Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT

Information and Software Technology 45 (2003) 929–939

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

Conceptual framework and architecture for service mediating workflow management☆

Jinmin Hu, Paul Grefen*

Computer Science Department, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

Received 22 August 2002; revised 27 March 2003; accepted 7 April 2003

Abstract

This paper proposes a three-layer workflow concept framework to realize workflow enactment flexibility by dynamically binding activities to their implementations at run time. A service mediating layer is added to bridge business process definition and its implementation. Based on this framework, we propose an extended workflow management systems architecture, in which service contracting layer, service binding layer and service invocation layer are extensions to support the proposed service mediating concept. According to an enactment specification, different instances of the same activity can be bound to different services to achieve enactment flexibility. The conceptual framework and architecture together provide a comprehensive approach to flexible service enactment in B2B collaborative settings.

Keywords: Workflow management; Service mediation; Enactment flexibility; Dynamic binding; Business-to-Business collaboration

1. Introduction

In the past years, workflow management systems (WFMSs) have been widely adopted by enterprises to streamline business processes. They make business process execution more efficient by offering means for explicit process specification, automated interpretation of process specifications, assignment of tasks to actors, and management of process exceptions. As Internet technologies are rapidly developing, the focus of many companies is shifting from the inside to the outside business world. They have started to intensify the collaborations with their customers, suppliers and business partners. The relationships among companies are fast changing, which demands that Business-to-Business collaboration should be dynamically set up. To support this business target, workflow-based enterprise information systems have to be extended to support dynamic inter-enterprise collaboration, for example, electronic service outsourcing. The extended workflow system has to be flexible enough to cope with the rapidly changing requirements and unexpected failures. Hence, flexibility and the ability to support inter-enterprise cooperation are becoming major challenges for workflow management (WFM). This sets the research focus of this paper.

The concept 'workflow flexibility' has many inter-related meanings [1]: easy design and change, easy enactment of changes in running workflow instances, good support of exception handling and failure recovery, dynamic workflow schema evolution and so on. One of the most important forms of flexibility is enactment flexibility. Enactment flexibility means that different instances of an activity can be dynamically bound to different implementations at run time. The conceptual architecture of most WFMS exhibits similar flexibility in dynamically binding 'Activity Instances' with 'Actors' through the use of 'Roles'. However, in spite of conceptually separating activity definitions from their implementations, which is characterized by the typical workflow framework, full enactment flexibility has not been yet reached in most current WFMSs. This is due to the fact that most WFMSs establish the associations between the activities with their implementations at build time. The pre-established associations allow workflow engines to invoke the corresponding applications at runtime. However, these pre-established associations are not allowed to change at workflow runtime. For example, in the IBM MQSeries Workflow product [10],

☆ A short and earlier version of this paper has been published in the Proceedings of the Fourth International Symposium on Collaborative Technologies and Systems (CTS), Orlando, Florida, USA, 2003, pp. 23–29.

* Corresponding author.
E-mail address: p.w.p.j.grefen@tm.tue.nl (P. Grefen).

0950-5849/03/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0950-5849(03)00092-2

programs for executing workflow activities are registered in the build time environment. References to the registered programs are the properties of program activities call execution. The descriptions of the registered programs even contain the file paths of the programs, which are parts of the FDL (MQSeries Flow Definition Language) specification. So, to change the implementation of an activity is not possible at run time but only at build time: "…However if you have already imported the workflow model in run time, you must export it from build time and import it again in runtime to use the new definitions" [10]. Some extensions to the static model have been realized—we discuss an example in Section 2.

To attain full runtime enactment flexibility, a WFMS must realize a mechanism that supports dynamically establishing or changing the association between an activity and its implementation at run time. This paper provides an approach to this dynamic binding by adopting a service-oriented approach. Our approach is to separate the activity specification from the workflow process specification and dynamically combine it with the descriptions of applications or services to generate an enactment specification (ES), which is used to support dynamically binding an activity instance to an appropriate service.

The second focus of this paper is WFMS support for B2B collaboration. Traditionally, the emphasis of WFM has been placed on homogeneous environments within the boundary of a single organization. As e-business is growing, extending a WFMS to support B2B collaboration is now a challenge for many enterprises. Though the Workflow Management Coalition (WfMC) has already defined Interface 4 in its workflow reference architecture [18] to support inter-operation between workflow systems, the reality is that this type of support for collaboration is too tightly coupled. It is not suitable for today's ever-changing business and technology environment. In a dynamic business environment, an enterprise requires flexibility to select its partners dynamically, for example, outsource the same business activity to different parties according to different QoS levels provided and demanded. So, a tightly coupled collaboration system is not suitable for today's dynamic and flexible B2B collaboration any more.

This paper combines the enactment flexibility issues and inter-enterprise support abilities as a combined research goal. A service-oriented approach provides us with the possibility to reach this combined goal. In our approach, both remote applications and internal applications are transparent to the workflow definition process and treated as services. We extract activity specifications from workflow process specifications and use them as service requirements to find corresponding services that can implement the activities. Our service mediating workflow system architecture supports integration of both the internal services implemented in internal information systems and external services imported from business partners to implement the total business workflow. Such a system enables an enterprise to implement flexible outsourcing strategies by dynamically binding and invoking e-services provided by different service providers.

The remaining sections of this paper are organized as follows. Section 2 introduces related work and places our contribution in context. Section 3 presents an illustrative example. The conceptual framework and conceptual model of service mediating WFM are proposed in Section 4. Section 5 presents the extended WFM architecture. In Section 6, we describe the service mediating layer in terms of description of ES. Section 7 discusses two implementation methods for service invocation. Section 8 draws a conclusion.

2. Related work and our contribution

Our work combines aspects of electronic services and WFM. Hence, we discuss related work in both domains below. In the e-service field, we place our work in the context of the standards developed. In the WFM field, we discuss the research related to the two main workflow aspects of our work: flexibility and support for cross-organizational workflows. We end this paragraph by describing our contribution to the field.

2.1. E-services

The e-service model and service-oriented paradigm have brought innovative approaches to develop enterprise information systems. They build on the functionality of existing e-business frameworks and extend them with the aim of making the Internet more practical and powerful for businesses [5,13]. The e-services framework enables an application developer who has a specific need to cover it by using an appropriate e-service published on the Internet, rather than developing the related code from scratch [11].

Web services provide a service-oriented and component-based approach to implement application-to-application interaction. Web services are relatively simple and easy to implement and deploy. The Web Service Description Language (WSDL) [19] based on XML is used to describe Web service interfaces. SOAP [14] provides a lightweight message transport protocol to enable involved parties to exchange message in a Web service transaction. UDDI [15], a registering standard, is used in the Web service architecture to support service publishing and discovery.

Microsoft, IBM and BEA have created a language called Business Process Execution Language for Web Services (BPEL4WS) [4] for composing web services to support business process execution. BPEL4WS replaces the existing IBM WSFL [20] and Microsoft XLANG [21] efforts by combining and extending the functions of these previous languages. At the core of the BPEL4WS process model is the notion of peer-to-peer interaction between services

described in WSDL; both the process and its partners are modeled as WSDL services.

2.2. Workflow management

Flexibility has been addressed in the workflow community in a number of research efforts. Different types of flexibility are discussed in Ref. [9]. A typical example of a software development effort is the WASA project, in which a WFMS was designed that supports flexibility through mechanisms supporting on-the-fly changes to process specifications of workflow instances [16]. A different way of achieving flexibility in workflow process specifications is by distinguishing between core process specifications and exception specifications [6].

Commercial WFM products offer varying degrees of flexibility. For example, HP Process Manager offers resource rules that allow some flexibility with respect to application invocation, based on polymorphism of services [21]. In this approach, polymorphism relies on sets of functions that have the same invocation interface. The mechanism for resource selection uses query facilities of underlying database or directory systems.

Cross-organizational workflow management has been the topic of a number of research projects on the integration of process support and electronic business. Well-known examples in this field are WISE and CrossFlow. In the WISE approach [3], a global process specification is used to link activities or subprocesses of organizations participating in a virtual enterprise. In the CrossFlow approach [7], dynamic virtual enterprises are supported in a service outsourcing environment. Cross-organizational workflow processes are specified in electronic contracts between consumer and provider in a service outsourcing relationship. A comparable approach to supporting flexible service provision in virtual business communities is addressed in Ref. [12]. Different from our approach to extend an underlying WFMS to support dynamic service binding, this approach relies on building a basic service model and a service infrastructure to support flexible service composition among partners. The work described in Ref. [12] leaves many detail open, however.

2.3. Our contribution

Our major contribution is a service mediating workflow management framework, model and architecture based on realizing workflow enactment flexibility through separating activity ESs from workflow definition. The proposed ES binds activity specifications with service descriptions, which allows late selecting and binding of services at run time. Hence, different instances of the same activity can be fully dynamically bound to different implementations. This also allows the flexibility to bind an activity instance to an intra- or cross-organizational service, hence proving a basis for dynamic outsourcing decisions.

3. An illustrative example

In this section, an illustrative example is presented to show how service concepts can be integrated into a workflow management context to support flexible business process execution. Fig. 1 shows a diagram describing a business process that handles a purchase order. The business activities within this business process are represented as regular rectangle blocks. When a new purchase order is received, activity ‘Check Order’ is executed to validate the order. If the order is valid, the acknowledgement is sent. Then, in parallel, both activity ‘Check Repository’ and ‘Invoice’ are carried out. Activity ‘Check Payment’ is to check whether the payment has arrived. If the payment has arrived and goods supplies are sufficient to fulfill this order, activity ‘Delivery’ is enacted to deliver the goods. Activity ‘Check Repository’ is responsible for checking the amount of goods in the repository. Activity ‘Arrange Production’ is enacted when there are insufficient goods in the repository. After activity ‘Produce’ is carried out, the amount of goods will be sufficient and the activity ‘Delivery’ can be started. After activity ‘Delivery’ is completed, activity ‘Check

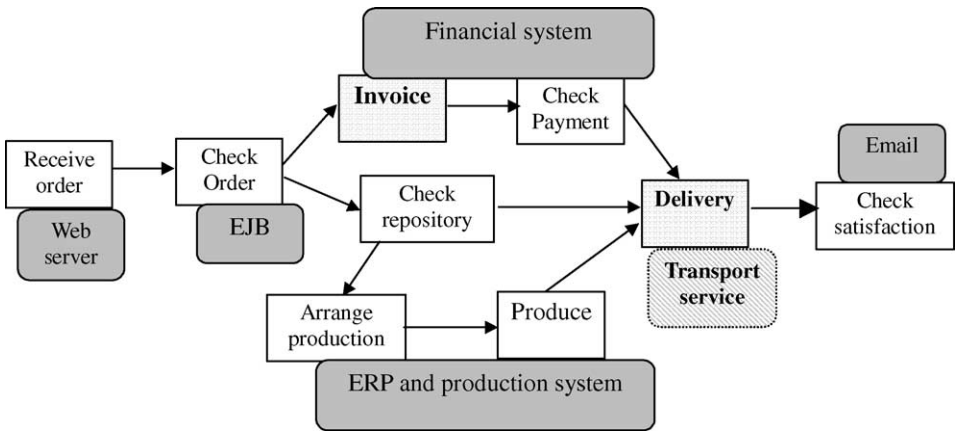

Fig. 1. Example workflow—activities and component deployment.

Satisfaction' is to check whether the customer is satisfied with the shipment.

To handle this process, the enterprise needs to deploy components or systems to support the WFMS to execute the activities. In Fig. 1, the components and systems that implement the corresponding activities are shown as rounded rectangle blocks. The 'blocks' can be wrapped legacy backend system, software components (based on COM, CORBA or EJB) or stand-alone systems (for example, an e-mail system). In traditional workflow management, the association of an activity and its implementation is established at build time. At run time, the workflow engine invokes the corresponding implementation to execute the activity instances. However, not all components that support the activity execution belong to the internal system. The enterprise may outsource some business activities to other companies. In this motivational example, activity 'Invoice' can be outsourced to a company that operates a billing system for external parties. Activity 'Delivery' is outsourced to transport companies, as the enterprise does not have its own transport system. So, the enterprise collaborates with transport companies to carry out this activity. From the view of activity outsourcing, the activity 'Delivery' is outsourced to the transport company to be executed there. However, from a service-oriented view, the enterprise imports the 'Transport Service' from the transport company and invokes it to execute activity 'Delivery'.

To achieve enactment flexibility, a flexible WFMS must allow dynamical deployment of a component to support activity execution. In this example, the email component through which an order is received can be replaced by a component that supports the SOAP protocol to receive the order sent in a SOAP message. Usually, in the business-to-business world, service contracts are signed to guarantee service provision, service availability and performance of the provided services. A large enterprise may establish many service contracts with its partners to obtain multiple external support facilities to execute a complex process. It is also possible that it makes different service contracts with different companies to get the same or similar service for an activity execution. For example, an enterprise may establish different service contracts for product delivery with road-transport companies, ship-transport companies and airlines to have multiple service options. Therefore, to reach the flexibility, the WFMS must have the ability to dynamically bind and invoke these contracted services and the internal services at workflow run time rather than at workflow build time.

4. Conceptual framework and model for service mediating WFM

Current workflow environments are characterized by being separated into two layers: the business process logic layer and the business application layer. This allows applications to support flexible business logic, for example changing the business activity sequences. However, the enactment flexibility we propose has not been reached by these WFMSs due to the fact that the dependencies of activities and their implementations are pre-established at build time rather than at run time. In this section, we propose an approach to separate the implementation concerns from a workflow process specification to loose the dependency, which allows dynamically binding a service or an application to an activity instance.

4.1. Three-layer conceptual framework

Our approach is to adopt a service mediating concept that dynamically bridges activity specifications and their implementations. Fig. 2 shows a three-layer conceptual framework. It includes a process definition layer, a service mediating layer and a service implementation layer. From the service-oriented viewpoint, the process definition is viewed as service requirements. It does not include any association to activity implementations. The service implementation layer corresponds to the business application layer in traditional workflow frameworks. Business applications (service implementations) are completely transparent to the process definition (service requirements), and vice versa. Hence, service requirements and service implementations can be dynamically associated through service mediating layer that dynamically binds activity specifications (service requirements) to the descriptions of the service implementations. The conceptual model as discussed below contains the details of our approach.

4.2. Conceptual model

Fig. 3 shows the ER model that describes service mediating concepts. According to the time of building the entities and establishing the relations among them, we separate the model into four levels: workflow build time

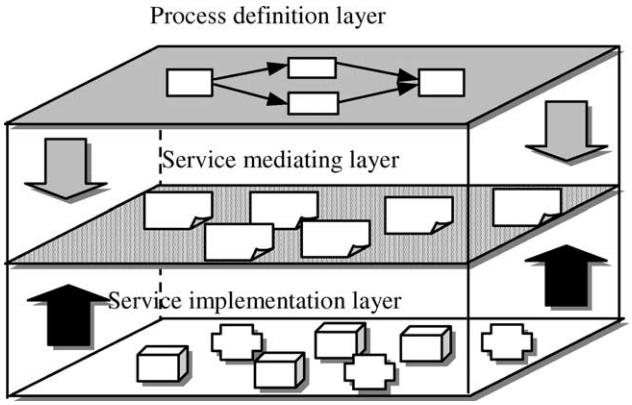

Fig. 2. Service mediating workflow management concept framework.

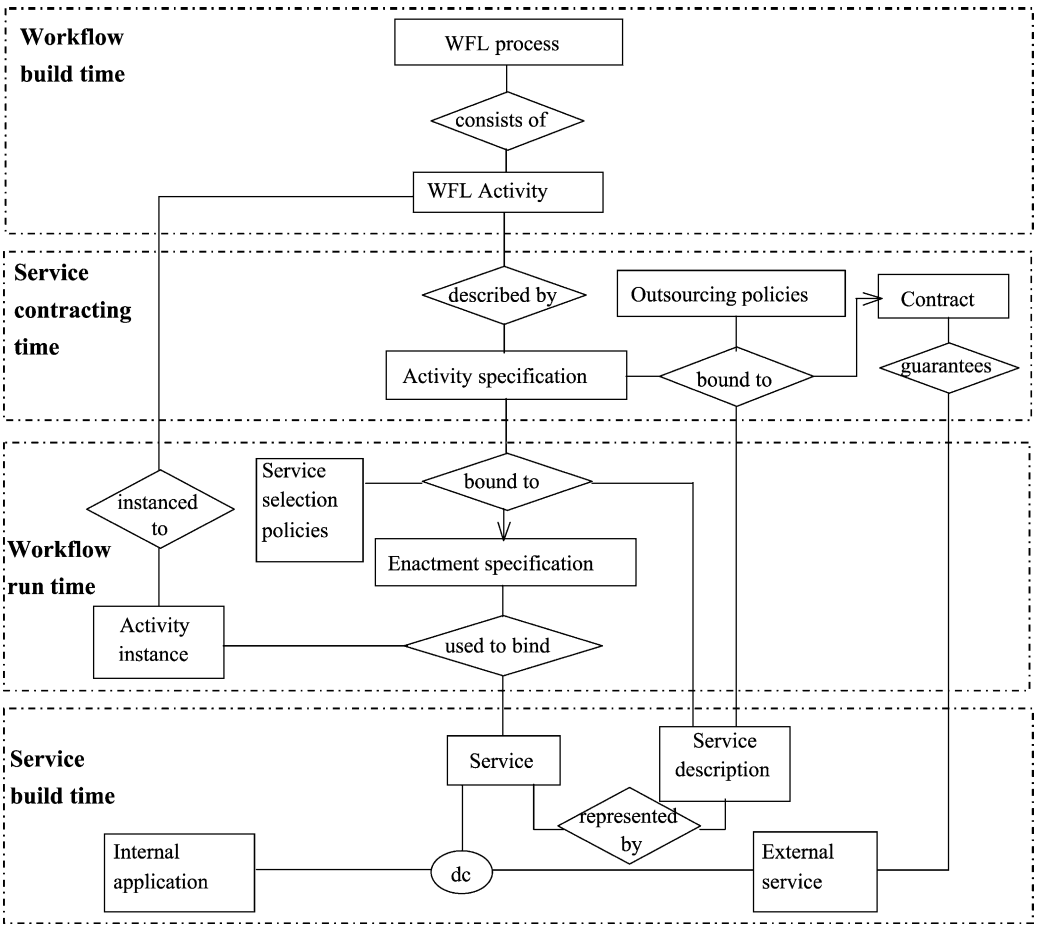

Fig. 3. Service mediating conceptual model.

level, service build time level, service contracting time level and workflow run time level.

At workflow build time, workflow processes that consist of workflow activities including workflow relevant data are specified and stored in a process specification.

At service build time, internal applications or external services are built as activity implementations. Corresponding service descriptions are created to describe the services. A service description of a standard service may be supported by different services that are provided by different service providers. This enables an enterprise to have flexibility to choose the most appropriate provider according to its outsourcing policies. To an enterprise, a service refers to an internal application or an external service that is guaranteed by a service contract. A contract specifies all relevant aspects (e.g. rights, obligations, service levels) of service provisioning [2].

At service contracting time, workflow activities with relevant data are used to generate activity specifications that are used for searching corresponding services. Through matchmaking between activity specification and implementation requirements and service descriptions, appropriate providers are selected according to certain business policies, for example, outsourcing policies. Through contracting, service contracts are established to guarantee the services provision and service qualities. Usually, a service description is specified in a contract and is part of service contract content. As an activity specification can be matched with several service descriptions provided by different providers, or even the same description supported by different services, several service contracts are likely created to guarantee one activity implementation, which allows flexible service selection at run time.

At workflow run time, selected service descriptions that satisfy the activity implementation requirements are bound to activity specification to dynamically generate an ES. According to such an ES, each instance of an activity can be dynamically bound to a corresponding service. Though an activity specification can be matched with multiple services, an instance of an activity can only be bound to one service according to the ES. Therefore, service selection policies are combined into the ES to support dynamic service selection according to the relevant data of the instances of the activity.

Fig. 3 shows that ‘Activity specification’, ‘Service description’ and ‘Enactment specification’ are the key elements to implement the service mediating concept.

5. Architecture for service mediating WFM

To support the proposed conceptual framework, we extend the architecture of current WFMSs. The extended architecture is shown in Fig. 4. It is divided into two parts: the left part is the simplified traditional workflow system architecture and the right part is the extension for supporting the proposed service mediating concept. Based on the service-oriented concept, the architecture is divided into five layers: workflow definition layer (and service definition layer), service contracting layer, service binding layer, service invocation layer and service layer. Compared to the conceptual model, the service binding layer in the extended architecture crosses the contracting time and workflow run time layers in the conceptual model.

5.1. Workflow definition layer

In the workflow definition layer, process and activities are built (defined) using workflow modeling tools. A workflow model interpreter interprets the workflow definition to extract the activity specifications. These will be used to search for external services, to match against existing internal applications or to build new applications. An activity specification mainly contains a number of activity description elements and workflow relevant data structure descriptions. It is similar to the interface description of an application or a service that can implement an activity, which makes it possible to use a matchmaking approach to find corresponding activity implementations: applications and services. The description of an activity specification is presented in Section 6.

5.2. Service contracting layer

The service contracting layer is used to find appropriate service descriptions and establish service contracts to guarantee service outsourcing. Outsourcing polices are used to determine which activities are to be outsourced. External services are to be found to support outsourced activities according to outsourcing policies. The service discovery and contracting system is integrated in this layer to discover external services that can meet the activity implementation specifications. Activity specifications are

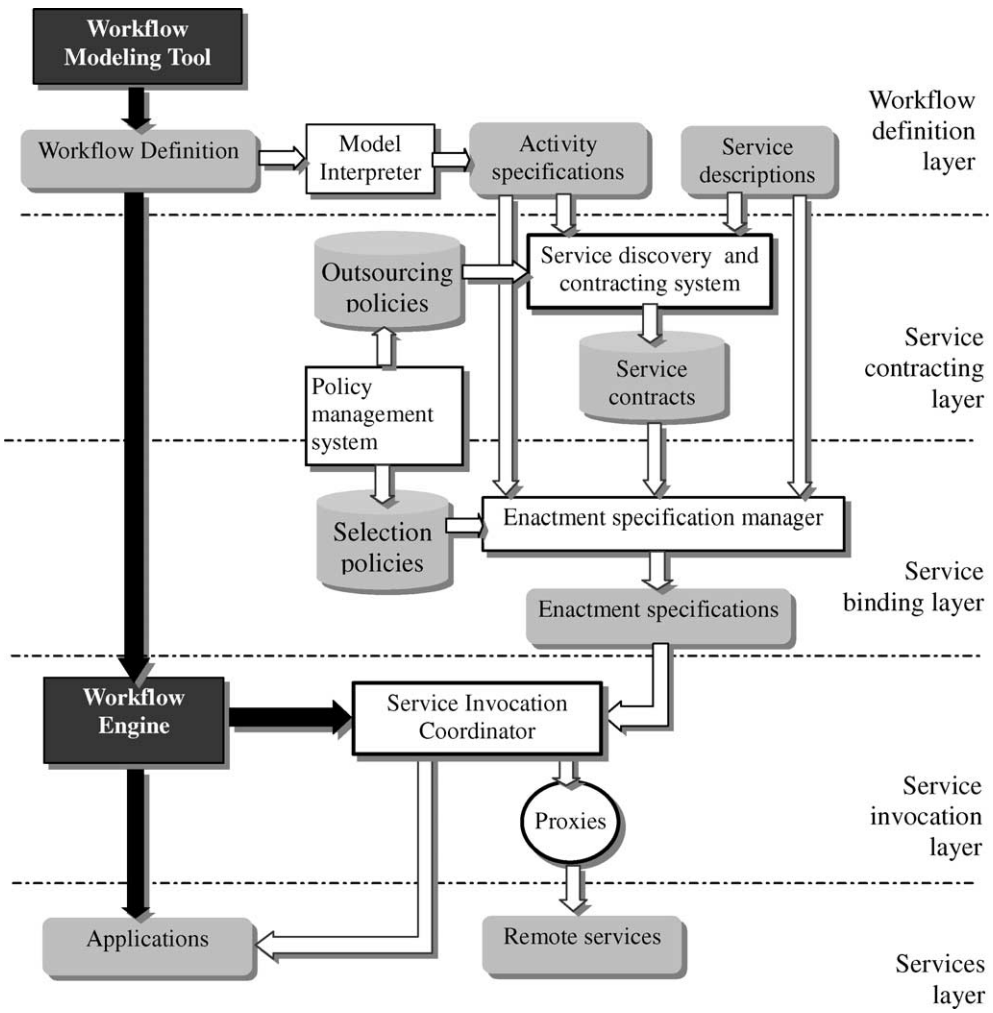

Fig. 4. Service mediating architecture.

used as service requirements to match with corresponding service descriptions. When suitable services are found through querying a service registry and matchmaking, the contracting system establishes service contracts with the service providers to guarantee the availability of the external services. Service descriptions are specified in the service contract and are part of the contract content. The service descriptions of contracted services are used to bind to activity implementation specifications to generate ESs. If no outsourcing strategy is adopted, existing or newly built internal applications that meet the activity implementation requirements will be invoked. Hence, service descriptions must describe both external and internal services. The details of service descriptions will be discussed in Section 6.

5.3. Service binding layer

As we have mentioned above, different providers can provide similar services to implement an activity, which could have the same or different service descriptions. An enterprise will subscribe to multiple services for the same activity implementation to attain flexibility. To realize this flexibility, a component called enactment specification manager is added. It works with the service selection policy management system at run time to dynamically select contracted services descriptions and bind them to the corresponding activity specifications to generate an ES. Service selection policies are combined into the ESs to be used for service invocation. Usually, an ES includes an activity specification, some service descriptions and some policies for service selecting, which will be described in Section 6. Consequently, in this layer, an activity specification is usually bound to one or multiple services. Since the relevant data of the activity instance is not known yet in this layer, the dynamical binding of a concrete instance with a concrete service is implemented in the service invocation layer.

5.4. Service invocation layer

The service invocation layer is an extension of the traditional workflow enactment system to support dynamic and flexible service binding and invocation. A Service Invocation Coordinator (SIC) is added to implement service invocation. When an activity is instantiated by the workflow engine, the invocation flow from the workflow engine is redirected to the SIC. The SIC binds the activity instances with the corresponding service according to the ES and activity instance information. According to the data mapping part described in the ES that we will discuss in the following section, the service coordinator converts the workflow relevant data to and from the data formats that the remote or local service supports. To invoke a remote service, the invocation messages with the relevant data are sent or received by a proxy. For example, a HTTP/SOAP proxy is used to exchange invocation messages and data with a remote HTTP/SOAP server that provides Web services. If the services are provided by an internal application, it will be invoked directly by SIC.

5.5. Service layer

The service layer is the layer that may cross organizations. Services as discussed here include both the internal applications that implement workflow activities and remote services provided by other parties to support the workflow activity executions. The internal applications can be information systems or software components obtained by wrapping legacy systems. Remote services are applications or components that are implemented by other parties and can be accessed based on specific messaging and transport protocols.

6. Descriptions for service mediating layer

As discussed above, activity implementation specifications, service descriptions and enactment specifications play important roles in implementing the service mediating concept. They altogether play the bridging role to connect workflow activity definitions (service requirements) to activity implementations (services).

6.1. Activity specification

The activity specification is generated from a workflow definition and is used for searching the activity implementations (which can be applications or services) (Fig. 5).

```
<ActivitySpec>
    <DataStructure Name=" Address" >
            <Parameter Name="Country" Type= "String" />
            < Parameter Name="City" Type= "String" />
            < Parameter Name="Street" Type= "String" />
            …
    </DataStructure>

    …

    <Activity Name="Delivery" />
        <Description> "This activity is implement product delivery" </Description>
        <InputData Name="DeliveryInput">
            <Para Name="Destination" Type="Address" />
            <Para Name="TransportMode" Type="String" />
            <Para Name="DeliveryDate" Type="Date" />
            <Para Name="SendAmounts" Type="Integer" />
        ….
        </InputData>
        <OutputData Name="Delivery Output>
            <Para Name="ArrivalDate" Type="Date" />
            <Para Name="ConfiredAmounts" Type="Integer" />
            <Para Name="Confirmation Type="String" />
        </ OutputData >

        <ImplementationReq>
            <Req> TransportMode="BY AIR" Or Mode="BY TRAIN" </Req>
            <Req> Protocol="HTTP/SOAP" OR Protocol="IIOP" </Req>
            ……
        </ImplementationReq>
    </Activity>
</ ActivitySpec >
```

Fig. 5. Activity implementation specification.

Usually, traditional workflow definitions include activity signatures and descriptions, data structures, programs or applications definitions, users and roles definitions, control flow and data flow. Elements of an activity specification include DataStructure, ActivityName, Input/output Data, which are extracted from workflow definition, and ImplReq (Implementation Requirement), which is an element added to describe the requirements of activity implementation. Implementation requirements may cover a very broad spectrum of aspects from technology requirements to business requirement. Generally speaking, they specify the required properties that a service should have. They are the major factors that should be considered when searching and contracting services.

6.2. Service description

According to the activity specification, the service discovery system searches for an appropriate service through querying published service descriptions in the service registry. Appropriate service descriptions that match the activity specifications will be the candidates for contracting. The basic service description describes the interface for accessing a service, which includes operations with input message, output message and transport protocols (or invocation protocol). A good example of a basic service description language is WSDL [19]. Advanced service descriptions can be based on WSDL and contain service endpoint properties description, such as the quality of service and the performance of service invocation. An example of this description language is the Web Services Endpoint Language (WSEL) [20]. In our approach, we use WSDL as service description language to describe services. A WSDL document that we call Web service description describes a set of endpoints to be accessed. Each WSDL document contains both an abstract definition of the service and the concrete bindings to particular network implementation and data format. Shown in Fig. 6, Type, Message, PortType are abstract definitions. Binding and Service are concrete definitions. Abstract Operations are parts of PortType. Operation implementations are parts of Binding. Ports are parts of Service. As we treat internal applications as services as well, the standard WSDL has to be extended to support internal service description. The current version of WSFL has already extended WSDL to support executable program, JAVA Class and other applications description, so we directly use them.

By comparing the elements of an activity specification and the elements of WSDL, we find that they have a similarity. An activity specification contains the abstract part (input and output message) of a service in WSDL and can specify the requirements for the concrete part: the binding protocol. The ES is used to dynamically bind these two.

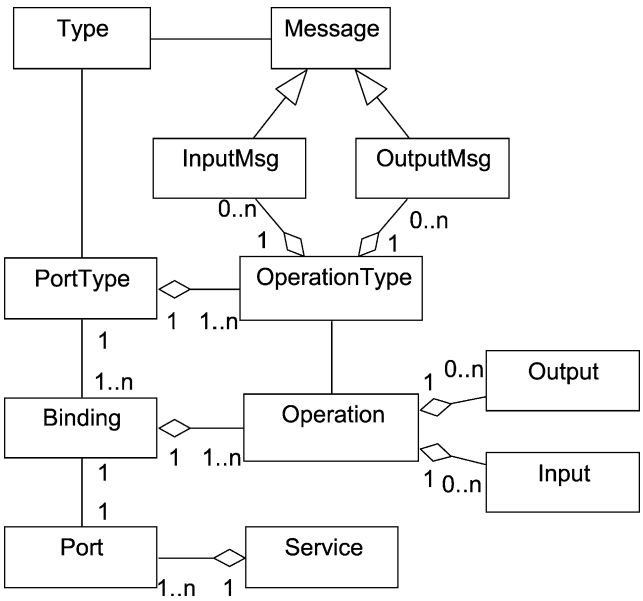

Fig. 6. Key elements and their relationships of WSDL document.

6.3. Enactment specification

An ES combines service description(s), service selection policies and an activity specification. It is interpreted by the SIC. According to the activity instance data, the SIC determines which service should be invoked to execute the activity. Generally speaking, an ES is a combination of an activity specification with one or multiple service descriptions. Besides, it addresses the following issues:

1. Activity specification aspect: the activity information and its input/output data structures, which are the major parts of activity implementation specification presented in Section 6.1.
2. Services and service descriptions, service provider information: information is described as a WSDL document.
3. Proxy information: the general implementation information of the corresponding proxy to the specified service, only for accessing remote service. This information indicates how the SIC can dynamically invoke the proxy program and pass message to it. The proxy program is viewed as an internal application.
4. Service selection policies: the policies to determine which service binds to which activity instance when multiple services are available. It can be changed from time to time to realize flexible binding.
5. Data mapping: mapping activity input data and output data with input message and output message of a service.
6. Performance control aspect: for example, monitoring aspects specified in a contract.

Fig. 7 shows an example to illustrate the overall structure and major elements of an ES. The service description in WSDL in this example has been omitted to keep it brief and to put emphasis on our extension. The ES we show in

```
<EnactmentSpec>
    <!-- Here import  service description in WSDL which defines Types,
    Message portTypes, Binding, Ports (operations) and Services -- >
    <!-- The following section describe activity specification-->
    <Activity Name="Delivery" />
        …
        <InputData Name="DeliveryInput">
            <Para Name="Destination" Type="Address" />
            …
        </InputData>
        <OutputData Name="DeliveryOutput>
            <Para Name="ArrivalDate" Type="Date" />
            …
        </ OutputData >

    <!--  The following section describes the implementation options and data mapping between activity
input/output data and input/output messages -->

        <Implementation>
            <ServiceOptions>
                <Option Service="NS_transport_service" port=" …"  operation="… ">
                    <DataMap>
                        <InputMap SourceData="DeliveryInput"  Para="Destination"
                                TargetMsg="NS_TransportInput" Part="Place" />
                        …
                        <OutputMap SourceMsg="TransportConfirm"  Part="ReachDate"
                                TargetData="DeliveryOutput" Para="ArrivalDate" />
                        …
                    <Proxy  Name="SOAPProcy" protocol ="HTTP/SOAP">
                        <Java method="Service.Transport.Proxy.HttpSoapCall" >
                    </Proxy>
                </Option>
                <Option Service="KLM_transport_service" port="… "  operation=". .." >
                    …
                </Option>
                <Option Service="Internal_Transport_service" Port="…" operation="…"
                    …
                </Option>
            </ServiceOptions>
        </Implementation>
        <!-- The following section describes selection policies -->
        <Selection>
            <policy Name="DestinationRule">
                <IF Delivery. Input. Destination. Country <> "the Netherlands" />
                <Bind  "KLM_transport" />
                <ELSE Bind  "NS_transport" />
                    <IF Delivery .Input. Destination. city = "Enschede" />
                    < Bind  "Internal_transport" />
                    <ELSE Bind  "Internal_Transport_Service" />
                    </ENDIF>
                </ENDIF>
            </policy>
            …
        </Selection>
    </Activity>
</ EnactmentSpec >
```

Fig. 7. Enactment specification.

the example is for enacting activity 'Delivery' as introduced in Section 3. Descriptions of transport services by local delivery system by railway system or by airlines are attached to this activity. Data mapping between services and the activity specification are specified. General proxy implementation information is given to help the service coordination invoke it dynamically. To support flexible service selection, selection policies are combined. According to the policy, different instances of the same activity can be bound to different services. In this example, 'Destination' of the 'Delivery' activity instance determines which service is bound: if the Destination is outside 'the Netherlands' then, transportation is operated by air ('KLM_transport'). If the Destination is inside the city 'Enschede', the enterprise uses its own transport system ('Internal_transport'). Otherwise, the transportation is operated by train ('NS_transport').

The ES has some similarities to the 'Flow Model' of WSFL. However, WSFL puts emphasis on describing

the flow rather than activity implementation as we do here. Besides, we allow multiple service options to attach to an activity implementation and adding service selection policies to support dynamical selection, which are the distinguishing differences between our approach and WSFL [20].

7. Implementation of service invocation

In our architecture, the associations between the activities and its implementations are described in the ES but not in the workflow definition. The ES binds service description and activity implementation specification. It is completely separated from the workflow definition and can be created or changed at run time. So the workflow engine does not know anything about this enactment specification and service description. The obvious question is how the workflow engine can invoke the corresponding applications or services to enact the activities since there are no associations between activities and their implementation in the workflow definition. There are two ways to bind the activities with their implementation at run time through the SIC.

7.1. PUSH method

In the first method, all activities that are not bound to fixed applications at build time can be statically bound to the SIC. As a result, the SIC is indicated as the executor of these activities at workflow build time. In this case, the SIC is viewed as a workflow application that can be invoked by the workflow engine to carry out these activities when the process is executed. Associations between these activities and SIC are established at workflow build time. At run time, the activity instances are pushed to the SIC by the workflow engine. The coordinator will invoke the corresponding applications locally or remotely according to the ES. We call this way the PUSH method, which is a reactive method since the SIC does not have any initiative by itself. By using this method, the applications and services are dynamically bound to the activities through the coordinator according to the ES when the activity instance information is sent to the coordinator.

7.2. PULL method

Another method is called PULL method. The invocation service coordinator actively pulls the corresponding activity instance and relevant data according to the ES when an activity is instanced by workflow engine. Hence, no association between an activity and the SICs is established in the workflow definition. Since no implicit and explicit associations are established at workflow build time, the SIC has to actively monitor the process execution and take over the workflow engine to execute the instanced activity if there is an ES that specifies this activity implementation.

7.3. Comparison

To support the PULL method, the implementation of the SIC module is more complex than to support the PUSH method because the coordinator must monitor the state of workflow instances and 'pull' the workflow instances and activity instance information actively. For the PUSH method, the workflow engine pushes the workflow instance information to the coordinator directly. In that case, the SIC is just a normal application to the workflow engine. Which method is to be chosen depends on the WFMS that is used. To support the PUSH method, the WFMS should support passing the workflow relevant data to the invoked application. To support the PULL method, the WFMS should provide an API to monitor workflow instances. Implementation of the PULL method is relatively complicated. However, the limitation of the PUSH method is that you have to specify the data to be transferred between workflow engine and the SIC as SIC program parameters at workflow build time, which cannot be changed at run time anymore. Hence, the PULL approach can support a more dynamic mechanism—deferring more aspects to run time.

8. Conclusions

This paper presents a service mediating approach to realize workflow enactment flexibility. Our service mediating approach can be viewed as an equivalent of service broking technology development in software engineering domain, especially distributed software system engineering, such as CORBA technology.

The service mediating WFM framework proposed in this paper separates workflow activity definition from its implementation and uses a service mediating layer to bridge them. Thus, A WFMS has more flexibility to select the most appropriate services at run time to execute the activity instances. We have extended the traditional workflow system architecture to implement our service mediating concept. Since different services can be bound to different instances of the same activity, the extended system can well support flexible service outsourcing.

We end this paper with a few thoughts on the practical applicability of our approach. One may have questions with respect to the feasibility of dynamic binding in practice. For example, CORBA allows for advanced dynamic binding strategies, but they have not been used much in practice. We believe that Web service environments both facilitate and require more dynamism in binding than 'traditional' distributed software engineering. Dynamic binding is facilitated because e-service markets will provide many implementations of compatible services, with similar functionality but with different quality of service

characteristics and with different pricing. Dynamic binding is required to match the dynamism of e-business markets—choice of business partners has to be made on-the-fly to profit best from market conditions. Current development of cross-organizational workflow management in e-service contexts will provide the basis of complex, inter-organizational control flow handling [8] and advanced, compensation-based transaction management [17], thereby providing facilities not offered in typical software engineering environments.

Acknowledgements

We thank Jochem Vonk for his work on setting up the MQSeries Workflow system and assistance with this software and proofreading this paper. We also thank Samuil Angelov for discussing with us on service contract and electronic contracting.

References

[1] A. Agostini, G. de Michelis, Improving flexibility of workflow management systems, in: W. van der Aalst, J. Desel, A. Oberweis (Eds.), Business Process Management, LNCS 1806 (2000) 218–234.
[2] S. Angelov, P. Grefen, A conceptual framework for B2B electronic contracting, Proceedings of the Third IFIP Working Conference on Infrastructures for Virtual Enterprises, Sesimbra, Portugal (2002) 143–150.
[3] G. Alonso, U. Fiedler, C. Hagen, A. Lazcano, H. Schuldt, N. Weiler, WISE: business to business e-commerce, Proceedings of the Ninth International Workshop on Research Issues on Data Engineering, Sydney, Australia (1999) 132–139.
[4] BPEL4WS, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, 2002
[5] F. Casati, M.C. Shan, Dynamic and adaptive composition of e-services, Information Systems 26 (3) (2001) 143–163.
[6] F. Casati, S. Ceri, S. Paraboschi, G. Pozzi, Specification and implementation of exceptions in workflow management systems, ACM Transactions on Database Systems 24 (3) (1999) 405–451.
[7] P. Grefen, K. Aberer, Y. Hoffner, H. Ludwig, CrossFlow: cross-organizational workflow management in dynamic virtual enterprises, International Journal of Computer System Science and Engineering 15 (5) (2000) 277–290.
[8] P. Grefen, H. Ludwig, S. Angelov, A Framework for E-Services: A Three-level Approach towards Process and Data Management, IBM Research Report RC22378, IBM Research Division, 2002.
[9] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, M. Teschke, A comprehensive approach to flexibility in workflow management systems, Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration, San Francisco, USA (1999) 79–88.
[10] IBM MQSeries Workflow—Getting Started with Buildtime, Version 3.2.1, IBM Corporation, 1999.
[11] IBM Web Services Architecture Team, Web Services Architecture Overview—The Next Stage of Evolution for e-Business, IBM Corporation, 2000, http://www-106.ibm.com/developerswork/web/library/w-ovr/
[12] A. Marton, G. Piccinelli, C. Turfin, Service provision and composition in virtual business communities, Proceedings of the Workshop on Electronic Commerce (at ISRDS-18), Lausanne, Switzerland (1999) 336–341.
[13] T. Pilioura, A. Tsalgatidou, E-services: current technology and open issues, Proceedings of the Second International Workshop on Technologies for E-Services, Rome, Italy (2001) 1–15.
[14] Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/SOAP
[15] UDDI, Universal Description, Discovery and Integration, http://www.uddi.org
[16] G. Vossen, M. Weske, The WASA2 object-oriented workflow management system, Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, Philadelphia, USA (1999) 587–589.
[17] J. Vonk, P. Grefen, Cross-organizational transaction supported for e-services in virtual enterprises, Journal of Distributed and Parallel Databases (2003) Kluwer Academic Publishers, in press.
[18] WfMC Workflow Reference Model, WFMC-TC00-1003 (issue 1.1), Workflow Management Coalition, 1995.
[19] Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl
[20] Web Services Flow Language, Version 1.0, http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
[21] HP Process Manager White Paper: Technical Architecture, http://www.hp.com/go/e-process, Hewlett-Packard, 2001

Further Reading

[22] XLANG, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm