6-4-2006

SwinDeW-S: Extending P2P Workflow Systems for Adaptive Composite Web Services

Jun Shen
University of Wollongong, jshen@uow.edu.au

Jun Yan
University of Wollongong, jyan@uow.edu.au

Yun Yang
Swinburne University of Technology

SwinDeW-S: Extending P2P Workflow Systems for Adaptive Composite Web Services

Abstract

SwinDeW, an innovative decentralised workflow management system, has established an underlying framework for peer-to-peer (p2p) based business process coordination environments. SwinDeW-S extends SwinDeW to support adaptive composite service orchestration in the era of service-oriented computing. This paper comprehensively presents features of SwinDeW-S, including the p2p network establishment, the messaging mechanism, the service deployment and enactment, the service discovery and advertisement, and the service flow execution. The prototypical extension of SwinDeW to SwinDeW-S and the advantages of SwinDeW-S are also examined and analysed. With the innovative integration of service and p2p-based enterprise application techniques, SwinDeW-S can support composite service orchestration, deployment and execution.

Disciplines

Physical Sciences and Mathematics

Publication Details

This paper was originally published as: Shen, J, Yan, J & Yang, Y, SwinDeW-S: Extending P2P Workflow Systems for Adaptive Composite Web Services, 2006 Australian Software Engineering Conference (ASWEC 2006), Sydney, Australia, 6 April, 2006, 61-69. Copyright IEEE.

SwinDeW-S: Extending P2P Workflow Systems for Adaptive Composite Web Services

Jun Shen[1, 2, 3], Jun Yan[1, 2], Yun Yang[2]

1-School of Information Technology and Computer Science
University of Wollongong
Wollongong, Australia, 2522
solo.shen@gmail.com; jyan@uow.edu.au

2-Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, Melbourne, Australia, 3122
yyang@ict.swin.edu.au

3-School of Computer and Information Sciences
University of South Australia
Mawson Lakes, Adelaide, Australia, 5095

Abstract

SwinDeW, an innovative decentralised workflow management system, has established an underlying framework for peer-to-peer (p2p) based business process coordination environments. SwinDeW-S extends SwinDeW to support adaptive composite service orchestration in the era of service-oriented computing. This paper comprehensively presents features of SwinDeW-S, including the p2p network establishment, the messaging mechanism, the service deployment and enactment, the service discovery and advertisement, and the service flow execution. The prototypical extension of SwinDeW to SwinDeW-S and the advantages of SwinDeW-S are also examined and analysed. With the innovative integration of service and p2p-based enterprise application techniques, SwinDeW-S can support composite service orchestration, deployment and execution.

1. Introduction

Service-Oriented Computing (SOC) is aiming at developing a platform and mechanisms to integrate information systems universally in a loosely coupled way. The basic elements for this paradigm are Web services. A Web service is an independent software component, which can be accessed by applications and other Web services using Web standards such as HTTP, SOAP, WSDL [5] and UDDI. Web services can be used to provide entry points to organisations' inner processes. Therefore, business processes across organisational boundaries can be implemented based on services. Building a composite Web service from component services is referred to as composite service orchestration. One trend in orchestrating services today is to use the workflow technology in the light of the industry de facto standard BPEL4WS [3]. In terms of workflows, there are two typical system architectures for workflow management, namely, the centralised client-server based architecture and the decentralised peer-to-peer (p2p) based architecture. Very recently, grid workflows have also been addressed which are mainly for computation intensive e-science and e-business processes.

The centralised architecture brings main weaknesses like poor performance, limited scalability, vulnerability, inflexibility and lack of openness, which resulted in the development of SwinDeW (Swinburne Decentralised Workflow), a p2p based workflow management system [16]. The advantage of SwinDeW is that the p2p architecture better reflects the decentralised nature of processes. Currently, as described in this paper, SwinDeW is extended to support orchestration of composite e-services. This new version of SwinDeW is named as SwinDeW-S (SwinDeW for Services). With SwinDeW-S, existing services can be deployed and orchestrated in a naturally decentralised p2p environment more flexibly and efficiently. Hence, once services are deployed, coordinated and monitored workflow peers from anywhere may invoke or execute them automatically. The heterogeneity involved may be tackled by XML-based business process and service languages. Thus, our prototype can be applied in more open real world enterprise integration environments.

In this paper we will present the architecture, features and applications of SwinDeW-S in composite service orchestration. The rest of this paper is organised as follows. In the next section, we analyse the requirements for SwinDeW-S. In Sections 3 and 4, the overall architecture and design of SwinDeW-S are presented, respectively. The prototypical mechanism of the p2p based service environment are detailed in Section 5.

Proceedings of the 2006 Australian Software Engineering Conference (ASWEC'06)
1530-0803/06 $20.00 © 2006 IEEE

Section 6 discusses major related work, followed by conclusions and future work in Section 7.

2. Requirements analysis

Almost all companies have business processes unique to their activities. Automating organisations' business processes is a very important topic in today's information system research. Business processes tend to evolve from time to time due to the changing business environment. This requires companies to be more flexible and to react faster. Business Process Management (BPM) aims at improving the efficiency and effectiveness of organisations' operations by providing solutions for organisations to be easily adaptable to the new conditions and environments. Workflow management [1] represents the operational aspect of a business process that specifies the order of tasks, the parties to perform the tasks, the data flows required for the tasks and the monitoring methods to control them [2, 16, www.wfmc.org].

In a WorkFlow Management System (WfMS), the build-time functions allow users to define the processes in terms of tasks, process logic, and business data [9, 10]. The run-time functions control the creation and execution of process instances, allowing users to start, suspend, resume or terminate a process instance. The data correlation among the process instances also belongs to run-time functions. In the past two decades, research and practice in the workflow management area have gained significant achievements. However, there still exist limitations in the conventional WfMS such as poor performance, limited scalability, vulnerability, inflexibility, and lack of openness [16]. In [16], the authors point out that the main cause of the above weaknesses is the client-server architecture adopted in the conventional WfMS. In the light of service oriented computing architectures, business processes by nature involve partners operating in a decentralised p2p manner with multiple activities being run in proactive and parallel services. Therefore, extending the conventional workflow system to service-oriented architecture becomes necessary.

Our research group has developed SwinDeW. Based on that, we need to upgrade SwinDeW to SwinDeW-S in order to support deployment and execution of e-services and processes. For doing so, there are some requirements for SwinDeW-S. Firstly, for concrete process execution, we must deal with inputs, outputs, preconditions and effects (IOPE) of related services. While SwinDeW itself currently only concerns about the coordination of tasks, SwinDeW-S should adaptively materialise the interdependency between services and business processes. For retrieval, process related information, those traditionally stored in a centralised data repository, should be maintained in a decentralised mode without losing information so that relevant peers in the system can access the data when necessary. Secondly, for system openness, the system should flexibly migrate and locate the services, those traditionally performed by a centralised workflow engine, to other ordinary sites (peers) in the whole system so that the decentralised run-time environment can be coordinated and self-managed effectively. Thus we must adapt SwinDeW in order to provide a plug-and-play framework for integrating workflow process applications, services and human participants which imply the non-existence of either a centralised data repository for data storage, or a centralised workflow engine for coordination. To improve system performance, we should provide means to efficiently locate service providers in a way that the traffic incurred by request and response messages could be guided to appropriate peers automatically.

3. SwinDeW-S architecture

SwinDeW is a decentralised workflow system built on top of JXTA, a Java implementation of a set of open, generalised p2p protocols developed by Sun Microsystems. The main purpose of JXTA is to specify a platform-independent framework that supports the common functions for all kinds of p2p networks, such as peer discovery and data communication. The formal specification and the complete guide of JXTA are covered in [8]. From a developer's point of view, SwinDeW's functions are divided into workflow functions and p2p network functions. On one hand, process definition, process instantiation and enactment, and process monitoring and administration are core workflow functions. On the other hand, the group service, peer service, pipe service, discovery service and advertisement service are p2p network functions. They are core built-in functions of the JXTA binding and can be invoked through the Java API. Peers on the network cooperate with each other by means of the core JXTA services performing workflow functions. This can be seen in all operation mechanisms of SwinDeW. One key mechanism is that SwinDeW peers form virtual communities. Each community is characterised by a capability. If a peer has a capability, it will automatically be a member of the community with that capability. This mechanism relies on the group service provided by JXTA.

After a peer accepts and instantiates an assigned task, it will consult the process definition to determine each succeeding task. Then the peer will look for the

community that has the required capability for the succeeding task and ask the community for a peer that can undertake the task. On receiving the request, peers in the community will negotiate to find the best peer for the task. This mechanism is supported by the peer group service, the discovery service and the pipe service of JXTA.

SwinDeW-S has added one more layer on top of SwinDeW as illustrated in Figure 1. The lowest layer is JXTA, which supports basic p2p functions that will be exploited by the middle layer peers. The upper layer is composed of deployed services, which are supported by SwinDeW-S. At the process instantiation stage, SwinDeW-S peers are discovered for tasks in a similar mechanism as in SwinDeW. A peer with the service that matches the capability of a task request is selected for the task. If there is more than one peer matching a certain task, the peer with the lowest workload will win. However, it is not necessary for each peer to have one-to-one mapping to a specific service. In other words, it is flexible for a peer to play different roles and execute different services during its lifecycle.

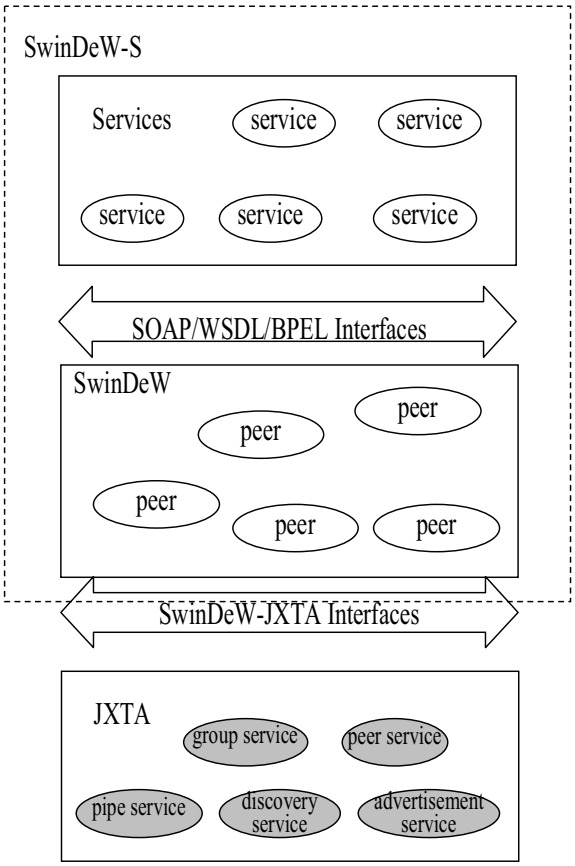

Figure 1: SwinDeW-S's functional architecture

SwinDeW-S has all the advantages of the decentralised composite service engines. Each peer joining the network can be an engine sharing the work of instantiating composite services and executing them. Therefore the more peers in the network, the better the overall scalability and efficiency of the system when compared with the client-server counterpart. Also, because peers communicate directly with other peers for data exchange, data are well distributed on the network and therefore the bottleneck block can be avoided. Another merit value of SwinDeW-S is that with the support of the p2p network, business partners only need to run SwinDeW-S peers on their own machines and register their Web services with the peers. Then the services will be dynamically discovered and bound in within the network. Furthermore, SwinDeW-S opens the opportunity to develop a sophisticated automatic discovery with more criteria and semantics. This is invaluable because customers will have a good solution for their needs and companies have chances to satisfy customers who need their services.

4. SwinDeW-S design

4.1 Service flow description in SwinDeW-S

We use BPEL4WS [3] to describe the composition of services to make a complex Web service because BPEL4WS is currently a mature language specifically for Web service based business processes. However, the challenge is that we have to fragment and distribute the BPEL4WS process description into a p2p network appropriately so that the distributed version is functionally equivalent. In BPEL4WS a control flow is constrained by both the structure activities and the links. The structure activities include the *flow* activity, the *sequence* activity, the *while* activity, the *switch* activity and the *pick* activity. It is easy for a centralised engine like IBM's BPWS4J to directly read a BPEL4WS process description and deploy it. However, the situation is difficult for a p2p based service engine. The problem is that in a p2p network, each peer does not and should not know all the information about the process but only the information that is necessary to carry out its mission. Therefore, it is necessary to convey the information presented by the structure activity in a p2p network.

Our solution is to convert a BPEL4WS process into the CFG (Control Flow Graph) form. Nodes in a CFG graph are basic activities. Each node knows a set of its predecessors and a set of successors as well as the conditions for it to be executed, if any. We developed a conversion algorithm named *Conv* whose input is a node of a tree *subRoot*. If *subRoot* is a <process> node, *Conv* will recursively apply the same conversion on the child node. If *subRoot* is a <flow> node, *Conv* will add all the

predecessors and all the successors of *subRoot* to the predecessor list and the successor list of each of *subRoot*'s child nodes, respectively. If *subRoot* is a <sequence> node, *Conv* will add all the predecessors of *subRoot* to the predecessor list of the first (leftmost) child of *subRoot* and add all the successors of *subRoot* to the successor list of the last (rightmost) child of *subRoot*. Then for each child of *subRoot* other than the first child, *Conv* adds the last atomic descendant node(s) of the node directly on the left of the child to the child's predecessor list. And for each child of *subRoot* other than the last child, *Conv* adds the first atomic descendant node(s) of the node directly on the right of the child to the child's successor list. The last atomic descendant node of a node is the node that represents an atomic activity and is positioned at the rightmost position of the sub tree rooted at the node. The first atomic descendant node of a node is the node that represents an atomic activity and positioned at the leftmost position of the sub tree rooted at the node. After *subRoot* is processed, the same conversion is recursively applied to each of its child nodes. Therefore the conversion processes all the nodes of the tree until all the leaves of the tree are processed. After the conversion completes, the atomic nodes in the tree are extracted and put into a list. The list is the graph version of the tree containing sufficient process structure information inside each of the atomic nodes.

Together with their own knowledge, the nodes maintain the same IOPE information kept by the structure activities. The knowledge about links is still preserved in each node and therefore the structure formed by the links is retained. In typical workflows, inputs and outputs between tasks are always business documents. To handle them, we use MIME-based attachments as BPEL input and output XML messages, which are supported by Sun's SAAJ packaged in J2EE.

4.2 Composite service discovery and distribution in p2p environment

Suppose a complex service workflow system for loan approval as shown in Figure 2, the tasks are distributed to a set of peers. Each peer must have some services in order to create and run any business process tasks. Each peer can only create and run tasks that require the services this peer can provide. If it has not been done already, a list of services needs to be created so that all peers can choose from the same list. For this to work properly, all peers in a virtual community need to be running when you add a new capability so that they all can receive the new information. The user input in Figure 2 will be translated into the BPEL4WS format, which will guide coordinating peers to deploy and enact services on corresponding peers or peer groups.

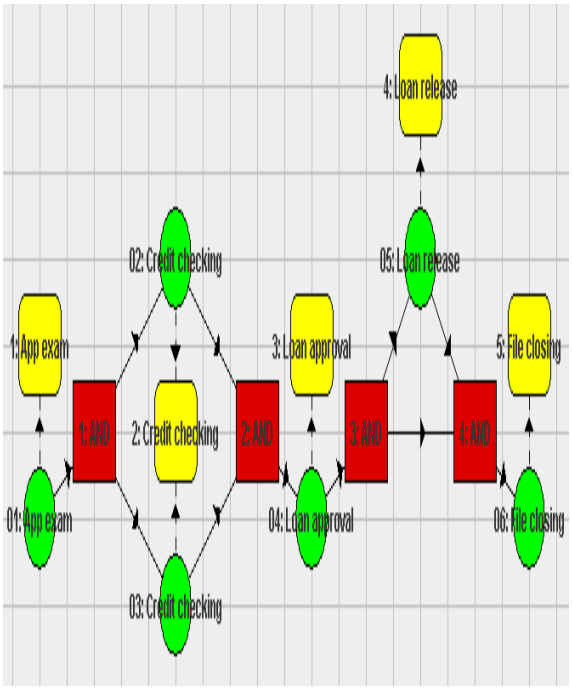

Figure 2: Loan approval process in SwinDeW-S

The peer discovery requires the description of activities at the peer that hosts the BPEL4WS process description and at the same time at the peers on the network that are capable of performing the activities. The more semantics the description carries, the better chance for the most suitable peer to be found for an activity, and hence the more effective interoperability the system offers. In current SwinDeW-S, for the purpose of illustration, we just use a simple description mechanism for activities, i.e. using their names as described in WSDL and UDDI. In the future, we will integrate OWL-S [12] to enhance the semantics of activity descriptions.

A peer is capable of performing an activity if at the build-time the administrator registers the activity to its capability manager. The invoked activities are registered only if the peers, which they are registered to, can deploy the component services they invoke. At run-time, after the coordinator peer, who plays temporary coordinating role, resolves a BPEL4WS process into basic activities, i.e., atomic services. The peer seeker of the coordinator will discover a suitable peer to host the service by sending a request message to peer groups that are able to perform the activity. Upon receiving the message, peers in each peer group will negotiate to find the most suitable peer for the activity or service. The peer that is chosen will reply to the current coordinator with a reply message. When all activities have known

their associated peers, the coordinator will generate routing data for each peer so that it knows where its predecessors and successors are, where the activities at the other end of its source links are and where the activities that use the same variables are, according to IOPE matchmakings. After the routing data are set up the current coordinator will send the detailed information of each activity/service to its corresponding peer. Then, the process can be executed.

4.3 Composite service execution in p2p environment

At run-time, an activity is executed when its start conditions are satisfied. The conditions include all of its predecessors having been completed and the join condition and the branching condition, if any, turning true. When the activity is complete, the peer will set its source link values according to the source links' transition condition expression. Then the peer will notify its successors about its completion by sending the source link values and variable values to the corresponding peers.

If an activity knows that it is not executed it will set its source link values to false and send them to the corresponding peers in order for the peers to decide if their activities are to be run or not. The first activities are often the receive activities and the last activities are often the reply activities. They are always executed on the current temporary coordinator peer to receive the message submitted by users or other applications and Web services and to send the result back to them. When an invoked activity is executed, it will call its component service, which is being deployed in the J2EE Sun Application Server on the same host. If the component service has not been deployed yet, the activity will deploy it automatically.

5. SwinDeW-S prototype

5.1 Composite service orchestration and execution

We extend SwinDeW smoothly for the purpose of composite service orchestration and execution. A peer has a capability and can join a peer group only if it can deploy a service on it. At the initial stage when a peer joins the JXTA network, the *Peer* object deploys all the Web services that the peer has by executing a batch file.

When the batch file is executed, it first facilitates the document transfer services between peers with a dedicated I/O mechanism. Essential documents which act as inputs and outputs between workflow nodes will be carried through the p2p network as SOAP attachments. It then invokes ASANT to deploy Web services on the J2EE platform running on the peer. ASANT is a portable command-line build tool extended from the ANT tool which is developed by the Apache Software Foundation. ASANT adds some more functions that can interact with the J2EE server administration functions. When ASANT is invoked, it will read the XML file for service build task, locating the target information it needs to deploy the Web services.

5.1.1 Publishing and discovering services in SwinDeW-S

In SwinDeW-S, peers join the groups in order to publish their services and to discover services they need. The peers' administrators decide which groups their peers will join based on the areas that their services would belong to. For example, some services might be booking for hotel rooms while some others might be looking for car retailers.

- Joining groups also supports peers to discover services in an elegant way. A peer finds the services it needs based on service descriptions about its desired services specified at build-time. Besides the WSDL descriptions, the description also includes the commercial groups that the desired services belong to.
- At run time, when a peer finds a service, it will check whether it is joining to the group that the service belongs to. If so, it will broadcast a discovering request message in the group and wait for the response. Otherwise, it will broadcast the message in all the groups that it joins.
- Upon receiving the message, if a peer in the network has the discovered service, it will send an acknowledge message back to the peer who starts the discovery. Otherwise, it will continue in the process of finding service by the same protocol described. In this way, we can ensure that the service will always be found if it really exists in the network while minimising the finding time as much as we can.

5.1.2 Communicating messages in SwinDeW-S

In SwinDeW-S all kinds of content to communicate over the p2p network are wrapped in instantiated objects of the *SwinDeW message* class. When a peer sends a *SwinDeW message* object, the object is converted to the XML format and put into an object of *Message*, a JXTA

class. The *Message* object is then transmitted in the network. At the receiving end, the receiver gets the *Message* object, extracts the XML message and converts it back to the *SwinDeW message* object. Finally, the content can be derived from this object. Below is an example of a *SwinDeW message* in XML format.

```
<Message>
  <Type>3</Type>
  <Count>0</Count>
  <Start>1111286628625</Start>
  <PeerName>Peer3</PeerName>
  <VisitedPeers>
    <Peer>Peer4</Peer>
    <Peer>Peer2</Peer>
  </VisitedPeers>
  <Advertisement>%3C%3Fxml+version%3D
      %221.0%22%3F%3E%0A%0A%3C%21DOC
      TYPE+jxta%3APipeAdvertisement%
      3E%0A%0A%3Cjxta%3APipeAdvertis
      ement+xmlns%3Ajxta%3D%22http%3
      A%2F%2Fjxta.org%22%3E%0A%09%3C
      Id%3E%0A%09%09urn%3Ajxta%3Auui
      d9616261646162614E504720503250
      33EBF62B92D0F349A7A59099E11C57
      CBB304%0A%09%3C%2FId%3E%0A%09%
      3CType%3E%0A%09%09JxtaPropagat
      e%0A%09%3C%2FType%3E%0A%09%3CN
      ame%3E%0A%09%09NetPeerGroup%2F
      Peer3%0A%09%3C%2FName%3E%0A%3C
      %2Fjxta%3APipeAdvertisement%3E
      %0A
  </Advertisement>
  <Body>
    <XMLData
      ClassName="swindew.message.WebSe
      rviceFindingMessage">
      <CommercialGroup>SWINDEW_GLOBAL<
        /CommercialGroup>
      <PortType>loanApprovalPT</PortTy
        pe>
      <ActivityName>invokeApprover</Ac
        tivityName>
    </XMLData>
  </Body>
</Message>
```

Every *SwinDeW message* has two parts, the header and the content. Compared to a *SwinDeW message* in SwinDeW, there are some features added to the header part of *SwinDeW message* for SwinDeW-S to support the service discovery protocol presented above. First, to avoid messages from being propagated forever in the network attribute *Count* is assigned to each message. When a message is generated, its *Count* attribute is set to 0. Each time the message jumps to a node in the network, *Count* is increased by 1. If the *Count* reaches a certain specified limit the message will be discarded from the network to avoid routing deadlock or loops. Each message is also assigned a time stamp in element *Start*. This attribute is used to create a timing window for response messages. For example, when a peer sends a request message, it sets *Start* to the current logic time of the message generation, sends the message and waits for the response. When a response message arrives, the peer compares the current time with the start of the response. If the difference is within a certain specified timeout the response will be processed, otherwise it will be discarded. Therefore, only response messages arrive within the specified timing window are processed. The name of the peer that generates a discovering request message is also attached in the header. This prevents receiving peers from doing duplicated work. The body part of a *SwinDeW message* is encoded in XML format. This XML piece keeps the full state of the object at the sending end. The responding peers try to match its own service profile with requested WSDL PortType and BPEL activity name as well as IOPE, if necessary. Based on the *PeerName* attribute and the *Start* attribute of any two messages that have the same *PeerName* and the same generation time, a peer can know that they are carrying the same content and it will discard the second message. To support this detection each peer has to maintain a history of incoming messages. Finally, attribute *visitedPeers* helps each peer to immediately detect and discard messages that has come to and been processed by them before. By checking whether it is included in the list of the visited peers of the message the peer can decide whether it is a new message or not.

5.1.3 Executing composite services in SwinDeW-S

The mechanism of launching composite services in a p2p network in SwinDeW-S remains unchanged as that of launching workflow in SwinDeW. Each task in SwinDeW-S maintains a list of its predecessors and a list of its successors. When a predecessor finishes its job it will send the task a message. When it receives the message from all of its predecessors the task begins to do its job. The task invokes its service through a service client. A client is a class that implements the Web service interface. It is precompiled and put into the system by the peer's administrator. First, the task uses the service port type name to request the peer to find a client that is used to call the service of the port type. The peer will return with the full name of the client class. The task then instantiates an object of the client class and requests it to call the service. This allows administrators to easily put their services in use. They simply build a service, deploy it somewhere, then write a client class for the service, compile it and register its name to their peers. The services and the clients can be implemented in anyway, using any technology.

After finishing its job, the task sends data to other tasks in other peers. The data are kept by the *Variable* objects. Each *Variable* object is equipped with a data routing table so that it knows which peers on the network have tasks that also need it. When a peer sends

data to other peers, it puts the *Variable* object into the body part of a *SwinDeW message* and sends the message to the peers. Finally the peers responsible for the last tasks of the process finish their jobs and return the results back to the current coordinator peer to notify the completion.

5.2. Prototype

Figure 3 is the SwinDeW-S interface for users to add capable services to specific peers. We can add the 'loanapprover' service to *Peer 1*, who has already been deployed with another service 'examiner'.

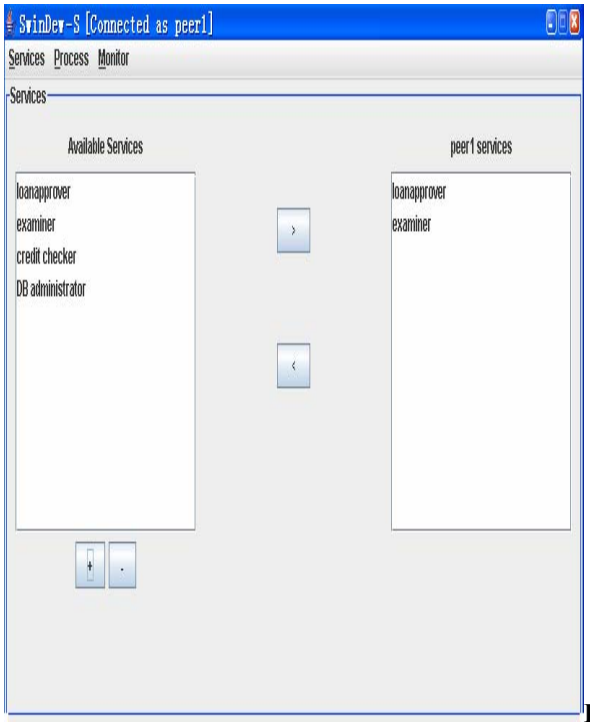

Figure 3: Adding services

The following BPEL4WS definition is a typical example of IOPE elements of the process, where application examiner receives inputs from loan applicant and then forwards them to credit checker to assess the risk. Based on the pre-condition of risk values, the effects are composed of further actions, that is, loan release by assigning 'yes' to the reply messages as an output, or another request for extra loan approval process and response accordingly.

```
<process name="loanApprovalProcess">
  <flow>
    <receive name="receive1"
      partnerLink="customer"
      portType="apns:loanApprovalPT"
      operation="app_exam"
      variable="request"
      createInstance="yes">
      <source linkName="receive-to-
      assess"
      transitionCondition="bpws:getV
      ariableData('request',
      'amount')&lt;10000"/>
      <source linkName="receive-to-
      approval"
      transitionCondition="bpws:getV
      ariableData('request',
      'amount')&gt;=10000"/>
    </receive>
    <invoke name="invokeExaminer"
      partnerLink="examiner"
      portType="asns:riskAssessmentP
      T" operation="credit_check"
      inputVariable="request"
      outputVariable="riskAssessment
      ">
      <target linkName="receive-to-
      assess"/>
      <source linkName="assess-to-
      setMessage"
      transitionCondition="bpws:getV
      ariableData('riskAssessment',
      'risk')='low'"/>
      <source linkName="assess-to-
      approval"
      transitionCondition="bpws:getV
      ariableData('riskAssessment',
      'risk')!='low'"/>
    </invoke>
    <assign name="assign">
      <target linkName="assess-to-
      setMessage"/>
      <source linkName="setMessage-
      to-reply"/>
      <copy>
        <from expression="'yes'"/>
        <to variable="approvalInfo"
      part="accept"/>
      </copy>
    </assign>
    <invoke name="invokeApprover"
      partnerLink="approver"
      portType="apns:loanApprovalPT"
      operation="loan_approve"
      inputVariable="request"
      outputVariable="approvalInfo">
      <target linkName="receive-to-
      approval"/>
      <target linkName="assess-to-
      approval"/>
      <source linkName="approval-to-
      reply" />
    </invoke>
    <reply name="reply"
      partnerLink="customer"
      portType="apns:loanApprovalPT"
      operation="loan_release"
      variable="approvalInfo">
      <target linkName="setMessage-
      to-reply"/>
      <target linkName="approval-to-
      reply"/>
    </reply>
  </flow>
</process>
```

6. Related work

Defining a descriptive and effective language to describe composite services is the core in the field of Web service composition. IBM, Microsoft and BEA are cooperating to define BPEL4WS (Business Process Execution Language for Web Services) [3]. BPEL4WS,

currently at version 1.1, allows modelling the behaviour of Web services in a business process interaction. It can be used to describe both executable processes and abstract processes and support for long running transactions. A BPEL4WS process, when running on an engine, has an interface to the outside world like any normal Web services. And it is supposed to be run by a centralised engine. Some extra operations to partition it into a decentralised workflow are needed for it to be deployed on a p2p or Grid network.

An ongoing p2p-based workflow project is conducted at Manchester Metropolitan University. This project presents a p2p architecture for dynamic workflow management, which is based on concepts such as Web Workflow Peers Directory (WWPD) and Web Workflow Peer (WWP) [7]. Benatallah et al. [4] proposed a peer-to-peer architecture, SELF-SERV, to execute composite Web services to overcome the shortcomings originating from the traditional client-server architecture. Their work presents the division and distribution of the work of composite Web service execution to multiple hosts. Developed by San Diego Supercomputer Centre, Matrix (www.npaci.edu/DICE/SRB/matrix) project delivers Grid workflow protocols and workflow language descriptions necessary to build a p2p infrastructure for Grid WfMS. This middleware allows applications and services based on Web service standards to communicate with data and other resources in Grid environments. PeCo [6] decentralises workflow by using a pluggable framework for integrating business process applications and human contributors. Though PeCo is aware of Web services, its deployed plug-in peers have not support service interfaces yet.

Automatic Web service discovery and dynamically binding component Web services at run-time are important aspects in orchestrating composite Web services. A significant attempt has been spent on leveraging OWL-S to add rich semantics to Web service descriptions. Researchers of W3C have formed a working group to focus on this branch of OWL, and hence the introduction of OWL-S. OWL-S can be thought of as WSDL and BPEL4WS plus rich semantics. Research groups at Stanford University and Carnegie Mellon University [11, 13] have been successful in mapping WSDL service descriptions to OWL-S profiles. We also have done some work in this area of semantic Web services [14]. In [15], we established bridges between BPEL4WS and OWL-S. These achievements are very valuable in enhancing Web service profiling and discovery.

Our work in this paper has integrated a p2p workflow environment and composite services architecture to support adaptive and decentralised service deployment and execution. The prototype is a light-weight tool, which transforms conventional centralised business processes into a loosely coupled service enactment environment. With the popularity of the p2p systems, SwinDeW-S facilitates a natural mechanism for modern Web-based business practices. Currently, the corresponding peers join or leave peer groups by claiming or disclaiming the service capabilities which they are responsible at the deployment or execution stage. We expect that the flexibility of this grouping process can be boosted with explicit accompanied OWL-S specifications, which can even support the descriptions of service quality metrics.

Compared with major related work, SwinDeW-S demonstrated some advanced features. By involving many peers in executing processes, the performance can be ensured. When the number of processes deployed and the complexity of the processes increase, more peers can be added to the network to keep the same level of performance. In addition, when more clients request services, more peers can be added to ensure the same level of service quality. It can cope with the expansion of organisations much easily as workload can be evenly distributed among existing and new peers. When considering implementation, it is more difficult to correlate between messages and process instances. Security is also harder to deal with. However it is easier to implement concurrent processing by taking advantage of all the available resources on the p2p network. Moreover, data is distributed almost equally on the p2p network, therefore the risk of data block is lower. Finally, Web services can be published and dynamically discovered inside the system. The system is open for more advanced Web service descriptions with rich semantics to be integrated in.

7. Conclusions and future work

In this paper we have presented our investigation of SwinDeW-S, an extension of p2p based workflow system called SwinDeW, for the purpose of orchestrating and executing composite services. The adopted messaging mechanism enables the system to deal with inputs, outputs, preconditions and effects more flexibly. The service deployment and discovery are naturally migrated into SwinDeW for SwinDeW-S so that service orchestration and enactment become more adaptive. The decentralised run-time environment can be coordinated and self-managed effectively with services being located to wide area peer hosts, who communicate with each other according to the de facto standard business process or workflow definitions.

In the future, some improvements still have to be carried out for SwinDeW-S. We will fully support

BPEL4WS to describe composite Web services more effectively. SwinDeW-S will also be enhanced with more suitable data distribution and communication mechanisms. Another challenging work is to facilitate automatic service discovery with rich semantics of OWL-S in terms of service quality.

Acknowledgement

The research work reported in this paper is jointly supported by Australian Research Council under Discovery Grant DP0210654 and DP0663841, and Linkage Grant LP0562500. The IBL student Jonathan Derham and Master of IT student Quang Huy Vu contributed significantly to the prototyping of SwinDeW-S.

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

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic and S. Weerawarana. "Business Process Execution Language for Web Services Version 1.1", *BEA, IBM and Microsoft*, http://www-106.ibm.com/developerworks/library/ ws-bpel/, August 2002.

[4] B. Benatallah, M. Dumas, Q. Sheng, and A.H.H. Ngu. "Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services". *Proc. of 18th Int. Conference on Data Engineering (ICDE'02)*, pp.297–308, San Jose, USA, February 2002.

[5] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana. "Web Service Description Language (WSDL) 1.1", http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001.

[6] M. D. Coon. *Peer-to-Peer Workflow Management White Paper*, http://www.proteus-technologies.com/cmm/docs /P2P_Workflow_Whitepaper.doc, August 2002.

[7] 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.

[8] J. D. Gradecki and J. Gradecki. *Mastering JXTA: Building Java Peer-to-Peer Applications*, John Wiley & Sons, 2002.

[9] F. Leymann and W. Altenhuber. "Managing Business Processes as an Information Resource", *IBM Systems Journal* 33(2): 326-348, 1994.

[10] F. Leymann and D. Roller. "Business Process Management with FlowMark", *Proc. of 39th IEEE Computer Society International Conference: Technologies for the Information Superhighway (COMPCON'94)*, pp. 230-234, San Francisco, USA, March 1994.

[11] D.J. Mandell and S. McIlraith. "Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation", *Proc. of 2nd International Semantic Web Conference (ISWC 2003), LNCS 2870*, pp.227-241, Sanibel Island, USA, October 2003.

[12] D. Martin et al. (eds) OWL-S 1.1 Release. "Upper Ontology for Services", http://www.daml.org/services/owl-s/1.1/. November 2004.

[13] M. Paolucci, N. Srinivasan, K. Sycara and T. Nishimura. "Toward a Semantic Choreography of Web Services: From WSDL to DAML-S", *Proc. of 1st International Conference on Web Services (ICWS'03)*, pp.22-26, Las Vegas, USA, June 2003.

[14] J. Shen, Y. Yang and B. Lalwani. "Mapping Web Services Specifications to Process Ontology: Opportunities and Limitations", *Proc. of 10th IEEE International Workshop on Future Trends in Distributed Computing Systems (FTDCS'04),* pp.229-235, Suzhou, China, May 2004.

[15] J. Shen, Y. Yang, C. Zhu and C. Wan. "From BPEL4WS to OWL-S: Integrating E-Business Process Descriptions", *Proc. of 2nd IEEE International Conference on Services Computing (SCC 2005)*, pp.181-188, Orlando, USA, July 2005.

[16] J. Yan, Y. Yang and G. K. Raikundalia. "SwinDeW - A Peer-to-peer based Decentralised Workflow Management System", to appear, accepted by *IEEE Transactions on Systems, Man and Cybernetics, Part A* in January 2005, available at http://www.ict.swin.edu.au/personal/yyang /papers/SwinDeW-TS MC.pdf.