Enabling Pervasive Execution of Workflows

Frederic Montagut
SAP Labs France, Institut Eurecom
805, Avenue du Docteur Maurice Donat
Font de l'Orme, 06250 Mougins, France
frederic.montagut@sap.com

Refik Molva
Institut Eurecom
2229 Route des Crtes
06904 Sophia-Antipolis, France
refik.molva@eurecom.fr

Abstract

As opposed to existing workflow management systems, the distributed execution of workflows in pervasive environments can not rely on centralized control. The dynamic nature of the pervasive environments on the other hand does not allow a permanent assignment of roles to devices. In this paper, we suggest an architecture supporting distributed execution of workflows in pervasive environments based on a fully decentralized control and dynamic assignment of roles to devices. The architecture is defined in terms of data structures associated with the workflow execution plan and dynamic information, a protocol for the exchange of the workflow data among devices and a new function called role discovery dealing with the dynamic assignment of roles to devices. This architecture is applied in the context of Web services using BPEL as workflow description language.

1. Introduction

Pervasive environments feature complex computer applications that allow mobile users to access ambient services. These services range from residential temperature control to customer assistance in a shopping centre [14]. In most existing examples of pervasive services, the mobile end-users act only as clients of complex business processes executed by the ambient infrastructure. In this paper, we suggest a distributed computation paradigm whereby mobile users do not act merely as clients but can also actively participate in the execution of the overall services, providing their resources as it has become possible to run more and more complex applications on mobile devices [8, 5]. Using such a distributed execution environment, complex interoperations between mobile devices can therefore be envisioned as an extension of the classical workflow concept. The execution of a workflow by a set of devices in the pervasive environment raises new requirements such as the compliance of the overall sequence of operations with the pre-defined workflow execution plan. As opposed to existing workflow management systems, the distributed execution of workflows in pervasive environments can not rely on centralized control. In addition, the dynamic nature of the pervasive environments does not allow a permanent assignment of roles to devices.

In this paper, we suggest an architecture supporting distributed execution of workflows in pervasive environments based on a fully decentralized control and dynamic assignment of roles to devices. In section 2 the architecture is defined in terms of data structures associated with the workflow execution plan and dynamic information, a protocol for the exchange of the workflow data among devices and a new function called role discovery dealing with the dynamic assignment of roles to devices. Section 3 depicts a detailed application of the conceptual model based on Web services and an extension of the workflow description language BPEL. Section 4 discusses the related work and section 5 gives some concluding remarks.

2. Pervasive workflow architecture

The challenges raised by pervasive execution of workflows are due to the lack of a dedicated infrastructure to perform workflow management tasks. The main objectives of our architecture is to cope with the lack of dedicated management infrastructure through distributed mechanisms. We first give an overview of the execution environment. The architecture is then described in terms of the runtime and cross-organisational specifications. The architecture lifecycle is finally described.

2.1. Problem statement and definitions

The pervasive nature of the execution environment is characterized by two basic assumptions:

- distributed control: no single device is in charge of managing the workflow execution and the overall con-

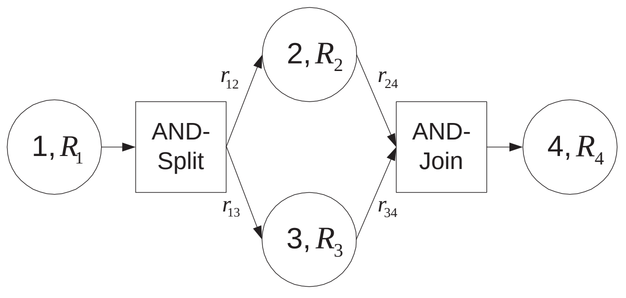

Figure 1. Process with two branches

trol is assured through the collaboration of the devices involved in the execution of the workflow.

- dynamic task assignment: the devices that will be executing an instance of a given workflow are not known in advance.

These two assumptions are basically the challenges our architecture design will need to cope with.

Our second assumption raises the need of a service to assure discovery of devices executing a workflow instance. In this paper, we focus on the workflow execution support, the main features of this service discovery are therefore detailed but its specification is out of the scope of this paper. We define two cases: source discovery enabling discovery of all devices at workflow initialization phase and runtime discovery enabling devices' discovery along with a workflow instance.

A pervasive workflow W is represented using a directed graph. In the graphical workflow model, each vertex represents all workflow actions contiguously performed by a device, whereas the edges represent the sequence of workflow steps among devices. We distinguish the following activities associated with the graph definition:

- t_{in}: task activities consisting of actions performed by devices participating in the workflow. Each task activity is locally executed by a given device and a vertex n represents a set of task activities locally executed on the same device. As a result, each vertex is linked to a single device and two neighbour vertices in the graph are linked to two different devices. Each task activity is identified by an index i and the index n of the vertex where it is executed.

- r_{ij}: routing activities used to transfer control and data between devices. These are the graph edges and represent the data exchanged between the devices linked to two consecutive vertices during the execution of a workflow instance. i and j are the indexes of the vertices linked by this edge.

- f: process control activities describing execution schemes and dependencies between routing and task activities. We consider three types of dependencies as they are defined in [1]:
  - Sequential execution: abstract interpretation of the directed graph
  - AND-Split/AND-JOIN: branches of routing and task activities executed in parallel
  - OR-Split/OR-JOIN: branches based on conditional choice

W is therefore a finite set of t_{in}, r_{ij} and f. In addition, we also consider the notion of role derived from usual workflows. A role R_i is defined as a set of attributes necessary to execute the task activities associated to a vertex. As we stated before, we assume that the execution of a workflow instance is not a priori assigned to any devices or users, they are discovered at runtime. Our workflow model therefore relies on the notion of role defining an abstract entity whereas usual workflows rely on the notion of business partners which are identified persons or organisations. In a first approach, we consider each role consists of two categories of attributes:

- Security credentials assuring the right to execute (user perspective) on the one hand
- Proof of compliance with the workflow task activities assuring the capability to execute the required tasks (device perspective) on the other hand

We associate this notion of role with the discovery service described above. The role discovery service is in charge of assigning devices/users owning the proper role for the execution of task activities at a given vertex.

We can for instance model a simple process with two branches of sequential activities using the graph depicted in Figure 1. The device executing the task activities of vertex 1 sends data to the devices in charge of vertices 2 and 3 (r_{12} and r_{13}). Vertices 2 and 3 are executed in parallel and the device executing vertex 4, which has to verify role 4, gathers the results. In our approach, we consider simple workflow scenarios where no synchronisation issues are dealt with, branches are executed in parallel without any constraints. We also do not focus neither on fault handling nor compensation mechanisms.

2.2. Runtime specifications

The runtime specifications of our architecture are studied. We first specify the components of the distributed workflow management system, the appropriate messaging protocols ensuring the proper deployment and execution of the workflow within this Workflow management system are then examined.

2.2.1. Distributed workflow management system. Considering a distributed setting which does not rely on any dedicated workflow management system implies that the workflow management task is distributed among all involved devices. As a result, a part of the workflow management infrastructure is also deployed on each device. A workflow engine needs therefore to be implemented on each device participating to the workflow. A workflow engine is a software in charge of interpreting the representation of a workflow using a computational form also called workflow description language. Each workflow engine is in charge of managing locally on the devices the execution of the workflow. With respect to the representation depicted above it basically consists in the following sequence of tasks:

- Reception of all incoming routing activities
- Execution on the device the required task activities corresponding to the current vertex.
- Sending a routing activity to the next devices

Due to availability issues introduced by the pervasive environment, we choose a stateless model. A single vertex can only be linked with a device at a time, meaning that after the execution of a given vertex, no residual information are stored on a device, they are all transferred to the next one. In the previous section, we defined the notion of pervasive workflow W which describes the complete workflow execution pattern. At the workflow engine level, we need to specify the part W_n of W locally executed on each device for each vertex n. W_n of a vertex n therefore consists of:

- The task activities t_{in} associated to the vertex n,
- The routing activities r_{in} arriving and r_{nj} leaving the vertex n,
- The set of process control activities f describing the dependencies between the considered t_{in}, r_{in} and r_{nj}.

The complete execution of W is therefore composed of the execution of the different W_n by the devices' local workflow engines. We need now to specify the messaging protocols used in the communications between the different workflow engines to allow a coherent execution of the W_n with respect to the complete workflow description W.

2.2.2. Messaging protocols in the architecture. The communications between the devices are managed at the workflow engine level according to the workflow specifications as depicted in Figure 2. The messaging between the engines refers to the routing activities defined above. Each routing activities specifies the information required to enable a coherent workflow execution. The main objective of the messaging within the architecture is first to allow a device which does not have any a priori knowledge about a running instance of a workflow to be part of it and execute what it is requested to at a given vertex. This device should then be able to forward operation requests to the appropriate recipients which are next in the workflow execution. The objectives of the information carried by a single message are therefore as follows:

1. Identify the vertex n of the workflow instance executed to be able to determine the current step of the execution
2. Retrieve the W_n associated to the considered vertex to actually know what to execute

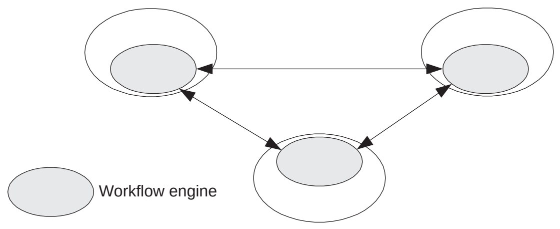

Figure 2. distributed WfMs

3. Identify the workflow instance associated to the message received (for convergence point of branches)
4. Retrieve the identity of the devices associated to a role to keep track of devices involved in the workflow
5. Retrieve the workflow data to modify the state of the workflow at each execution step

As a result, we consider the following parameters in the messaging format for the routing activities:

- W: complete workflow definition, which is a representation of the directed graph defined in 2.1
- Iid: Instance identifier, unique id to represent a given instance of a workflow. The Id is composed of random number plus the workflow initiator identity (IP address)
- Workflow vertex number n that the message recipient has to execute
- Map: Mapping table between roles and devices. This table is updated with respect to the discovery mode used; it can either be complete at the workflow initialisation phase with source discovery or updated at each workflow vertex with runtime discovery
- Data: This includes the three data types defined in [1]: control data, relevant data and application data.

Using such a messaging format allows a device upon reception of a request to cope with the five defined objectives:

- n is included in the message,
- W_n is derived from W with the knowledge of n,
- The workflow instance is uniquely identified by Iid
- The association roles to devices is stored in Map
- Workflow data are transported

2.3. Cross organizational aspects

The mobile devices involved in a workflow instance are an extension of the organisation they belong to. They provide their contribution with respect to their available mobile applications and are also in charge of managing their participation to the workflow execution. Each device therefore consists of two parts: a set of mobile applications supported by the device and an embedded workflow engine. The mobile applications of each device need to be kept private, they

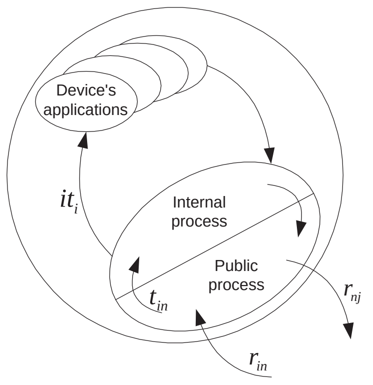

Figure 3. Mobile device representation

can only be invoked by the device itself. The workflow engine needs therefore to act like a proxy between these applications and the other devices of the collaboration. In this section, we detail how the access to these mobile applications through the workflow engine is performed.

2.3.1. Device representation. To cope with the requirements introduced by cross organisational collaboration workflows, we adopt the representation of "private-public" processes introduced in [15]. As a result, the embedded workflow engine acquires mainly two roles: managing the device internal applications and coordinating the execution of W_n. Thus, two processes corresponding to these roles are defined within each device:

- The public process is the subset W_n specified in 2.2.1. This process definition includes the activities the device is asked to perform during a workflow instance.
- The private process is an internal one developed by the device user himself with respect to his device's embedded applications. The internal process of device k is noted IP_k. This process is not considered to be distributed but coordinated in a centralized manner.

We therefore consider a hierarchical model with two levels to represent the task activities the devices will have to perform in our workflow model. The first level corresponds to the public definition of the workflow provided by the global process definition W and the subsets W_n derived from it. At this first level, these task activities are called t_{in} and their specifications are known by all involved devices. We consider that these public task activities are themselves processes involving different steps to be completed. The atomic definitions of these public t_{in} are specified within the private processes implemented on each device; this constitutes the second level. The atomic definition of a given task activity is thus only known by the device in charge of executing this specific task. The link between these two levels is realized through communications between public and private processes.

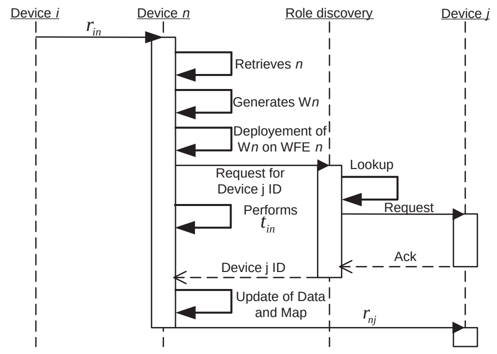

Figure 4. Architecture sequence diagram

2.3.2. Private, public processes link specification. To represent the internal process of a given device, we adopt the same notation as the one defined in 2.2. The internal process of device k IP_k is as a result a set of internal activities:

- Internal task activities it_i specifying the atomic tasks performed by the device
- Process control activities f specifying the execution schemes of the internal task activities.

This process is indeed centralized, routing activities are thus not defined. Using this notation, we can define a public task activity t_{in} as a subset T_i of IP_k composed of it_i and f. The internal process IP_k of a device k is therefore divided into different subsets T_i corresponding to the public task activities t_{in} defined in W_n. The internal communications resulting from this description are depicted in Figure 3. Whenever the mobile device needs to perform a task activity t_{in} after having received a routing activity r_{in}, the public process forwards the request to the internal process which executes the subset T_i corresponding to t_{in}. The local applications required to achieve that subset are thus invoked. The result is transferred to the other devices via the public process by the routing activity r_{nj}.

2.4. Complete architecture mechanisms

So far we detailed the behavioural characteristics of the elements part of our pervasive workflow architecture. In this section, we present how these elements work together and the basic mechanisms of this architecture. Figure 4 presents a sequence diagram depicting the interactions between three devices in charge of executing three vertices i, n and j and the ambient role discovery service. We assume a runtime discovery scenario. Upon reception of routing activity r_{in}, device n first retrieves n and generates the W_n associated to the vertex it has to execute. This process is then deployed on its local workflow engine and executed. In the meantime, it makes a request to the role discovery service in order to get the Id of the device that will execute vertex j. Once W_n is completed and device j's Id is received, Data and Map fields are updated, the next routing activity r_{nj} is sent.

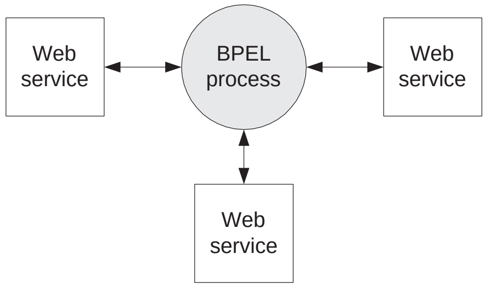

Figure 5. BPEL centralized paradigm

3. Web services application

Section 2 presented the characteristics of the pervasive workflow architecture we designed. We now present its application using Web services technology whose composition properties are appropriate for distributed workflow applications. Web services technology makes use of BPEL [3], workflow description language and WSDL [10] standard to enable the composition of services within workflows. This section is organised as follows. In the first part, we recall the basis of usual BPEL processes. We then give an overview of the infrastructure components associated to our architecture in the context of Web services and finally analyse the deployment of the architectural mechanisms.

3.1. Basis of BPEL

BPEL processes mainly involve two types of entities as depicted in Figure 5: the central BPEL process and the Web services part of its execution. The BPEL process is locally executed on a BPEL engine which is itself a Web service. BPEL processes are therefore strongly linked to WSDL standard used to describe Web services functionalities and how to communicate with them. In this perspective, we present the role of WSDL before giving details on BPEL.

3.1.1. Role of WSDL. WSDL is used to describe how to communicate with a given Web service. Basically it includes the definition of available operations, variable formats, service uri and messaging formats. In the context of BPEL, the following WSDL parameters are used: operation, portType and partnerLinkType. A WSDL portType defines a set of operations that can be invoked on the service. The partnerlinkType parameter is used to describe in case of BPEL processes a communication channel between two Web services. It associates a portType of each service to form a channel composed of two unidirectional links. Within it, each service is assigned to a unique role and can send information to the other using the latter's specified portType. BPEL processes are built on top of the WSDL definitions of all involved Web services.

3.1.2. BPEL processes. BPEL is an XML-based workflow description language coordinating Web services in a centralized perspective. As depicted in Figure 5, Web services part of a BPEL workflow indeed only have interactions with the BPEL engine in charge of managing and interpreting the whole process execution. The interactions within the process are described in terms of partnerLink which associates two Web services in a WSDL partnerlinkType by assigning the partnerlinkType roles to them. BPEL language uses a set of activities to represent the workflow among which are `<Invoke>` (Web service invocation), `<Receive>` (waiting state), `<Sequence>` (sequential execution), `<Flow>` (concurrent execution), `<Switch>` (conditional scheme), `<While>` (iterative process), `<Pick>` (conditional scheme based on external events).

3.2. Infrastructure components

In the architecture definition we analysed the requirements on the workflow management system to enable the execution of our pervasive workflow model. Our conclusion was the deployment on each device of workflow engines. In the case of Web services, these workflow engines are BPEL enabled and communicate via a Web service interface. Each device executing a vertex of a workflow instance will have to support this infrastructure. Figure 6 identifies the infrastructure components present on devices taking part in a pervasive workflow instance execution.

Two BPEL processes are deployed on the BPEL engine, a BPEL public process W_n and an internal one IP_k. Each BPEL process has its own WSDL description. The public process is contacted via routing activities; the public process WSDL therefore includes the specifications of routing activities as WSDL operations. Yet routing activities are the only one used to contact the public BPEL process, we also need to advertise the device's functionalities i.e. the task activities it is able to execute so that it can be identified as a potential actor of a workflow instance during the discovery process. We choose to represent these functionalities using the Web service ontology OWL-S [4]. This OWL-S specification will indeed help a role discovery service to perform the selection of the devices executing the workflow. Part of this selection process will be based on what is required by a workflow in terms of task activities, and what available devices advertise in their OWL-S specification. The details of the OWL-S discovery system are not in the scope of this study, [2, 11] are examples of such semantic matching services. Integrating an ontological definition in our infrastructure assumes that each potential actor of a workflow shares a common ontology. The private process only receives requests from the public process via task activities. The specification as WSDL operations of these task activities need thus to be part of the private process WSDL definition. By definition the private process is kept private

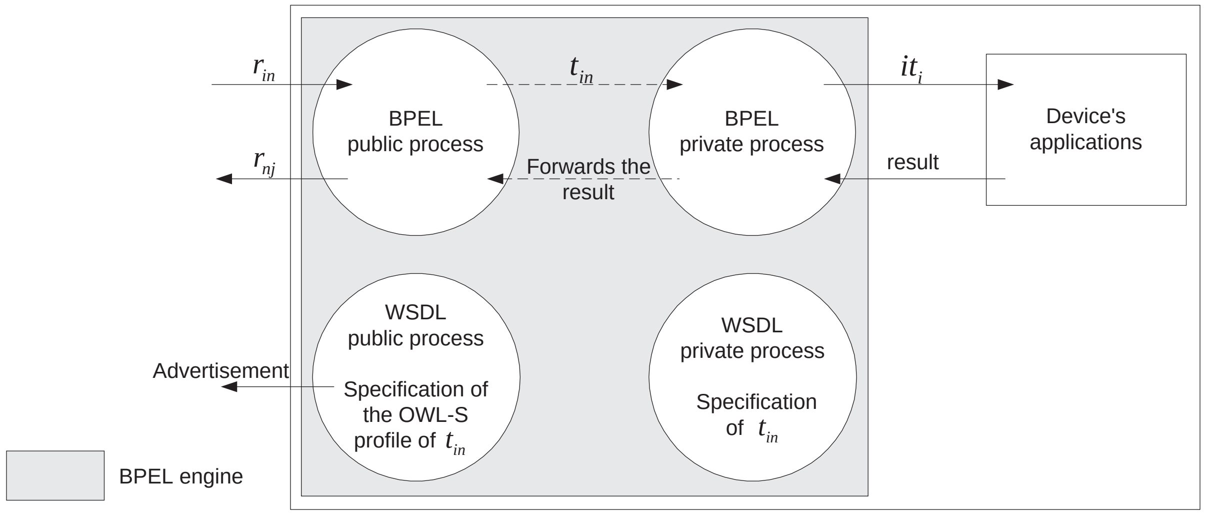

Figure 6. Web services based device infrastructure

and is only available to his owner. What is here kept private is the atomic definition of the task activities part of the public process, i.e. how they are locally executed by a device.

Now that we have a clear idea of the components part of the Web services infrastructure, we need to specify the deployment of runtime mechanisms exposed in our architecture. This includes first the computational representation of W to enable a device to retrieve W_n expressed in BPEL to execute a given vertex n. The specification of W has also to enable the retrieval of valid WSDL specifications deployed with the W_n BPEL process itself. We also need to precise the specification of the link between public and private process using BPEL.

3.3. Specification of W

The specification of W is a crucial point in the infrastructure since it is used by all involved devices to know what to execute. We want to represent pervasive workflows W using BPEL characteristics. We thus need to define an extension of both WSDL and BPEL associated to our pervasive workflow representation. The main ideas of the extension of BPEL we define are first presented; we then specify the extension including WSDL and BPEL definitions.

3.3.1. Process representation. Our goal is to use BPEL activities to represent a distributed workflow W depicted by a directed graph. We first translate the t_{in}, r_{ij} and f defined in section 2 into BPEL activities:

- t_{in}: These are `<invoke>` activities invoking the BPEL private process of a device.
- r_{ij}: These are also `<invoke>` activities, the device executing vertex n invokes the device executing the next vertex with the required workflow data.
- f: These are `<flow>`, `<sequence>`, activities.

We use the concepts of BPEL processes and map them to the directed graph approach of section 2 in order to define a workflow description language capable to completely specify pervasive workflows W. We call this representation distBEPL standing for distributed BPEL. The goal of this extension is therefore to represent distributed processes from a global perspective whereas BPEL is currently limited to depict processes from a centralized perspective. BPEL control activities are used to represent graph dependencies. For instance, `<flow>` construct is used to represent AND-Split, AND-Join; `<sequence>` construct represents sequential execution. Figure 1 can therefore be represented by the following XML representation. We assume that each vertex of this process implements one task activities.

```
<sequence>
  <t11>
  <flow>
    <sequence>
      <r12>
      <t21>
      <r24>
    </sequence>
    <sequence>
      <r13>
      <t31>
      <r34>
    </sequence>
  </flow>
  <t41>
</sequence>
```

The task activity of vertex 1 is first executed; a parallel execution of two sequence branches is then performed.

3.3.2. Defining WSDL partnerLinkTypes and OWL-S profiles. Deploying a BPEL process includes the deployment of the WSDL definition of the Web services involved in the process as well as the partnerLinkType definition. In our architecture, we want to enable execution of pervasive processes whose participants are discovered at runtime. The WSDL specification of the involved Web services will as a result only be known when the latter are identified. Yet, our service discovery procedure includes the verification of the capability of a device to actually execute all required task activities for a given vertex n. This capability matching procedure uses OWL-S ontology as we stated in 2.2.1 and more specifically the OWL-S service profile. We choose therefore to associate each vertex n to a OWL-S profile specifying the task activities of n. This complete description will be transported as part of our distBPEL representation of W. The other crucial point is also to include as part of W a WSDL representation of the partnerLinkType. Defining partnerLinkType supposes that portType are already specified. In our case, the communications between devices are done with the routing activities. As stated above, routing activities are simple invocations and we call generically jreceive the operation invoked on the device executing vertex j, the portType defining jreceive is called jportType. Each graph edge is associated to a partnerLinkType in a WSDL file part of W with the form[1]:

```
<pLT name="nj">
   <role name="ji">
     <pT="jportType"/>
   </role>
</pLT>
```

where n and j are the vertices number linked by the graph edge, i is the role number associated to vertex j. Each link is mapped to an unidirectional channel since communications between two vertices are unidirectional. This link states that vertex n sends information on jportType of vertex j.

3.3.3. DistBPEL representation. Now that the WSDL and OWL-S details part of W are defined, distBPEL routing and task activities can be specified. The first parameters we need to precise are the partnerLink. We use unidirectional links in partnerLinkTypes, BPEL partnerLink are therefore also unidirectional and a single role is assigned. We use the following representation for partnerLink:

```
<pL name="nj" pLT="nj" jRole="ji"/>
```

where n and j are the vertices number linked by the graph edge, i is the role number associated to vertex j. In the distBPEL partnerLink notation, we modify the usual BPEL my/partnerRole parameter into a jRole. This transformation expresses the shifting between a centralized to a global point-of-view. We are now able to specify the representation of routing and task activities in distBPEL. These two activities are basic standard BPEL `<invoke>` activities. A routing activity r_{ij} has the following form:

```
<invoke pL="nj" pT="jportType" operation="jreceive"/>
```

[1]partnerLinkType, partnerLink and portType are respectively denoted in our code subsets: pLT, pL and pT

with the same notation as defined above. The complete specification of a task activity t_{in} will be depicted later on since it is internal to a device. Yet we adopt the following representation in case of a signature operation for example:

```
<invoke pL="ninternal" pT="nportType"
 operation="signature"/>
```

This mainly translates the idea of a task executed internally since the partnerLink and portType associated to the operation signature are not known by others.

3.3.4. Example of process representation in distBPEL. We provide a complex example of a process representation using distBPEL of Figure 7. For the sake of simplicity, we only focus on the abstract process definition. We consider a process with parallel execution of two branches between activities and dependencies between the branches. The workflow proposes two parallel branches of activities executed in sequence, the process control activities `<sequence>` and `<flow>` are thus used. This example is particular due to the routing activity r_{25} which links two vertices from different branches. To have a coherent and consistent process definition, this activity needs to be specified twice in the distBPEL process definition. The computational representation of the directed graph has indeed to take into account all dependencies between routing and task activities. In that specific case, r_{25} has to be executed in parallel with r_{23} but has also to be received by vertex 5 before executing t_{15}. This activity appears as a result twice in the process representation.

```
<sequence>
  <invoke pL="1internal" pT="1portType" operation="11"/>
<flow>
<sequence>
  <invoke pL="12" pT="2portType" operation="2receive"/>
  <invoke pL="2internal" pT="2portType" operation="12"/>
<flow>
  <invoke pL="23" pT="3portType" operation="3receive"/>
  <invoke pL="25" pT="5portType" operation="5receive"/>
</flow>
  <invoke pL="3internal" pT="3portType" operation="13"/>
  <invoke pL="36" pT="6portType" operation="6receive"/>
</sequence>
<sequence>
  <invoke pL="14" pT="4portType" operation="4receive"/>
  <invoke pL="4internal" pT="4portType" operation="14"/>
<flow>
  <invoke pL="45" pT="5portType" operation="5receive"/>
  <invoke pL="25" pT="5portType" operation="5receive"/>
</flow>
  <invoke pL="5internal" pT="5portType" operation="15"/>
  <invoke pL="56" pT="6portType" operation="6receive"/>
</sequence>
</flow>
  <invoke pL="6internal" pT="6portType" operation="16"/>
</sequence>
```

3.3.5. Components of W. We finally sum up what we detail above. The specification of W in our Web services infrastructure is composed of:

- The representation of the workflow directed graph in distBPEL
- The OWL-S specification the task activities executed for each workflow vertex n
- The WSDL specification of the partnerLinkType representing the r_{ij}

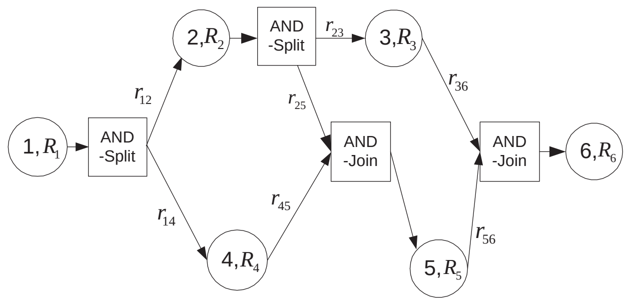

Figure 7. Complex Business process

3.4. Deriving W_n

As we stated in our architecture, the execution of a vertex n described in W is linked to its local description W_n. We depicted in the last section a way to represent W using an extension of BPEL, distBPEL. Based on this representation, we now present an algorithm enabling the retrieval of W_n associated to a vertex n with the only knowledge of n. The deployment of W_n includes both WSDL and BPEL specifications. The WSDL definitions to be deployed are the ones from the device's public and private processes, the WSDL representation of the next devices executing the next nodes and the WSDL specification of the partnerLinkTypes. The involved devices provide their WSDL representations, here we therefore only focus on the BPEL definition of W_n. The procedure consists of five steps which are now detailed. We consider the generation of local BPEL specification for vertex 5 in the workflow depicted in Figure 7.

Step 1: Specification of coherent partnerLinks. In distBPEL, we specify the partnerLink section from a global point-of-view, when considering W_n, we focus on a vertex n. The distBPEL partnerLinks are therefore modified accordingly. Here we consider two cases: either the distBPEL partnerLink represents an incoming link in which case, the "nRole" becomes "myRole", or it represents an out coming link and the "nRole" becomes "partnerRole". For instance in the case of vertex 5 we get (only partnerLinks involving vertex 5 are kept in the declaration):

```
<partnerLinks>
 <pL name="45" pLT="45" myRole="55"/>
 <pL name="25" pLT="25" myRole="55"/>
 <pL name="56" pLT="56" partnerRole="66"/>
</partnerLinks>
```

Step 2: Focus on the considered vertex. The input of this procedure is the global process description specified in section 3.3.4. We want to derive from this input the local vision of vertex 5 and only consider the incoming and out coming routing activities as well as the task activities performed at vertex 5. The first step of our procedure is therefore to focus on those specific activities and delete the activities in which vertex 5 is not involved. The other aspect is also to keep the dependencies specified by the process control activities. According to our procedure, r_{25}, r_{45}, t_{15}, r_{56} are only kept:

```
<sequence>
<flow>
<sequence>
<flow>
 <invoke pL="25" pT="5portType" operation="5receive"/>
</flow>
</sequence>
<sequence>
<flow>
 <invoke pL="45" pT="5portType" operation="5receive"/>
 <invoke pL="25" pT="5portType" operation="5receive"/>
</flow>
 <invoke pL="5internal" pT="5portType" operation="15"/>
 <invoke pL="56" pT="6portType" operation="6receive"/>
</sequence>
</flow>
</sequence>
```

Step 3: Simplification of redundant routing activities. As we stated in the description of distBPEL, all dependencies between activities need to be specified in a distBPEL process description. As a result, in case of dependencies between concurrent branches, activities creating these dependencies are specified twice. When focusing on a single vertex, we only keep the instance of a redundant activity that creates dependencies on the activities concerning the vertex. In our example, the second instance of r_{25} introduces dependencies (sequential dependency) with the other activities of vertex 5. The first instance is thus erased:

```
<sequence>
<flow>
<sequence>
<flow>
</flow>
</sequence>
<sequence>
<flow>
 <invoke pL="45" pT="5portType" operation="5receive"/>
 <invoke pL="25" pT="5portType" operation="5receive"/>
</flow>
 <invoke pL="5internal" pT="5portType" operation="15"/>
 <invoke pL="56" pT="6portType" operation="6receive"/>
</sequence>
</flow>
</sequence>
```

Step 4: Process control activities simplification. So far, we restricted the global process description to the activities involving a given vertex. This created inconsistency with the process control activities. A process control activity indeed specifies an execution scheme between at least two routing or task activities. Process control activities which do not verify this property are deleted from the specification. To do so, we perform an iterative procedure till all process control activities properly specify an execution scheme of at least two activities:

```
<sequence>
<flow>
 <invoke pL="45" pT="5portType" operation="5receive"/>
 <invoke pL="25" pT="5portType" operation="5receive"/>
</flow>
 <invoke pL="5internal" pT="5portType" operation="15"/>
 <invoke pL="56" pT="6portType" operation="6receive"/>
</sequence>
```

Step 5: Translation into standard BPEL. The last step now consists in retrieving a standard BPEL specification of the vertex local code. In distBPEL we only specified `<invoke>` type activities to model the process steps. In the code resulting of the third step, only remain either incoming or out coming routing activities and the vertex task activities. The task activities are let here untouched since they are internal tasks and has to be specified by the device executing the vertex. The incoming routing activities have to be modified. From the vertex point-of-view, these activities correspond to a waiting state. This is expressed in BPEL by the

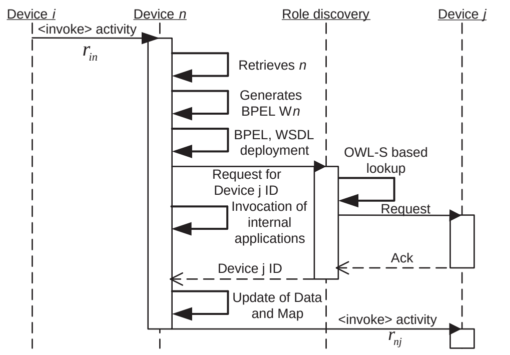

Figure 8. Infrastructure sequence diagram

activity `<receive>`. Incoming routing activities are therefore transformed into `<receive>` activities with the same partnerLink, portType and operation. Out coming routing activities are on the contrary let untouched. We finally get this BPEL code:

```
<sequence>
<flow>
 <receive pL="45" pT="5portType" operation="5receive"/>
 <receive pL="25" pT="5portType" operation="5receive">
</flow>
 <invoke pL="5internal" pT="5portType" operation="15">
 <invoke pL="56" pT="6portType" operation="6receive">
</sequence>
```

The BPEL specification obtained at step 5 is still an abstract representation. Prior to the final deployment on the BPEL engine, still remains the specification for the different inputs and outputs of the operations invoked on the internal process. These are added using the WSDL specification provided by the latter.

3.5. Cross-organisational aspects in BPEL

As we defined in the architecture definition, the execution of the task activities associated to a workflow vertex is performed by a device on which are deployed two BPEL processes, a public one and a private one. In this section, we detail the communications between the private and public in the case of BPEL. The public process manages the participation of a device in the global workflow while the internal process manages the execution of the task activities assigned to the device. We first provide a template for the internal process specification and then specify the task activity which is the link between the two processes.

3.5.1. Internal process specification in BPEL. As part of our architecture each device owns an internal process managing its embedded applications. This process is defined by the device user and describes the operations the device is able to perform. It is only invoked inside the device by the public process. Hence, the internal process definition is organised to assure this role and divided into subsets T_i as we defined them in section 2.3.1. The process control function `<pick>` of BPEL enabling to make choices based on external events is therefore used to represent this division. In that case the external events are the incoming task activities t_{in} made by the public process and asking to perform specific operations. The BPEL code is organized in different sections delimited by `<onMessage>` tags. Each section corresponds to a specific subset T_i and is activated based on incoming task activities.

```
<process name="internalprocess">
<pick createInstance="yes">
 <onMessage pL="publicprivateLink" pT="IPport"
  operation="signature192" variable="input">
          <invoke.../>
 </onMessage>
 <onMessage pL="publicprivateLink" pT="IPport"
  operation="encryption192" variable="input">
          <invoke.../>
 </onMessage>
...
</pick>
...
</process>
```

3.5.2. Task activity specification. The specification of W provides a general description on what a task activity consists in, its atomic definition has to be specified at runtime when a device asked to perform vertex n generates W_n. It is indeed the link between this general description and its associated subset T_i of the internal process. The task activity is an `<invoke>` activity including portType, partnerLink and operation proper to the internal process of a device. We provide an example of task activity definition linked to the internal process we depict above. Upon reception of r_{12}, the device is asked to perform a signature. The signature192 operation is invoked on the internal process with t_{12} defined in the public process. Internally, the private process then executes the section corresponding to the `<onMessage>` part where operation signature192 is mentioned.

```
<sequence name="processsequence">
 <receive name= "r12" pL="12"pT="2portType"
  operation="2receive" variable="input"/>
 <invoke name="t12" pL="publicprivateLink" pT="IPport"
  operation="signature192" inputVariable="input"
  outputVariable="output"/>
 <invoke name="r23" pL="23" pT="3portType"
  operation="3receive" inputVariable="input"/>
...
</sequence>
```

3.5.3. Execution scheme of a distributed workflow in the infrastructure. In section 2.4 we provide a sequence diagram of the conceptual architecture, we detail in this section the mapping with the concepts we have just presented. Figure 8 depicts the sequence diagram associated to the infrastructure:

- Routing activities are specified as BPEL invocations,
- The generation of the W_n is performed with the procedure of section 3.4
- The discovery procedure is based on the comparison between the OWL-S definition of W and the one provided by the devices

4. Related work

The areas of research tackled by this work include workflow execution in pervasive environments, semantic composition of Web services and distributed execution of workflows.

Most of the research in the area of workflow execution in pervasive environments concerns the interaction between end-users and the process. In [14] a workflow-based architecture to help mobile users of unfamiliar pervasive environments is proposed. Human interactions within BPEL processes are studied in [9] where an architecture adapting to workflow users' mobility is introduced. These architectures still consider workflow end-users with a low level of implication within the workflow execution managed by the ambient infrastructure. In comparison, we propose a self-managing architecture enabling a completely distributed workflow execution. Our architecture allows running more complex pervasive collaborations between mobile workers.

Despite quite extensive work on semantic composition of Web services falls in one of two approaches: assignment of Web services instances at process design time [6] or at runtime [12]. Our architecture addresses the runtime approach and adds to existing solutions a stateless distributed environment which allows more flexibility for the execution.

Several pieces of work provide an architecture for distributed execution of workflows yet they all suffer from a static design and far from meeting the requirements of pervasive environments. In [13] a decentralized orchestration scheme for Web services is proposed. Starting from a centralized BPEL process specification, a distributed one is produced and deployed on already designated Web services. Self-Serv [7] implements a distributed orchestration platform for composite Web services. Both of these solutions propose distributed composition of Web services chosen at the process designing phase. Our solution on the contrary introduces an extension of BPEL language interpreting distributed processes to allow a runtime discovery of involved Web services.

5. Conclusion

We presented an architecture and its associated Web services infrastructure to support the execution of workflows in pervasive environments. Our solution allows the execution of completely distributed business processes where business partners can be discovered at runtime. We believe that thanks to the fully decentralized control that is the underpinnings of the execution model, this architecture will foster the development of new business cases suited to pervasive environments. Lack of centralized control on the other hand raises serious concerns about security and fault recovery. Our future work will thus be twofold. On the one hand, it will focus on security requirements brought up by this architecture and the design of solutions. On the other hand, fault handling mechanisms will be examined.

6. Acknowledgements

This work has been partially sponsored by EU IST-Directorate General as a part of FP6 IST project MOSQUITO and by SAP Labs France S.A.S.

References

[1] The workflow reference model. Technical Report WFMC-TC-1003, Workflow Management Coalition, 1995.

[2] Parameterized semantic matching for workflow composition. Technical Report RC23133, IBM, 2004.

[3] Business process execution language for web sevices, http://www.ibm.com/developerworks/library/ws-bpel/.

[4] Owl-s specifications, http://www.daml.org/services 2003.

[5] Web services tool kit for mobile devices, http://www.alphaworks.ibm.com/tech/wstkmd 2002.

[6] V. Agarwal, K. Dasgupta, N.Karnik, A. Kumar, A. Kundu, S. Mittal, and B. Srivastava. A service creation environment based on end to end composition of web services. In *Proceedings of the WWW conference*, 2005.

[7] B. Benatallah, M. Dumas, and Q. Z. Sheng. Facilitating the rapid development and scalable orchestration of composite web services. *Distributed and Parallel Databases*, 17(1):5–37, 2005.

[8] S. Berger, S. McFaddin, C. Narayanaswami, and M. Raghunath. Web services on mobile devices - implementation and experience. In *Fifth IEEE Workshop on Mobile Computing Systems and Applications*, 2003.

[9] D. Chakraborty and H. Lei. Pervasive enablement of business processes. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications 2004. PerCom 2004*, pages 87– 97, March 2004.

[10] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1, http://www.w3.org/TR/wsdl 2001.

[11] M. Laukkanen and H. Helin. Composing workflows of semantic web services. In *Workshop on Web Services and Agent-based Engineering*, 2003.

[12] D. J. Mandell and S. A. McIlraith. Adapting bpel4ws for the semantic web: The bottom-up approach to web service interoperation. In *Proceedings of International Semantic Web Conference*, October 2003.

[13] M. G. Nanda, S. Chandra, and V. Sarkar. Decentralizing composite web services. In *Proceedings of Workshop on Compilers for Parallel Computing*, January 2003.

[14] A. Ranganathan and S. McFaddin. Using workflows to coordinate web services in pervasive computing environments. In *Proceedings of the IEEE International Conference on Web Services 2004*, pages 288–295, July 2004.

[15] K. A. Schulz and M. E. Orlowska. Architectural issues for cross-organisational b2b interactions. In *21st International Conference on Distributed Computing Systems Workshops (ICDCSW '01)*, pages 79–87, 2001.