if they are unable to resolve the request. This procedure is repeated until the destination information is obtained and returned to the requester. Using this service, a peer in the system is always capable of locating other peers with certain capabilities. Various types of messages can be transferred directly between peers thereafter.

IV. System Functions

Regarding system functions in SwinDeW, both build-time and run-time functions are supported in a decentralised manner. Especially, aspects of process definition, process instantiation and instance execution are emphasised in this section. The mechanisms to carry out these functions in a decentralised manner are presented.

A. Process Definition

The process definition service in Fig. 1 regards workflow representation and storage in SwinDeW. A workflow definition, which consists of a network of tasks and their relationships, can be represented using different definition languages and stored in different data format. Workflow processes in SwinDeW are defined by a *definition peer* which is associated with an authorised participant such as a process engineer. It is not the purpose of the authors to invent yet another new process modelling approach. However, the storage of a process needs to be addressed due to its uniqueness. Since there is an intentional absence of a centralised data repository in SwinDeW, as mentioned in Section III, process definition data should be stored in the process repositories of various peers instead. It is obvious that replicating the complete workflow definition to each peer is resource-consuming and error-prone for consistency maintenance. More importantly, an individual peer normally needs to be concerned with only part of the process. With respect to this philosophy, SwinDeW presents a "*know what you should know*" policy [27], which allows a peer to obtain only essential knowledge of a process. According to this policy, a workflow is partitioned into individual tasks after it is modelled completely, and definition of individual tasks is then distributed to appropriate peers for storage.

A task partition, which is a notation of a task extracted from a workflow, consists of the information about an individual task in the context of a process. Formally, a task partition is represented by a six-tuple task notation *T (Process-id; Task-id; C_{pre}; C_{post}; Capability; Specifics)*:

- *Process-id:* process identifier, which identifies a workflow uniquely.
- *Task-id:* task identifier, which identifies a task uniquely in the context of a workflow.
- C_{pre}*:* precondition *(taskset1, logic).* The relationships between the task and its predecessors where
  - *taskset1: (((taskid1, capability1, input-data1)* | C_{pre}, | *null), ((taskid2, capability2, input-data2)* | C_{pre}, | *null)).* Taskset1 represents a set of preceding tasks with corresponding input data.
  - *logic: (straight, and, or).* By combining either an incoming or an outgoing flow with logic, a variety of types of logic operators are produced. The operators produced are *straight in, straight out*, *and out, or out, and in, or in,* which represent various routing structures like sequence, branching and convergence.
- C_{post}: post condition (*taskset2, logic*). The relationships between the task and its successors where
  - *taskset2:* (((*taskid1, capability1, output-data1*) | C_{post}, | *null)*, ((*taskid2, capabiltity2, output-data2*) | C_{post}, | *null*)). Taskset2 represents a set of successive tasks with corresponding output data.
- *Capability:* The participant capability that is necessary to fulfil the task.
- *Specifics:* the specifics about the task, which may include other information about how to execute this task such as resources and tools.

Regardless of the definition language and database actually used, a workflow definition can always be converted into a set of task partitions, each of which is represented by this notation. After that, the individual tasks are distributed to appropriate peers one-by-one, according to a capability matching. That is, a task requiring a capability to support its execution will be distributed to all the peers in a community characterised by this capability. These peers need to know the detail of this task because they are associated with participants who are capable of carrying out the instances of this task later. The procedure of task distribution may consist of three steps. First, the definition peer invokes the peer discovery service automatically to discover the locations of a peer with the required capability. Second, the definition peer transfers the full definition of the task to the discovered peer through a point-to-point connection. Finally, the discovered peer propagates the task definition within the community in order that all the peers with required capability insert a new record for this task definition into their process repositories. Using this mechanism, a workflow definition is finally stored in a decentralised fashion.

The above discussion is based on an assumption that the workflow management system is static where the peers remain unchanged. However, a workflow management system is always considered dynamic if it takes account of workflow participants' behaviour. A workflow participant may join and leave the system sometimes, or a workflow participant may be trained or promoted to obtain extra capabilities. SwinDeW's open framework eases the support for participants' dynamic behaviour and offers additional scalability. As discussed in Section III, the information about participating peers is stored in the distributed peer repositories. Apparently, the alteration of the participating peers only occurs locally and influences the small scope of the system. Adjusting peer repositories of relevant peers locally can reflect the alteration of the participating peers. Therefore, this perspective of system scalability of SwinDeW is enhanced. In particular, various actions are taken to support a peer's dynamism when the following three situations occur:

1) a workflow participant joins the system,
2) a workflow participant leaves the system, and
3) a workflow participant's capabilities are changed.

When a new workflow participant joins the system, the associated peer actually joins the corresponding virtual communities according to the user's capabilities. A join request can be taken by any existing peer, which will invoke the peer discovery service to find existing peers in the corresponding communities. The existing community members then update their peer repositories and pass the related process definition data to the new peer under the request. Thus, the new peer gains all essential information about the potential work to be done and is capable of taking part in the workflow system immediately. An existing workflow participant may leave the system either explicitly or

implicitly. In the former case, the associated peer informs the system of its departure by sending a special message to all the peers in its peer repository. Correspondingly, the relevant peers update their peer repositories when receiving this message. In the latter case, the associated peer leaves the system quietly, for example, a peer leaves the system accidentally when some exception occurs. To keep the system informed of this situation, another special message, say peer availability status, is propagated periodically indicating that the peer issued this message is active. If a peer has not been heard from for a period of time, it is considered an inactive peer and will be removed from the system automatically. On the last occasion, i.e., the capabilities of a peer are changed, the peer may join and leave some specific communities according to the capability alteration. At that time, this peer may request some new data and delete some irrelevant data accordingly.

B. Process Instantiation

The process enactment service in Fig. 1 provides support for run-time functions of SwinDeW, which consists of two logical steps, process instantiation and instance execution. These two steps need to be carried out in sequence in order to create and execute workflow instances, respectively. A workflow instance, which consists of a network of task instances, represents one individual enactment of the workflow defined at build-time. Similarly, each task instance in a process instance represents a single invocation of a task.

The procedure of process instantiation creates various task instances and also assigns task instances to workflow participants for execution. In a client-server based workflow management system, various task instances required are created on the server side and presented to the participants via a work list. However, in a decentralised workflow environment, a process instance cannot and should not be created at a single site. The mechanism discussed in this paper allows various peers to coordinate with one another in order to create relevant task instances at different sites. In this way, task instances are created one-by-one from the starting task to the termination task [28]. The procedure of process instantiation can be summarised as follows:

1) A starting task instance is created by a peer under management, sometimes manually, or as a response to a coming event, for example, receiving an application in an application processing workflow system. This peer is known as the *current instantiation peer*.
2) The current instantiation peer looks for other peers to instantiate the direct succeeding tasks automatically with the mechanism addressed later in this section. The instantiation of a task instance is considered complete only after either all of its direct succeeding task instances are created or it has no succeeding tasks at all. The information about the succeeding tasks is stored in the peer's process repository.
3) If the succeeding tasks have their own succeeding tasks, the selected peers act as the current instantiation peer and repeat step 2), one-by-one.
4) If there is no current instantiation peer, process instantiation is completed.

From the above description, it is clear that the procedure of process instantiation in SwinDeW is the procedure of work allocation as well. The work represented by a task instance is assigned to the participant associated with the peer creating this task instance. The distinction of this mechanism is to allow various peers to create required task instances from the starting task to the termination task. The outcome of the process instantiation is a network of relevant peers on behalf of relevant participants performing various tasks. Again, the instance data are stored in a decentralised manner. After its creation, a task instance is inserted into the corresponding task repository, waiting for scheduling.

The key step in this process instantiation is to create task instances in order. There are three types of structures to control the order in which the tasks are executed, i.e., *sequential*, *branching* and *convergent* structures. Correspondingly, there are different procedures for peers to instantiate tasks as follows.

The sequential structure denotes the procedure where tasks are executed in order. In a sequential relationship, a task only has one direct succeeding task. Assume that an instance of task T_i with definition $T\ (P_i;\ T_i;\ C_{pre\text{-}i};\ ((T_j,\ Capability_j,\ output_j),\ straight);\ Capability_i;\ Specifics_i)$ is created by peer N_i. N_i then activates the peer discovery service to search for a peer that has $Capability_j$. After N_k is returned to N_i as the result of the peer discovery service, N_i sends an *instantiation request* to N_k. Subsequently, N_k broadcasts the request in the corresponding virtual community to let other peers know about this work. Capable peers, i.e., those peers that have the required capability and are available for this task instance, negotiate automatically to decide who will carry out this task instance eventually, as discussed later in this section. Finally, N_j, the peer selected to accept T_j, creates an instance of T_j, sends a response to N_i, and restarts the instantiation process to find successors of itself. All the communication in this procedure is through direct message exchange between peers. Each message is structured in XML format and can be interpreted properly by the WfPS of the peer which receives it.

The branching structure denotes the procedure of splitting a task thread into multiple task threads, which are executed in parallel, or the procedure of making a decision about which branches to take when encountering multiple choices. In this case, task $T_i\ (P_i;\ T_i;\ C_{pre\text{-}i};\ C_{post\text{-}i};\ Capability_i;\ Specifics_i)$ has more than one succeeding task. During the instantiation stage, each succeeding task should be instantiated. If peer N_i has created an instance of T_i, it instantiates each task in the $C_{post\text{-}i}$ of T_i with the mechanism described above for sequential structure.

The convergent structure denotes the procedure of converging some parallel task threads into a point and triggering the execution of a single task thereafter, either synchronously or asynchronously. In this case, task $T_i\ (P_i;\ T_i;$

$C_{pre\text{-}i}$; $C_{post\text{-}i}$; $Capability_i$; $specifics_i$) has more than one direct preceding task. Each of T_i's direct preceding tasks requests the instantiation of T_i independently. To deal with this situation, the peers in the virtual community postpone their negotiation to the time when all the requests from the preceding peers are received. Then all the requests are considered and balanced to reach an agreement upon which peer accepts the requests to create an instance of T_i. Again, the selected peer restarts the instantiation procedure to continue the instantiation of the next task(s).

Note that work allocation in this approach is negotiation-based, especially when more than one available peer can accept the task instance. It is up to the automatic negotiation among relevant peers to determine the allocation of the task instance. The main goals of this dynamic allocation are to balance workload and optimise system performance. SwinDeW simply uses workload as a measurement of system performance and always assigns a task instance to a peer with the least load at that time. It is believed that balancing the workload may achieve better performance [28].

In general, the first peer receiving the task instantiation request, which is discovered by the discovery service, is responsible for the negotiation process. All the other capable peers calculate their workload independently and advise the first peer upon receipt of the instantiation request. Then the first peer picks out the right peer based on the allocation policy which will be described next in this section and asks the selected peer to confirm this allocation. Since the first peer that has been discovered may be different for different instantiation requests, all the peers in the community are able to exhibit the same functionality in managing the negotiation process.

To optimise the system locally, all the capable peers are identified first. Then the workload of each available peer i in a time period such as one day is calculated using formula $w_i = \sum t_k$, where t_k is the workload of task instance k that has been assigned to peer i in this time period. The task instance is eventually assigned to the peer with minimum w. The objective of this algorithm is to bring the workload into proportion among the available peers on the basis of the current condition. The coexistence of idle peers with overwrought peers could be avoided on a local scale.

However, it often happens that some key tasks require some capabilities of high-level skills that only belong to a small number of participants such as managers. These more capable people usually can perform some lower-level skilled tasks as well. If the selection policy is based on individual workload balance only, these people may be busy with performing low-level skilled tasks at a time. In this case, when a key task arises, it could be the case that no peer associated with these participants can accept it because all of them are engaged. Thus, the whole process instance is blocked and the global performance is degraded.

To optimise system performance globally, the performance bottleneck of the whole system should be identified and relieved. Given a workflow system with n virtual communities, the task instances are assigned to different communities according to the capability attributes, and are taken by various members involved in the communities. Thus, the community with the heaviest workload determines the overall system performance. To a particular community i with m_i members, the mean workload of this community in the current time period is $\overline{w}_i = \sum_{k=1}^{m_i} w_k \Big/ m_i$, where w_k is the workload of member k in the community. Therefore, the community with the heaviest workload becomes the bottleneck of the system performance, i.e., in a workflow system with n communities, the global performance is determined by $max(\overline{w}_i, i \in (1,n))$.

Based on the above analysis, the following algorithm is designed which considers the philosophy that for a peer involved in more than one community, the assignment of a task adds workload to all the communities in which the peer is involved.

for each available peer j

$$w_j = max(\frac{\overline{w}_i \times m_i + w}{m_i}),\ i \in \{community\ c \mid j\ is\ involved\ in\ c\}$$

assign the task instance to k with minimum w_k

where m_i is the number of peers in community i, $\overline{w}_i$ is the current mean workload of community i. Obviously, every time when a new task instance needs to be created, this algorithm seeks a peer to accept the instance, which pursues the lowest $max(\overline{w}_i, i \in (1,n))$, i.e., the optimised performance of the whole system.

From the system performance point of view, this instantiation process may achieve better system performance as work is assigned dynamically to balance the load. From the user support point of view, the work allocation can take human satisfaction into account. The vast number of ordinary participants are allowed to negotiate automatically with the assistance of peers, which in turn enables the participants to play more active roles. The formulas used in this approach can be easily extended. Participants' working habits, styles and preferences can be regarded as factors influencing the decision-making of work allocation. Extra traffic due to decentralised process instantiation and negotiation-based work allocation is inevitable and worthwhile in order to achieve load balancing and user satisfaction. However, compared with client-server based workflow systems, it is very unlikely that this extra traffic will cause network overload. This claim is justifiable because of the following reasons. First, messages like instantiation request and confirmation are also needed in client-server based workflow systems where the server advises the clients with the work allocation messages and the clients confirm the receipt of these messages. Second, traffic for negotiation is light because peers only exchange a small

amount of information like workload. Therefore, issues related to possible network overload can be ignored.

C. Instance Execution

As discussed in Section IV.B, once a process instance is created, a peer network is also constructed for carrying out this process instance. Such a process instance is executed under the management of the workflow system. Various task instances are scheduled to enact at different sites, step-by-step. In general, the execution of a task in workflow depends on the satisfaction of two conditions: the *information condition* and the *control condition*. The information condition of a task defines the start condition of this task from the data dependency perspective. In most cases, a workflow task requires some input data, which are normally the output data of its preceding tasks, and generates some output data, which are transferred to its succeeding tasks as their input data. A task can be executed only after essential input data are available. Correspondingly, the control condition of a task indicates the start condition of a task from the control dependency perspective. In this case, a task can be executed only after some relevant work has been completed logically. Hence, to schedule the execution of various task instances in a proper order without the assistance of a centralised workflow engine, relevant peers performing various task instances should collaborate with one another [28].

Again, this collaboration is realised through direct message exchange between peers. There are two kinds of messages flowing between peers, i.e., *information messages* and *control messages*, which are structured in XML format. The former messages transfer data related to the process instance to coordinate application data dependency between tasks, according to the rules defined in the workflow definition. An information message normally transfers application data to match the output-parameter of a task instance with the input-parameter of another task instance. Upon receipt of an information message, a peer evaluates the information condition of its task instance. The latter messages, which are emphasised here, deliver workflow control data to coordinate control dependency between tasks, again, according to the rules in workflow definition. A control message is a notification from one peer to another, facilitating the recipient to perform functional operations. In short, SwinDeW supports the following four types of control messages:

- *Completion of a task instance*: A peer that has just completed a task instance notifies its successor peers that the work has been completed successfully.
- *Cancellation of a task instance*: A peer that is in charge of a task instance notifies its successor peers that the task instance has been cancelled.
- *Reallocation of a task instance*: A peers notifies other relevant peers that a task instance is reallocated to a different peer.
- *Detection of exception*: A peer notifies relevant peers of a detected exception.

Upon receipt of a control message, a peer may respond accordingly to manage its own task instance. When a peer receives a completion notification, it evaluates the control condition of its task instance. Execution of a task instance can start when both the information and the control conditions are satisfied. When a peer receives a cancellation notification, it evaluates the start condition of its task instance and determines whether the instance needs to be cancelled. In addition, when a peer receives a reallocation notification which normally occurs in exceptional situations, it updates its successor or predecessor peers, and is ready to interact with new peers. Finally, the handling of exception detection messages is beyond the scope of this paper.

In summary, process instance execution is coordinated by direct communication among the peers performing relevant task instances, which is light-weight and occurs at a relatively low cost. A peer receives messages from its predecessor peers directly, evaluates the information and control conditions of the task instance independently, starts working when both the conditions are satisfied, and notifies its successor peers directly by delivering information messages and control messages after the task instance is completed. The successor peers repeat the same procedure until the completion of the whole process instance.

V. Prototype Implementation

To demonstrate the key ideas discussed in this paper, a prototype based on Sun MicroSystem's JXTA (http://www.jxta.org) has been implemented. Project JXTA is an active and progressive project dealing with current p2p problems and providing an implementation in the JXTA protocol. The project does not bind itself to one company or one programming language for interoperability purposes. Since 2001, JXTA has been a popular open source p2p framework. For these reasons SwinDeW easily places itself on top of the JXTA framework.

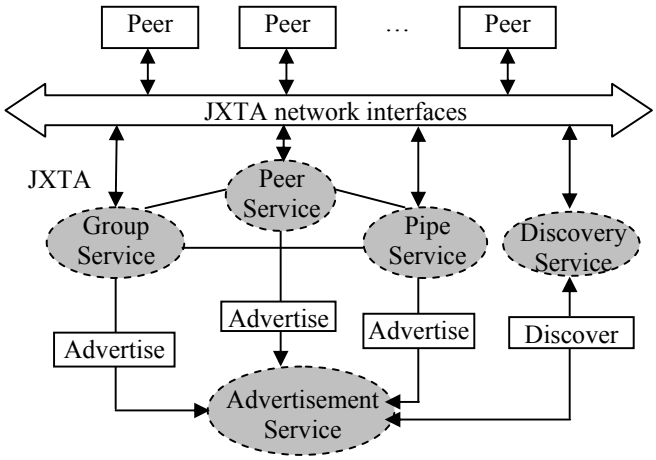

Fig. 3. Framework of SwinDeW prototype

The current SwinDeW prototype is written in the Java[tm] programming language utilising J2SE version 1.4 API. As shown in Fig. 3, the prototype takes advantage of the JXTA

key components like the advertisement service, the group service, the peer service, the pipe service, and the discovery service. The advertisement service is used to publish contents such as peers, peer groups and pipes in the JXTA virtual network. The group service of JXTA is exploited for virtual community management. When a new peer is started up, it looks up the virtual communities labelled by the capabilities it represents through the advertisement service. If such communities exist, the new peer joins these communities. Otherwise the new peer creates one or more new virtual communities accordingly and publishes through the advertisement service. The peer service is central to managing the peers in the JXTA virtual network. Each peer publishes a network interface through which direct point-to-point connections can be established between two peers. The pipe service deals with physical message transfer between two peers in both point-to-point and propagation modes. The discovery service of JXTA is used to search for advertisements in the JXTA virtual network. All the communication relies on the JXTA messaging protocol and the message that is traded between peers uses XML format. Thus, the core services of SwinDeW are realised through the invocation of the JXTA core services. This invocation is encapsulated in a set of self-developed JXTA network interfaces, which make the JXTA implementation transparent to application development.

This prototype consists of a graphic process modelling tool. For interoperability purposes, the workflow definition language used in SwinDeW is the XML Process Definition Language (XPDL - http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf), which is a publicly-accepted standard released by WfMC. The software components within each peer described in Fig. 2 are implemented. These components invoke JXTA services to interact with the components of other peers so that the SwinDeW services depicted in Fig. 1 are offered.

VI. Case Study

In a typical university, the student registration service deals with the registration concerns of students, which include registration in courses, change of timetable, withdrawal of courses, etc. This service is normally well-defined and can be viewed as a flow of tasks that accomplish an objective. Thus, workflow solutions are well-suited in this scenario.

Two characteristics of the student registration service which need to be addressed properly are illustrated in this case study. First, the student registration service is fairly distributed. To provide the requested services, various staff from different units of the university may be involved. For example, course advisors approve registration requests, technical staff manage student computer accounts, treasurers handle payments, and enrolment officers carry out paper work. These staff are distributed in terms of physical location and administration. Second, the student registration service may experience a heavy load from time to time. Thousands of students, new or current, may lodge their registration requests just before the deadline. Therefore, the performance of the registration processing system clearly is a major concern. The system should be capable of handling a large amount of requests in a short period of time. With respect to these two characteristics, SwinDeW is evidently an applicable system for this type of scenarios, as decentralisation and performance are two of the many advantages of SwinDeW. Fig. 4 illustrates a possible workflow definition for the student registration process, which consists of a set of tasks in a certain order.

For the purpose of simplicity, assume that this process involves seven workflow participants which are facilitated by peers P_1, P_2, P_3, P_4, P_5, P_6 and P_7, respectively. Of these seven peers, P_1 demonstrates the capability of the enrolment officer only, P_2 demonstrates the capabilities of both the enrolment officer and the course advisor, P_3 demonstrates the capability of the course advisor only, P_4 and P_5 demonstrate the capability of the technical staff, and P_6 and P_7 demonstrate the capability of the treasurer. After an initial configuration, these

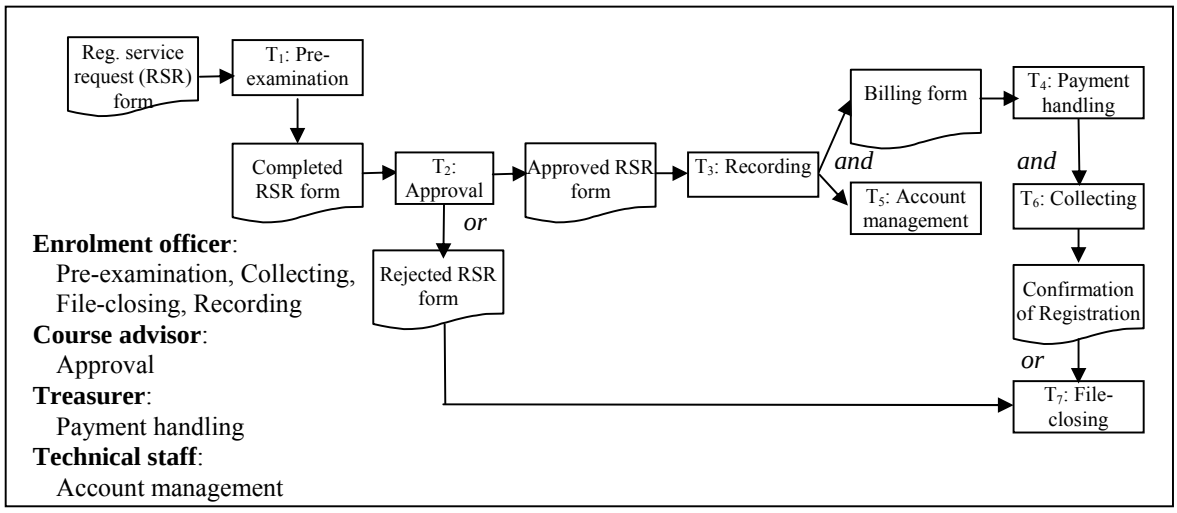

Fig. 4. Typical student registration process

peers form and join corresponding virtual communities according to the capabilities that their associated participants represent, as shown in Fig. 5. At the same time, the system configures the peer repositories of some peers to ensure all the virtual communities are somewhat interconnected, represented by the lines in bold in Fig. 5. During the operation of the system, new staff members can come and go with the support of the peer management service, as discussed in this paper. With support of such virtual communities, the peer discovery service can be fulfilled. For example, P_3 can discover P_2 and P_4, as P_2 and P_4 are in P_3's peer repository. P_3 is also able to discover P_1 and P_5, via P_2 and P_4, respectively. Subsequently, P_3 can discover P_6 via P_4 and P_5, and P_7 via P_4, P_5 and P_6.

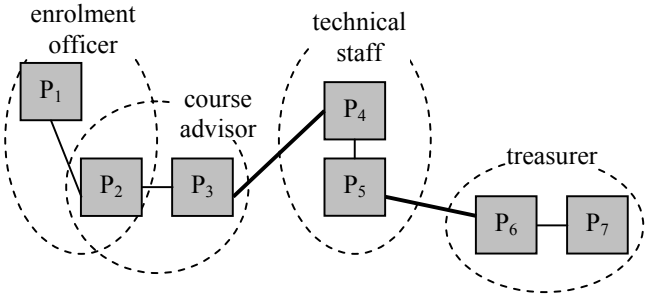

Fig 5. Virtual communities of peers

During build-time, this student registration process is modelled and specified. Then the process definition is divided into task partitions with the six-tuple task notation presented in this paper. These task partitions are distributed to various peers accordingly for storage. For example, task T_2 in Fig. 4 which is represented as *T (P-ID; T_2-ID; (T1-ID, enrolment officer, completed RSR form), straight); ((T_3-ID, enrolment officer, approved RSR form), (T_7-ID, enrolment officer, rejected RSR form), or); course advisor; specifics)* will be distributed to the peers associated with the course advisors, i.e., P_2 and P_3, so that these participants can carry out the future instances of this task within the context of the process.

Once a new registration request is received, a new process instance needs to be created to handle this request. The procedure of process instantiation creates task instances one-by-one, from task T_1 for *Pre-examination* to task T_7 for *File-closing*. First, an instantiation request, which may be issued manually by management or automatically by an external system, is dispatched to the community of enrolment officers, asking for the instantiation of the starting task. The peers associated with the two enrolment officers, i.e., P_1 and P_2, may negotiate with each other about who serves this request. The negotiation can be based on workload balancing so that the peer with less load, say, P_1, creates an instance of T_1 for *Pre-examination* to check the validity and correctness of the coming registration request. P_1 then sends a request to the community of course advisors, asking for the instantiation of the task for *Approval*. Again, P_2 and P_3 may negotiate with each other to determine who accepts this request to create an instance of T_2. After that, the selected peer, say P_3, starts looking for peers to instantiate succeeding tasks of T_2. This procedure repeats until none of the created task instances has succeeding task(s). Eventually, the output of the process instantiation, i.e., a peer network connecting peers that create various task instances, is generated to serve this registration request. Later on, these peers will perform corresponding task instances in the right order with system support for instance execution. In addition, when another registration request comes, a second peer network will be constructed to handle this case. Two peer networks for two cases may be different as the result of dynamic work allocation. For example, the task instance for *Pre-examination* in the second process instance may be created by peer P_2.

In SwinDeW, there is an absence of any centralised coordination for the enactment of process instances. The peers engaged in the created process instance, i.e., the peers that instantiate various tasks at the process instantiation stage act independently to perform task instances. Moreover, the peers performing various task instances in the created process instance communicate with one another directly by transmitting information and control messages in order to route work in the proper sequence. In enacting the process instance created above, first, P_1 can start its work straight away because the start condition of the task instance of T_1 is satisfied upon the receipt of the registration request. After the completion of the instance of T_1, P_1 notifies its successor peer, i.e., P_3, with a completion message and attaches a completed RSR form to an information message. Upon the receipt of the message from P_1, P_3 is able to evaluate the start condition of the instance of T_2 independently. Obviously, this start condition is satisfied after the receipt of both the information and control (completion) message. P_3 can start its work then. If the registration request is approved in this instance, a completion notification is sent to the peer performing the instance of T_3 to continue the execution along this branch. At the same time, a cancellation notification is sent to the peer performing the instance of T_7 to skip the execution along this branch. Similarly, the subsequent task instances can be executed under the control of the decentralised instance execution support. As a result, the whole process instance is executed properly as defined in the SwinDeW decentralised environment.

Some benefits of SwinDeW can be reflected with this case study. First, the computing and storage capacities of the computers involved in this workflow system can be shared to enhance the system performance. Direct interaction between peers who perform adjacent tasks would reduce communication delay and thus may achieve better performance. Second, system robustness is likely to be enhanced because unavailability of a single peer would normally not bring the whole system down. For example, when a peer associated with an enrolment officer becomes unavailable, the work assigned to this peer can be quickly reassigned to another enrolment officer for execution. Thus, execution of process instances can proceed. Third, the system is much more scalable as new staff members can join the system easily to offer more processing capacity. Fourth, use of

SwinDeW may satisfy staff members better as staff members are more autonomous and enjoy the ability to bypass centralised control. Finally, SwinDeW offers an open framework so that it is possible to incorporate some external services into this student registration service. For example, the task for *Pre-examination* could be carried out by some external education agents who provide services over the Internet.

Note that statistical results are not collected in this case study to support the claimed benefit in terms of performance directly. The research prototype has not been experimented within real world student registration so that quantitative results are not available. In addition, empirical performance analysis of workflow management systems has not been addressed satisfactorily so far across the board, even for client-server based systems. This makes quantitative comparison difficult, if not impossible. Also, like Grid Computing, the claimed advantage of SwinDeW in terms of performance would normally become evident when it is used to support large-scale and/or computation-intensive applications such as coordination of a large number of workflow instances simultaneously.

VII. Related Work and Discussion

Many research efforts have been placed to address problems discussed in this paper, i.e., poor performance and reliability, limited scalability, user restrictions and unsatisfactory system openness. The importance of associating "workflow management" with "distribution" has been emphasised in a lot of literature [10], [15], [22]. At the same time, some conceptual approaches have been introduced. Most of the approaches are still based upon the client-server architecture. To name a few, the Exotica/FMQM project [3] focuses on distribution, scalability and fault-tolerance by minimising the need for centralised control structures. ADEPT [6] focuses on enterprise wide workflows and cross enterprise workflows, and supports both static and dynamic server assignments. Endeavors [16] and WorldFlow [18] use HTTP to provide a coordination mechanism for distributed process execution and tool integration on the Internet. DartFlow [8] uses transportable agents as the backbone to control the execution of process instances. The METEOR workflow management system [21] provides CORBA+Java based and Web-based enactment service for fully distributed scheduling where tasks are mapped onto task managers. XRL/flower [25] supports highly-dynamic workflow by describing processes at the instance level. The set of work items are sent to distributed participants by the work distribution module and are coordinated by a Web server and a Petri-net engine. Moreover, there also exist few approaches based upon p2p computing technology. For example, Fakas presents a conceptual p2p technology for dynamic workflow management, which is based on concepts such as a Web Workflow Peers Directory (WWPD) and Web Workflow Peer (WWP) [11]. Using this technology, peers are proposed to register with the system and offer their services and resources to other peers. PeCo [9] decentralises workflow management using collaborative technologies and concepts while providing a pluggable framework for integrating business process applications and human contributors. Matrix [20] delivers grid workflow protocols and workflow language descriptions necessary to build a p2p infrastructure for Grid Workflow Management Systems. This middleware allows applications and services based on standards such as Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP) to communicate with data and other resources in Grid environments.

On one hand, the approaches based on the client-server architecture are still limited by centralised management. Thus, they either address the problems partially, or require complicated languages or complex algorithms. In addition, the remaining centralised services like centralised process instantiation and work assignment make them relatively inflexible in some application domains. On the other hand, research on implementing workflow in a peer-to-peer environment is still at a very initial stage. The few existing approaches are mostly conceptual ideas without concrete system analysis and design. A number of issues such as decentralised data storage are not addressed by these approaches. However, the ideas of combining workflow with p2p are valuable and are taken into consideration when SwinDeW was designed.

Since 2001, the authors have carried out innovative and concrete research on p2p-based, decentralised workflow [26]–[29] with ideas formed much earlier. Problems caused by centralised management have been analysed comprehensively and needs for p2p-based workflow have been revealed. A system framework has been designed carefully based on which relevant mechanisms for build-time and run-time functions have been discussed broadly. The research reported in this paper is based on these preliminary work with significant extension and improvement. The ideas presented before are naturally integrated in order to provide complete workflow support. This research views the unsolved problems from a different perspective because it is believed that these problems arise due to the mismatch between system requirements and system realisation. The aim of this research is to address these issues rudimentally by incorporating workflow with the p2p technology. The approach presented in this paper is expected to provide better support for processes in some application domains where performance, reliability, scalability, user support, system openness have not been addressed satisfactorily with client-server based approaches. In summary, the advantages and effectiveness of SwinDeW in such domains are as follows:

- SwinDeW may enable better system performance because it completely distributes both data and control to highly utilise the computing and storage capabilities of the entire enterprise. The decentralised coordination provided by SwinDeW is regarded as light-weight and cost-effective.

In addition, the dynamic work assignment based on load-balancing, which is another distinguishing feature of SwinDeW, may also contribute to achieving better system performance.

- Compared with client-server based, or partly client-server based approaches, SwinDeW may involve much less risk where the whole system fails simply because some individual bottlenecks are overwhelmed by heavy computation.
- System scalability may also be enhanced as peers retain a loosely-coupled topology. Virtual communities are dynamic so that workflow participants can come and go. Changes to the system do not require modifying and updating the centralised workflow server. New peers can easily join the system at any time and through any existing peer with no need to change the settings of a particular site. The system may thus be capable of coping with the dynamic system size.
- SwinDeW may loosen restrictions to workflow participants. The novel philosophy of "know what you should know" is proposed for peers to gain more knowledge and control. With essential data, human beings are able to participate in workflow systems more actively than ever before. Ordinary users can enjoy the abilities of being involved in system management when necessary. Personal preference can also be expressed and incorporated. At the same time, direct communication amongst the peers allows the participants to coordinate in a more suitable way. Hence, this approach may offer more user control which is an important feature needed in teamwork.
- SwinDeW utilises novel techniques involving p2p execution of processes. This open model may also be able to support service-oriented workflow well because it naturally exploits the distributed nature of the Internet. In particular, the composition and execution of Web services can be facilitated properly by peers through techniques such as publish and subscribe [7], [24].

However, moving from client-server to p2p also brings some tradeoffs which can be potential limitations. Some of the tradeoffs of the proposed approach are summarised as follows, although they are outweighed by the advantages it offers:

- As a tradeoff, management and monitoring of workflow execution becomes more difficult in a p2p-based workflow system, as workflow execution is coordinated by distributed peers. In SwinDeW, special management peers are implemented which communicate with ordinary peers directly to obtain the related information (instance status, performance data, historical information, etc.).
- The ability to handle exceptions and erroneous situations may be impaired in SwinDeW. Unlike client-server based workflow systems where errors and exceptions can be detected and handled by centralised servers, SwinDeW requires more complicated mechanisms to deal with aspects of flexibility such as general exception handling and dynamic change handling, which are not addressed in this paper and regarded as future work.
- The approach presented in this paper assumes that ordinary workflow participants have the abilities to perform some basic management operations. The approach is probably not appropriate when the management operations which are very complex and require high-level skills. Workflow participants may need to be further trained in p2p concepts, role profiles, and workflow processes as part of staff development.
- Issues of authentication and security may become more important in SwinDeW because p2p-based applications enable networked access of the resources. These issues may need to be addressed for certain applications in the future.

Moreover, this paper only focuses on completely specified process support. Issues such as incompletely specified process support and WfMS interoperability in intra- or inter-organisational settings are beyond the scope of this paper and will be discussed elsewhere [30].

VIII. Conclusions and Future Work

In this paper, some unsolved problems in the workflow area, i.e., poor performance, lack of reliability, limited scalability, user restriction and unsatisfactory system openness are analysed carefully. The architectural limits of the client-server paradigm are believed to be the main causes of these problems. To reflect workflow's distributed nature better, it is essential to have a light-weight, cost-effective, decentralised workflow management system. SwinDeW, a decentralised workflow management system introduced in this paper, applies the peer-to-peer (p2p) technology to the workflow scenario, which is viewed as a future trend in workflow management. This approach provides a decentralised workflow execution environment by removing centralised servers from the system. Correspondingly, services usually provided centrally are offered in a decentralised manner. To achieve this, both data and control are distributed: the workflow representation is partitioned and distributed properly, and the process execution relies on the direct communication and coordination of individual peers. Referring back to the problems, the performance bottleneck and single points of failure are eliminated, which may result in better performance and increased resilience to failure. At the same time, system scalability is enhanced because a p2p workflow system is able to deal with dynamically-changed systems better. Workflow participants are also better supported as this system allows for more user control ability. In addition, the open, collaborative framework of SwinDeW makes it more suitable for service-oriented applications.

In the future, decentralised workflow will be explored further. The system will be amended by developing extra facilities such as organisational management facilities. Aspects of incompletely specified process support and

workflow interoperability will be reported separately. At the same time, experiments will be conducted to support different kinds of processes. Experimental data will be collected for analysis, comparison and system improvement purposes. Moreover, it is a logical next step to integrate this research with the rapidly-growing research on service management and knowledge sharing. More specifically, one of the authors' ongoing activities is to extend SwinDeW to Web services support where a workflow process will be defined in BPEL4WS (Business Process Execution Language for Web services).

ACKNOWLEDGMENT

The authors are grateful for the constructive comments of the anonymous reviewers and the prototyping work of L. Setiawan and J. Derham.

REFERENCES

[1] W. M. P. van der Aalst and K. M. van Hee, Workflow Management: Models, Methods, and Systems, MIT Press, Cambridge, MA, 2002

[2] K. Aberer and M. Hauswirth, Peer-to-Peer Information Systems: Concepts and Models, State-of-the-art, and Future Systems, Proc. of the 8th European Software Engineering Conference (ESEC) and the 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9), 326-327, Vienna, Austria, Sept. 2001

[3] G. Alonso, C. Mohan, R. Gűnthőr, D. Agrawal, A. El Abbadi and M. Kamath, Exotica/FMQM: A Persistent Message-based Architecture for Distributed Workflow Management, Proc. of IFIP Working Conference on Information Systems for Decentralised Organisations, Trondheim, Norway, Aug. 1995

[4] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan, Functionality and Limitations of Current Workflow Management Systems, Research Report, IBM Almaden Research Center, 1997, http://www.almaden.ibm.com/cs/exotica/wfmsys.pdf

[5] N. A. Assimakopoulos and A. E. Lydakis, The Use of Systemic Methodologies in Workflow Management**,** Proc. of the 47th Annual Meeting of the International Society for the Systems Sciences, Hersonissos, Crete, July 2003

[6] T. Bauer, P. Dadam, Efficient Distributed Control of Enterprise-Wide and Cross-Enterprise Workflows, Proc. of the Workshop Informatik '99: Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, 25-32, Paderborn, Germany, Oct. 1999

[7] B. Benatallah, M. Dumas, Q. Z. Sheng and A. H. Ngu, Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services, Proc. of the 18th International Conference on Data Engineering (ICDE'02), 297-308, San Jose, USA, Feb. 2002

[8] T. Cai, P. A. Gloor and S. Nog, DartFlow: A Workflow Management System on the Web Using Transportable Agents. Technical Report PCS-TR96-283, Dartmouth College, 1996, ftp://ftp.cs.dartmouth.edu/TR/TR96-283.pdf

[9] M. D. Coon, Peer-to-Peer Workflow Management White Paper, Aug 2002. (http://www.proteus-technologies.com/cmm/docs/P2P_Workflow_Whitepaper.doc)

[10] J. Eder, E. Panagos, Towards Distributed Workflow Process Management, Proc. of the Workshop on Cross-Organisational Workflow Management and Coordination, San Francisco, USA, Feb. 1999

[11] G. J. Fakas and B. Karakostas, A Peer to Peer Architecture for Dynamic Workflow Management, Information and Software Technology Journal, Elsevier, 46(6): 423-431, 2004

[12] L. Fischer, ed., Workflow Handbook 2002, Lighthouse Point, Fla.: Future Strategies, 2002

[13] D. Georgakopoulos, M. Hornick and A. Sheth, An Overview of Workflow Management: From Process Modeling to Infrastructure for Automation, Journal on Distributed and Parallel Database Systems, 3(2):119-153, Apr. 1995

[14] J. Grundy, M. Apperley, J. Hosking, and W. Mugridge, A Decentralised Architecture for Software Process Modelling and Enactment, IEEE Internet Computing, 2(5): 53-62, Sept./Oct. 1998

[15] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, M. Teschke, A Comprehensive Approach to Flexibility in Workflow Management Systems, Proc. of the International joint Conference on Work Activities Coordination and Collaboration (WACC99), 79-88, San Francisco, USA, Feb. 1999

[16] A. S. Hitomi, P. J. Kammer, G. A. Bolcer and R. N. Taylor, Distributed Workflow Using HTTP: Example Using Software Pre-requirements, Proc. of the 20th International Conference on Software Engineering, Apr., 1998

[17] S. Jablonski, C. Bussler, Workflow Management - Modeling Concepts, Architecture and Implementation, International Thomson Computer Press, Sept. 1996

[18] M. Kamath, K. Ramamritham, N. Gehani, D. Lieuwen, WorldFlow: A System for Building Global Transactional Workflows, Proc. of the 7th International Conference on High Performance Transaction Systems (HPTS'97), Asilomar, California, Sept. 1997

[19] J. B. Masters, Peep-to-Peer Technologies and Collaborative Work Management: The Implications of "Napster" for Document Management, in L. Fischer, ed. Workflow Handbook 2002, 81-94, 2002

[20] Matrix project, San Diego Supercomputer Centre, http://www.npaci.edu/DICE/SRB/matrix

[21] J. Miller, D. Palaniswami, A. Sheth, K. Kochut and H. Singh, WebWork: METEOR2's Web-Based Workflow Management System, Journal of Intelligent Information Systems - Special Issue on Workflow Management Systems, 10(2): 185-215, Kluwer Academic Publishers, Mar./Apr., 1998

[22] P. Muth, D. Wodtke, J. Weissenfels, A. Kotz Dittrich and G. Weikum, From Centralised Workflow Specification to Distributed Workflow Execution, Journal of Intelligent Information Systems - Special Issue on Workflow Management, 10(2): 159-184, Kluwer Academic Publishers, Mar. 1998

[23] G. Piccinelli, A. Finkelstein and S. L. Williams, Service-Oriented Workflow: The DySCo Framework, Proc. of the 29th Euromicro Conference (EUROMICRO'03), 291-297, Belek-Antalya, Turkey, Sept. 2003

[24] C. Schuler, H. Schuldt and H. Schek, Supporting Reliable Transactional Business Processes by Publish/Subscribe Techniques, Lecture Notes in Computer Science, Vol. 2193, 118-131, Springer-Verlag, 2001

[25] H. M. W. Verbeek, A. Hirnschall and W. M. P. van der Aalst, XRL/Flower: Supporting Inter-organizational Workflows using XML/Petri-net Technology, Web Services, E-business and the Semantic Web, Lecture Notes in Computer Science, Vol. 2512, 93-108, Springer-Verlag, 2002

[26] J. Yan, Y. Yang and G. K. Raikundalia, A Decentralised Architecture for Workflow Support, Proc. of the 7th International Symposium on Future Software Technology (ISFST2002), CD ISBN: 4-916227-14-X, Wuhan, China, Oct. 2002

[27] J. Yan, Y. Yang and G. K. Raikundalia, A Data Storage Mechanism for Peer-to-Peer Based Decentralised Workflow Systems, Proc. of the 15th International Conference on Software Engineering and Knowledge Engineering (SEKE2003), 354-358, San Francisco, USA, July 2003

[28] J. Yan, Y. Yang and G. K. Raikundalia, Enacting Business Processes in a Decentralised Environment with p2p-based Workflow Support, Advances in Web-Age Information Management, Lecture Notes in Computer Science, Vol. 2762, 290-297, Springer-Verlag, 2003

[29] J. Yan, Y. Yang and G. K. Raikundalia, Decentralised Coordination for Software Process Enactment, Software Process Technology, Lecture Notes in Computer Science, Vol. 2786, 164-172, Springer-Verlag, 2003

[30] J. Yan, A Framework and Corresponding Technologies for peer-to-peer based Decentralised Workflow Systems, PhD thesis, Swinburne University of Technology, Aug. 2004

[31] Y. Yang, An Architecture and the Related Mechanisms for Web-based Global Cooperative Teamwork Support, International Journal of Computing and Informatics, 24(1): 13-19, 2000

[32] Y. Yang, Tool Interfacing Mechanisms for Programming-for-the-large and Programming-for-the-small, Proc. of the 9th Asia Pacific Software Engineering Conference (APSEC02), 359-365, Gold Coast, Australia, Dec. 2002

Jun Yan was born in JiangSu, China, in 1976. He received the Bachelor of Engineering degree and the Master of Engineering degree, both in computer application technologies, from Southeast University, Nanjing, China, in 1998 and 2001, respectively, and the PhD degree in information technology from Swinburne University of Technology, Melbourne, Australia, in 2004.

He is currently a Postdoctoral Research Fellow in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Australia. His research interests include process management and workflow technology, Grid services and Web services, Internet computing and e-commerce.

Yun Yang (M'97) was born in Shanghai, China. He received the Bachelor of Science degree from Anhui University, Hefei, China, in 1984, the Master of Engineering degree from The University of Science and Technology of China, Hefei, China, in 1987, and the PhD degree from The University of Queensland, Brisbane, Australia, in 1992, all in computer science.

He is currently a full Professor and Academic Leader of computer science and software engineering discipline in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Australia. Prior to joining Swinburne as an Associate Professor, he was a Lecture and Senior Lecturer at Deakin University during 1996-1999. Before that, he was a (Senior) Research Scientist at DSTC - Cooperative Research Centre for Distributed Systems Technology during 1993-1996. He also worked at Beijing University of Aeronautics and Astronautics during 1987-1988. He has edited one book and published more than 100 papers on journals and refereed conferences. His research interests include software technologies, p2p and grid workflow systems, service-oriented computing, Internet computing applications, CSCW and e-business processes.

Gitesh K. Raikundalia (M'00) received the Bachelor of Economics degree from The University of Sydney, Australia, in 1991, the Master of Computing degree from The University of Newcastle, Australia, in 1994, and the PhD degree in CSCW from Bond University, Australia, in 1998.

He is currently a Senior Lecturer in the School of Computer Science and Mathematics and a member of the Internet Technologies & Applications Research Lab at Victoria University, Melbourne, Australia. He was a Lecturer at Swinburne University of Technology, Melbourne, Australia, previously. Prior to this, he was an Associate Lecturer and then a Lecturer at Southern Cross University, Coffs Harbour, Australia. His research interests include electronic meeting systems, real-time collaboration and group awareness, workflow and scenario-based design.

Dr Raikundalia is a member of Association for Computing Machinery and Australian Computer Society. He is a consulting editor for the Australasian Journal of Information Systems and has served on various programme committees, particularly those of the Australasian Document Computing Symposium (for which he has been Symposium chair twice) and the Asia Pacific Web Conference.