Workflow Semantic Description for Inter-organizational Cooperation

Nomane Ould Ahmed M'Bareck and Samir Tata

GET/INT, 9 rue Charles Fourier, 91011 Evry, France

Abstract. The work we present here is in line with a novel approach for inter-organizational workflow cooperation that consists of workflow advertisement, interconnection, and cooperation. For advertisement, workflows should be described. Nevertheless, by using a description language like *XPDL* only syntactic problems can be solved. In this paper, we propose a three steps method for workflows semantic description. First, workflows described using *XPDL*, are annotated to distinguish cooperative activities and non cooperative ones. Second, to preserve privacy, a view, that we call cooperative interface, for each different partner is generated. Third, cooperative interfaces are described using *OWL* according to an ontology we have defined for cooperative workflows.

1 Introduction

In context of globalization, organizations are increasingly using process-aware information systems to perform automatically their business processes. Based on such systems, organizations focus on their core competencies and access other competencies through cooperation, moving towards a new form of network known as virtual organization.

To support cooperation, one has to deal with issues such as automatic workflow discovery, established workflow preservation, and privacy respect. In fact, cooperation needs a certain degree of inter-visibility in order to perform interactions and data exchange. Nevertheless, cooperation may be employed as a cover to internalize the know-how of partners. In order to preserve privacy and autonomy, one must reduce workflow inter-visibility to be as tiny as the cooperation needs.

For inter-organizational workflow cooperation, two main families of approaches are developed: top-down and bottom-up. Within the top-down family, several approaches were defined [2]. We can cite among others: capacity sharing, subcontracting, case transfer, loosely coupled, and public-to-private approaches. The most promising approach is the public-to-private one [1] that consists of three steps. First, the organizations involved agree on a common public workflow, which serves as a contract between these organizations. Second, each task of the public workflow is mapped onto one of the domains (*i.e.*, organization). Each domain is responsible for a part of the public workflow, referred to as its public part. Third, each domain makes use of its autonomy to create a private workflow, using some inheritance rules. Problems to be encountered with this family of approaches include mainly autonomy of local workflow processing, confidentiality that prevents complete view of local workflow [3], and especially flexibility that needs no definition of a global workflow that defines cooperation between local workflows. In addition, a drawback is the lack of the preservation of pre-established workflows. In fact, in this approach, one has to look for which rules, in what order and how many times one has to apply them in order to match the pre-established workflow with the public part which is deduced from partitioning of the public workflow. If not impossible, this is hard to do. Moreover, there is no defined procedure to do that.

Within the bottom-up family, we have developed the *CoopFlow* [4, 5] approach inspired by the Service-oriented Architecture that requires three operations: publish, find, and bind. Service providers publish services to a service broker. Service requesters find required services using a service broker and bind to them. Accordingly, our approach consists of three steps: workflow advertisement, interconnection, and cooperation. To be advertised, workflow must be described. Nevertheless, languages for workflow description lack semantics, which holds up the issues of automatic discovery of workflows. In line with our approach, we propose here a three steps method for workflows semantic description.

The rest of this paper is organized as follows. Section 2 discusses related work in the area of workflow description. Section 3 summarizes our bottom-up approach to inter-organizational workflow cooperation. Section 4 is devoted to semantic description of workflows. Conclusion and perspectives are presented in Section 5.

2 Related Work

To describe inter-organizational workflows, big efforts have been made and many languages have been proposed. In the following we present a survey of these propositions.

Business Process Execution Language for Web Services (*BPEL*) [7] is a language for specifying business processes behavior based on Web services and business interaction protocols. *BPEL* process allows the definition of abstract and executable processes. It does not support many concepts that are paramount for inter-organizational cooperation. In fact, it does not profit of the rich concepts of exiting workflow management systems as the notion of manual activities, applications, nor addresses the integration with them, since it uses Web services exclusively which represent a limit to call other types of services (i.e. activities).

XML Process Definition Language (*XPDL*) [8] was proposed and standardized by Workflow Management Coalition. Its principal entities are *Workflow Process*, *Activity*, *Transition*, *Application*, and *Participant*. Certainly the description provided by *XPDL* is rich but it cannot be published to a registry especially for two reasons. The first is that the description provides too many information and consequently it doesn't preserve the privacy of partners' workflows. The second reason is that the description is syntactic and so it doesn't allow automatic discovery.

OWL-S [10] is an ontology and a language based on *OWL* [9] and dedicated to describe semantically Web services. It was developed to achieve automatic Web service discovery, invocation, composition and inter-operation. *OWL-S* ontology is organized in three modules: *ServiceProfile*, *ServiceModel*, and *ServiceGrounding*. *ServiceProfile* describes *"what the service does"*; it specifies inputs and preconditions required by the service as well outputs and effects. Also, other important attributes for the discovery are specified as temporal and geographical constraints. *ServiceModel* describes *"how the service works"*. It describes how to interact with the service. In other term, it describe the flow of control of services. *ServiceGrounding* shows how an agent can access the service. *OWL-S* is not adapted to workflow description. Indeed, the information necessary for discovery is divided between the *Service Profile* and the *Service Model*. Consequently, in order to discover a workflow described with *OWL-S* both the *Service Profile* and the *Service Model* should be published and the algorithm of matching must jump between them to decide if two workflows match or not. Moreover *OWL-S* does not dictate any constraint between *Service Profile* and the *Service Model*. Consequently, there is no mean to identify inconsistencies between them.

WSMO [12] is an ontology for describing various aspects related to semantic Web services. It is intended to achieve automatic Web services discovery, invocation, composition and inter-operation. *WSMO* is based on four concepts: *ontologies*, *web services*, *goals* and *mediators*. *Ontologies* provide machine-readable semantics for *Web services*, *Goals* and *Mediators*. *Goal* specifies objectives that a client might have when consulting a *Web services*. *Mediator* allows to link heterogeneous resources and resolves incompatibilities that arise at different levels. The *WSML* language [13] is used to formalize *WSMO*. The use of *WSML* for describing *WSMO* ontologies is its main drawback. In fact, there are not yet tools to create, parse or manipulate these ontologies. Finally the jargon used by *OWL-S* and *WSMO* is not meaningful for workflows.

3 *CoopFlow*: An Approach for Workflow Cooperation

To support virtual organizations, we have developed a novel approach that consists of three steps: workflow advertisement, interconnection, and cooperation. In this section we present this approach using a running example.

3.1 Example

Consider an example involving a client and a product provider. Figure 1-(a) presents the client's workflow using Petri nets [11]. First, the client sends an order for a product. Then she receives a notification. When the product is ready, she receives the delivery and the invoice. Finally, she pays for the ordered product. Figure 1-(c) presents the provider's workflow. First, the provider waits for an order request. Then he notifies the client that his/her order was taken into account and he assembles the components of the product. After that, two cases can happen: the client is a subscriber (s/he often orders products) or s/he is not. In the first case, the provider sends the product and the invoice and waits for the payment. In the second case, the provider sends the invoice, waits for the payment and then sends the product. Cooperative activities, represented by filled transitions in Figure 1, are the ones that send and/or receive data to/from external partners. The examples given here only show cooperation between two partners. Nevertheless, our work also addresses cooperation between more than two partners.

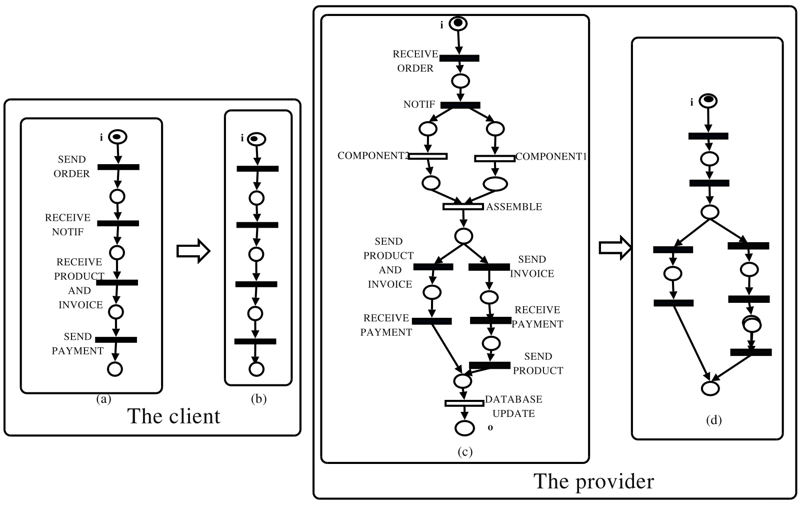

Fig. 1. The client and the provider workflows and their cooperative interfaces.

3.2 Workflow Advertisement

To set up cooperation, each partner has to advertise its offered and required activities within their workflows, control flow, and data flow, that we call cooperative interface. Broadly speaking, a workflow cooperative interface is a projection of a workflow on the cooperative activities that interact with one partner type [4]. Figure 1-(b) presents the workflow cooperative interface of the client and Figure 1-(d) presents the provider's one. Semantic description of cooperative interfaces are published to a registry (i.e. the control flow, the data flow and a semantic description of cooperation activities).

3.3 Workflow Interconnection

Partners loking for organizations with complementary skills can make use of cooperative interfaces they published. In order to construct an inter-organizational workflow, we have to match cooperative interfaces. Matchmaking takes into account the flow of control, the data flow and semantic descriptions of cooperation activities. Given two cooperative interfaces, the matchmaking result can be (1) positive (*i.e.* interfaces match) (2) negative (*i.e.* interfaces do not match at all) or (3) conditional (*i.e.* interfaces match if a given condition holds). If the matchmaking result is not negative, the cooperative interfaces are then interconnected. In our example presented above, the matchmaking result is conditional: the client cooperative interface (see Figure 1-(b)) and the provider cooperative interface (see Figure 1-(d)) match if the client is a subscriber. The result of this step (*i.e.* workflow interconnection) is an inter-organization workflow, and a set of cooperation policies that define cooperative partners and their interfaces (*i.e.* cooperative activities, their order of execution,. . .) and constraints on workflow interactions (*e.g.* the matchmaking condition).

3.4 Workflow Cooperation and Monitoring

The third step within our approach for workflow cooperation consists in the inter-organizational workflow deployment and execution. To do that, we have developed the *CoopFlow* framework [5] that allows different WfMSs to interconnect and cooperate their workflows. In addition to cooperation, this framework mainly enforces cooperation policies identified during the workflow interconnection step.

4 *OWL4W*: OWL for Workflow Description

One of our objectives is to allow partners to automatically discover each other. Therefor, we focus, in this section, on identifying important information to be described to allow inter-organizational workflow cooperation and how this information is described using *OWL*-based workflow description language.

4.1 Identification of Important Information for Cooperation

Our goal is to describe workflows in such manner that allows, for a given need, automatic and dynamic discovery. Let us, for example, suppose that a company of mobile phones with SIM card wishes to look for suppliers. Complex requests can be formulated as *"look for a supplier located in France, able to treat an ordering of mobile phone with SIM card and return me a notification before September 20, 2005"*. Currently, this task must be performed by human. To be able to deal with such request, it is not always necessary to advertise all workflow activities to describe the provided service.

To identify information to be described, we considered two main aspects. First of all, we propose to hide non co-operative activities to preserve privacy since they are only defined to achieve internal tasks while cooperative activities interact with the external partners. Secondly, even if the description of workflow is intended to be computer-interpretable, it should be understandable by person who isn't specialist. That enables to have a high level abstraction of the workflow and as consequence allows the workflow to be discovered by a greater number of users. To meet this end, in our description we give only the control flow (represented by activities and their *transitions*), the data flow (represented by Input/Output) and some additional attributes like deadline of activities, quality and location of provided service... We do not describe information concerning implementation of activities, resources nor activities' execution mode.

4.2 Description of Identified Information

Several languages were proposed to describe workflows. Among those we can cite *XPDL*. The description that we propose consists in starting from a first internal *XPDL* workflow description and generate from it a semantic description of cooperative interfaces. The choice of *XPDL* as a starting point can be motivated by the fact that it contains all information that somebody can wish to know on a workflow. We find there description of data flow, control flow and also a description of the resources. In the following we present how we proceed to generate a semantic description from *XPDL* one.

This is done in three steps: annotation of *XPDL* description, abstraction of interfaces, and semantic description.

The first step consists in annotating the initial description of workflow by adding for each cooperative activity information about the partner (formal parameter) with which it will interact. Thanks to annotations, the second step generates a set of interfaces for each workflow. The Figure 1-(d) shows the interface generated for the workflow shown in Figure 1-(c). Finally, the third step describes each interface using *OWL*. The partial conversion from XPDL to OWL description that is out of this article's scope can be done automatically. In the following we give the *XPDL* description of the example given in figure 1-(c) and we use this example to illustrate these steps.

4.3 XPDL Description of the Example

Since descriptions of all the activities of the example are broadly the same, we give here the description of the *NOTIF* activity (*i.e.* the provider notifies the client that his/her order was taken into account). We suppose that we have an application called *Email* which implement the activity *NOTIF*. *Email* has two inputs, *orderNum* and *EmailAdss*, and one output *Message* which is the notification sent to the partner. *EmailAdss* is extracted from the order and the *orderNum* is generated by the activity *RECEIVE ORDER*.

```
<Activity Id="2" Name="NOTIF">
 <Implementation>
  <Tool Id="Email" Type="APPLICATION">
   <ActualParameters>
    <ActualParameter>orderNum</ActualParameter>
    <ActualParameter>EmailAdss</ActualParameter>
    <ActualParameter>Message</ActualParameter>
   </ActualParameters>
  </Tool>
 </Implementation>
 <TransitionRestrictions>
  <TransitionRestriction>
   <Split Type="AND">
    <TransitionRefs>
     <TransitionRef Id="notif2compo1"/>
     <TransitionRef Id="notif2compo2"/>
    </TransitionRefs>
   </Split>
  </TransitionRestriction>
 </TransitionRestrictions>
 <StartMode>Automatic</StartMode>
 <FinishMode>Automatic</FinishMode>
 <Performer></Performer>
</Activity>
```

4.4 Step 1: XPDL Description Annotation

To distinguish between non cooperative and cooperative activities that interact with external partners on one hand and between cooperative activities that interact with different partners on the other hand, this first level of description was introduced. For this end we add an attribute *Partner* which is specified for each cooperative activity and which is defined by the means of the element *Extended Attribute*. Cooperative activities of a workflow which will cooperate with the same partner (those which will belong to the same interface) will have the same name. This acts like a formal parameter. The value of this element (the identity of the client and if s/he is a subscriber or not) will be provided only at the interconnection (cf. Section 3.3) step within the *CoopFlow* approach.

```
<ExtendedAttribute Name="Client" Value="" />
```

In our example we quote that the following activities are cooperative: *RECEIVE ORDER*, *NOTIF*, *SEND INVOICE*, *RCEIVE PAYMENT*, *SEND PRODUCT*, *SEND PRODUCT AND INVOICE* and *RECEIVE PAYMENT*. The others are not. The value of *name* for *NOTIF* cooperative activities is *Client*. We give in the following the description of this activity. The description of the others can be easily deduced.

```
<Activity Id="2" Name="NOTIF">
 <Implementation><!-same as above-></Implementation>
 <TransitionRestrictions>
   <!-same as above->
 </TransitionRestrictions>
 <StartMode><!-same as above-></StartMode>
 <FinishMode><!-same as above-></FinishMode>
 <Performer></Performer>
 <ExtendedAttribut Name="Client" Value="" />
</Activity>
```

4.5 Step 2: Interfaces Abstraction

The realization of a consequent task requires the implication and the cooperation of a set of organizations. However, this cooperation should not lead to the disclosure of the know-how of each organization. In order to preserve this know-how we abstract each workflow by a set of interfaces and those interfaces will be advertised. Each interface consists of a set of activities which will cooperate with the same partner. Abstraction is done using an original technique based on the exploitation of linear invariants in Petri nets [6]. This abstraction became possible thanks to the annotation done in step 1.

Figure 1-(d) shows the abstract interface of the workflow in figure 1-(c). Now we give the description of the interface shown in figure 1-(d). We can observe that the activity *NOTIF* is not connected any more to the activities *COMPONENET 1* and *COMPONENT 2* but it is now connected to *SEND INVOICE* and *SEND INVOICE AND PRODUCT* which are cooperative. The impact of this modification is that the transition restriction of the *NOTIF* will change. The type of restriction was *AND* and it becomes *XOR* because the restriction type of *ASSEMBLE* activity was *XOR*. Since this later will be hidden then the restriction on the transition between *NOTIF* activity on the one hand and *SEND INVOICE* and *SEND INVOICE AND PRODUCT* activities on the other hand will be *XOR*. We give in the following the description of this *NOTIF* activity. The description of the others can be easily deduced.

```
<Activity Id="2" Name="NOTIF">
  <Implementation>
   <Tool Id="Email" Type="APPLICATION">
    <ActualParameters>
     <ActualParameter>orderNum</ActualParameter>
     <ActualParameter>EmailAdss</ActualParameter>
     <ActualParameter>Message</ActualParameter>
    </ActualParameters>
   </Tool>
  </Implementation>
  <TransitionRestrictions>
    <TransitionRestriction>
     <Split Type="XOR">
      <TransitionRefs>
       <TransitionRef Id="notif2SenInv"/>
       <TransitionRef Id="notif2SenInvAndProd"/>
      </TransitionRefs>
     </Split>
    </TransitionRestriction>
  </TransitionRestrictions>
</Activity>
```

4.6 Step 3: Semantic Description with OWL

The description we present here consists in defining ontology for the co-operative workflows. This ontology aims at describing, in a non ambiguous way, the cooperative workflows so that a software agent can automatically exploit information on those. Benefits of the use of an ontology are known and multiple, but the first which can come to mind relates to the improvement of the discovery of workflows' interfaces and their interconnection to construct an inter-organizational workflow. This ontology endeavors to describe workflows in order to be able to carry out the automatic workflow discovery. The ontology we propose here is the semantic description of the interface obtained in step 2. It thus describes, in *OWL*, the data flow, the control flow and a set of additional attributes for workflows. Also it gives a textual description of the provided service and the name and the address of the company which holds the workflow.

The ontology we give here enables the semantic description of abstract interfaces obtained in the step 2. We point out that an interface is a workflow. The upper concept in this ontology is the *Interface* (see Figure 2). The *Interface* is composed of a set of activities. Each activity has a set of transitions (*transition*), a set of parameters (*parameter*), a deadline and possibly some restrictions (*restriction*) on transitions. An activity can be located at a well specified place and it could have attribute which specify the quality of provided service. Each parameter has a type and an order. The *order* makes it possible to order inputs and outputs when this is necessary and it is used primarily by the matchmaking algorithm. For example, let us consider that we have two activities *A1* and *B1* and each one has one input and one output. *A1* receives a product and then sends the payment while *B1* receives the payment and then it sends the product. Even if inputs and outputs of these two activities refer to the same concepts of an ontology, these two activities don't match because the sending and reception order of the parameters is not the same. To deal with this issue, we added the attribute *order*. To express the deadline of the activity we define the class *Deadline* as being an instance of the class *TemporalThings* of the *Time* ontology [1] employed in *OWL-S*. Each *transition* has a starting activity (*TransTo*), an arrival activity (*TransFrom*) and also it has a *condition*. We represent a *condition* as logical Formula. The concept *Region* refers to an ontology containing the areas of the world. The *quality* of services provided by the activity should be specified by a third entity. Finally, there are four kinds of *restrictions* an activity could have on transitions: *SplitXOR*, *SplitAND*, *JoinXOR*, and *JoinAND*. A *Split* specifies that the incoming transitions of the activity are split. A *Join* specifies that the outgoing transitions of the activity are joined. The type of both *Split* and *Join* could be *AND* or *XOR*. *AND* means that the transition are executed in parallel. *XOR* means that one transition is executed.

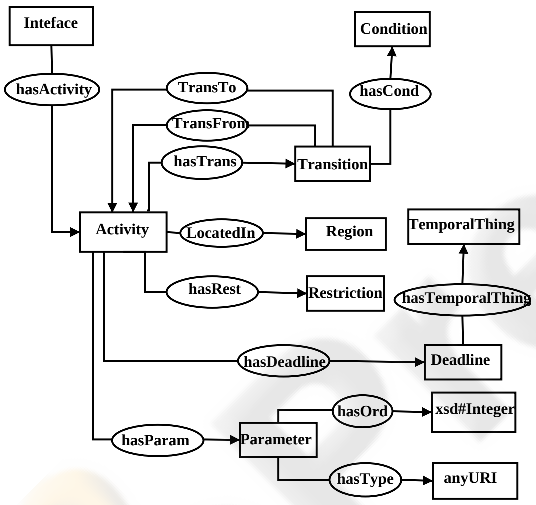

Fig. 2. OWL4W: classes and properties.

Now we give the description of the *NOTIF* activity given in example. Firstly we define the type of input/output used by the activity. Each type refers to a concept in an ontology. Then, we give the definition of the *NOTIF* activity and one of its transitions.

```
<Input rdf:ID="orderNum">
 <hasType
  rdf:resource="http://www.w3.org/2001/XMLSchema#Integer"/>
</Input>
<Input rdf:ID="Email">
 <hasType
  rdf:resource="http://www.bpOnto.org/order#EmailAddress"/>
</Input>
<Output rdf:ID="Message">
 <hasType rdf:resource="http://www.bpOnto.org/order#Notificat"/>
```

[1] http://www.isi.edu/ pan/damltime/time-entry.owl

```
</Output>
<Activity rdf:ID="NOTIF">
 <hasInput rdf:resource="#orderNum"/>
 <hasInput rdf:resource="#Email"/>
 <hasOutput rdf:resource="#Message"/>
 <hasRest rdf:resource="#SplitXOR">
 <RsetOn rdf:resource="#notif2SenInvAndProd"/>
 <RestOn rdf:resource="#notif2SenInv"/>
</Activity>
<Transition rdf:ID="notif2SenInv">
 <TransFrom rdf:resource="NOTIF" />
 <TransTo rdf:resource="SendInvoice"/>
</Transition>
```

5 Conclusion and Perspectives

In this paper we presented a new approach for workflow inter-organizational cooperation, by proposing a workflow description combining *XPDL* and *OWL*. This description is semantic and preserve the privacy of partners. To elaborate this description, workflow is firstly described in *XPDL*. Then we identify the cooperative activities of workflow. Once the cooperative activities are identified, they are described using *OWL* according to an ontology we have defined for cooperative workflows. Our objective now within the *CoopFlow* approach is to develop a registry to which we can publish semantic descriptions of workflows to enable dynamic and automatic discovery of workflows. We are currently implementing a matchmaking algorithm based on graph similarity.

References

1. van der Aalst W.-M.-P., Weske, M.: The P2P Approach to Interorganizational Workflows. CAiSE (2001) 140-156
2. van der Aalst W.-M.-P.: Loosely Coupled Interorganizational Workflows: Modeling and Analyzing Workflows Crossing Organizational Boundaries. Information and Management Journal **37** (2000) 67-75
3. Zhao, J.-L.: Workflow Management in the Age of E-Business. The 35th HICSS. (2002)
4. Chebbi, I., Dustdar, S., Tata, S.: The view-based approach to dynamic inter-organizational workflow cooperation. Data Knowl. Eng. **56(2)** (2006) 139-173
5. Chebbi, I., Tata, S.: CoopFlow: A Framework for Inter-organizational Workflow Cooperation. OTM Conferences **(1)** (2005) 112-129
6. Kali, K., Tata, S.: Abstraction-based Workflow Cooperation Using Petri Net Theory. 14th IEEE WETICE. Sweden (2005)
7. T. Andrews and al.: Business Process Execution Language for Web Services. (2003)
8. Workflow Management Coalition. XML Process Definition Language. (2005)
9. Smith, M. K., Welty, C., McGuinness, D. L.: OWL Web Ontology Language Guide. (2004)
10. M. David and al.: Bringing Semantics to Web Services: The OWL-S Approach. Semantic Web Services and Web Process Composition Workshop. (2004) 26–42
11. van der Aalst W.-M.-P. : Interorganizational Workflows: An Approach based on Message Sequence Charts and Petri Nets. SAMS journal. (1999) **34(3)** 335–367
12. Feier, C., Domingue, J.: D3.1v0.1 WSMO Primer, Final Draft. DERI. 2005
13. J. De Bruijn and al.: Web Service Modeling Language, Final Draft. DERI. 2005